Matchmaking multicritère basé sur une différence sémantique pour la logique de description EL. Application à la recommandation de documents en ligne dans le domaine de la métrologie.

Axel Mascaro

Matchmaking multicritère basé sur une différence sémantique pour la logique de description $\mathcal{EL}$.

Application à la recommandation de documents en ligne dans le domaine de la métrologie.

THÈSE

présentée et soutenue publiquement le 15 Décembre 2023

pour l'obtention du

Doctorat de l'Université Clermont Auvergne

(mention informatique)

par

Axel Mascaro

Composition du jury

Président :	Antoine Zimmermann	Professeur à l'École des Mines de Saint-Etienne, LIMOS
Rapporteurs :	Salima Benbernou	Professeure à l'Université Paris Cité, LIPADE
	Chan le Duc	Professeur à l'Université Sorbonne Paris Nord, LIMICS
Examinatrice :	Catherine Roussey	Chargée de recherche INRAE Centre Occitanie Montpellier, UMR MISTEA
Directeur :	Farouk Toumani	Professeur à l'Université Clermont Auvergne, LIMOS
Co-Encadrant :	Christophe Rey	Maître de conférences à l'Université Clermont Auvergne, LIMOS

Mis en page avec la classe thesul.

Remerciements

Je remercie tout d'abord le professeur Farouk Toumani, mon directeur de thèse et le maitre de conférences Christophe Rey, du LIMOS, mon co-encadrant de thèse, pour leur accompagnement sans faille tout au long de cette aventure.

Je remercie la chargée de Recherche Catherine Roussey de l'UMR MISTEA, INRAE Centre Occitanie Montpellier, pour sa participation à mon comité de thèse et ses remarques pertinentes sur mon travail.

Je remercie toutes les personnes m'ayant soutenu au sein du laboratoire du LIMOS, et notamment le professeur Sebastien Salva et le professeur Engelbert Mephu Nguifo, pour leurs précieux conseils.

Je remercie tous les acteurs du projet STAM et particulièrement Richard Degenne et Thomas Chaptal de Perfect Memory pour leur implication dans mon travail.

Je remercie le professeur Chan Le Duc, de LIMICS Bobigny, Université Sorbonne Paris, la maitre de conferences Anni-Yasmin Turhan de l'université de Dresden le professeur Steffen Staab de l'université de Stuttgart et la professeur Claudia Schon de l'université de Koblenz-Landau pour leurs commentaires sur mon travail et les références qu'ils m'ont proposées en rapport avec celui-ci.

Enfin je remercie le FEDER pour le financement de ces recherches.

Table des matières

Table des figures

Glossaire

Notation	Définition
$Score_{Dem}(Rep)$	Score de la réponse *Rep* par rapport à une demande utilisateur *Dem* dans un problème de matchmaking.
$\mathbf{C}$	Ensemble des noms de concept d'une TBox.
$\mathbf{r}$	Ensemble des noms de rôle d'une TBox.
$\mathsf{size}(C)$	Taille d'un concept.
$\mathsf{rd}(C)$	Profondeur d'un concept.
$\mathsf{wid}(C)$	Profondeur d'un concept.
$\mathsf{sig}_{\mathcal{T}}$	Signature d'une TBox.
$\mathcal{T}_{\mathcal{EL}}$	Ensemble de tous les $\mathcal{EL}$ concepts qui peuvent être construits à partir de $\mathsf{sig}_{\mathcal{T}}$ et $\top$.
$\mathcal{T}_{Bnf}$	TBox normalisée selon Baader et al. (2005, 2017).
$Cl_{\mathcal{T}_{Bnf}}$	Classification de la TBox $\mathcal{T}_{Bnf}$.
$\mathbf{S}(A)$	Ensemble des concepts qui subsumment A dans une classification.
$\mathbf{R}(r)$	Ensemble des couples (A, B) de concept tel que $A \sqsubseteq \exists r.B$.
$\mathcal{T}^*(C)$	Expansion complète de C par rapport à $\mathcal{T}$.
$x \succsim y$	Préférence (x est meilleure que y) dans un problème de comparaison multicritère.
$\phi_i(x, y)$	Indice de préférence partielle de l'alternative x par rapport à y pour le critère i.
$c(x, y)$	Score relatif de l'alternative x par rapport à y.
$\mathcal{C}_E$	Composante E et ensemble des concepts subsumés par E dans une TBox.

Notation	Définition
O^E	Projection d'un concept O sur la composante E.
$rest^E_{Dem}(Rep)$	Rest (information manquante) pour la composante E de la réponse Rep par rapport à une demande utilisateur Dem dans un problème de CCO.
$miss^E_{Dem}(Rep)$	Miss (information superflue)pour la composante E de la réponse Rep par rapport à une demande utilisateur Dem dans un problème de CCO.
$C \ominus_{\mathcal{T}} D$	Différence des concepts C et D selon le CSO.
subd_C	Sous-description d'un concept C.
$\mathsf{subd}^{\mathsf{prim}}_{C,\mathcal{T}}$	Ensemble des sous-description de C où les noms de concept sont des concepts primitifs ou $\top$ par rapport à une TBox $\mathcal{T}$.
$\mathcal{T}^{\mathsf{prim}}_{\mathcal{EL}}$	Ensemble des concept $\mathcal{EL}$ qui peuvent être construit à partir des concepts primitifs de $\mathcal{T}$ et de $\top$.
$\mathsf{br}_{\mathcal{T}}$	Ensemble des branches d'une TBox $\mathcal{T}$.
$\mathsf{br}^{\mathsf{prim}}_{\mathcal{T}}$	Ensemble des branches primitives d'une TBox $\mathcal{T}$.
br_C	Ensemble des branches d'un concept C.
$\mathsf{br}^{\mathsf{prim}}_{C,\mathcal{T}}$	Ensemble des branches de C qui sont primitives dans la TBox $\mathcal{T}$.
$\mathsf{char}^{\mathcal{T}}_C$	Ensemble des branches caractéristique de C par rapport à la TBox $\mathcal{T}$.
$\mathsf{dcom}^{\mathcal{T}}_{C,D}$	Ensemble des $\mathcal{T}$-similarités descriptionnelles de C vers D.
$C \vartriangle D$	Différence des concepts C et D selon le TSO.
$Fup_{\mathcal{T}}(C)$	Fill-up du concept C.

Introduction

Ce travail de thèse s'inscrit dans le cadre du projet Services and Tools for Advanced Metrology (STAM). C'est un projet initié par Deltamu [1], entreprise de métrologie auvergnate. Il vise à développer une plateforme répondant aux nombreux besoins des professionnels de la métrologie. Ce partenariat scientifique et technique entre différents acteurs de la Fabricant permet la naissance de la plateforme STAM et de la marque We Are Metrology (WAM) [2]. On compte parmi les autres partenaires de ce projet Phimeca [3] et Perfect Memory [4] du côté des entreprises et deux laboratoires de recherche le LIMOS [5] et le CleRMa [6].

L'association de ces acteurs a donné le jour à trois outils majeurs de la plateforme STAM. L'outil propose une communauté métrologique pour une amélioration générale des pratiques de chacun et la transmission des savoirs entre les différents secteurs du domaine. Elle propose également une batterie d'outils d'aide à la mesure, en proposant des façons nouvelles d'appréhender l'incertitude de la mesure. Enfin, grâce à la plateforme de gestion de documents de Perfect Memory, STAM offre une base de données de recherche métrologique, utilisant une ontologie, pour permettre à ses utilisateurs de consulter et d'alimenter en ressources (images, documents PDF, descriptions d'instruments ou définitions d'une notion) une base documentaire du domaine. La recherche s'effectue entre autres par l'intermédiaire de mots-clés détectés automatiquement dans les documents et le classement ontologique des sujets abordés.

La plateforme Perfect Memory, bien que très performante, n'a pas la capacité de servir un utilisateur dont la demande précise n'a pas de réponse. C'est-à-dire qu'en cas d'absence de résultat exact à une demande utilisateur, qui satisfait l'ensemble des informations demandées par l'utilisateur, c'est à celui-ci de modifier sa recherche et de trouver une réponse proche de ses besoins. C'est dans cette partie du projet que cette thèse s'inscrit.

Comme la plateforme Perfect Memory utilise les LD, et que ce formalisme s'est déjà montré efficace dans des problèmes de matchmaking, il était logique de les appliquer également dans ces travaux. Nous avons fait le choix d'utiliser la LD $\mathcal{EL}$. En effet, il a été prouvé dans Baader et al. (2017), que $\mathcal{EL}$ conserve un temps polynomial pour la détermination de la subsomption et de la satisfiabilité en présence d'une TBox. Cette LD est une des moins expressives existantes, mais l'est suffisamment pour un grand nombre d'applications, ce que nous avons vérifié dans la construction de l'ontologie de la métrologie pour le projet STAM. Enfin, un dernier atout de cette LD est d'être à la base du profil $\mathcal{EL}$ de OWL2 [7], le langage standard pour la modélisation d'ontologies sur le web, ce qui rend possible l'usage des outils associés existants (éditeurs et raisonneurs, comme Protegé [8] (voir Musen (2015))).

1. https ://www.deltamu.com/
2. https ://wearemetrology.com/
3. https ://www.phimeca.com/
4. https ://www.perfect-memory.com/
5. Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes, https ://limos.fr/
6. https ://clerma.uca.fr
7. https ://www.w3.org/TR/owl2-overview/
8. https ://protege.stanford.edu

Afin de proposer un meilleur service de recherche de ressources métrologiques dans STAM, nous proposons un procédé de recommandation de réponse à une demande, utilisant les principes du matchmaking sémantique, où les descriptions des réponses et des demandes sont formalisées à l'aide des LD (voir Baader et al. (2017)). On souhaite départager les réponses par rapport à leurs informations manquantes et superflues par rapport à la demande. Une information manquante est une information présente dans la demande et absente de la réponse. Une information superflue est une information présente dans la réponse et absente de la demande. Ce procédé de recommandation emprunte au matchmaking multicritère la découpe du matchmaking en sous-parties, appelées composantes, pour permettre un plus grand contrôle de l'expert du domaine sur la recommandation. L'utilisation de composantes réduit également l'impact de la structure de l'ontologie sur les résultats. En effet, la construction même des ontologies impliquent souvent que certains aspects d'une réponse nécessitent beaucoup de concepts pour être explicités, sans pour autant revêtir plus d'importance aux yeux d'un utilisateur. Cela signifie qu'une réponse très pertinente pourrait être écartée si elle ne possède pas cet aspect de la demande, minime d'un point de vue pertinence mais maximum en termes de taille. Plutôt que d'effectuer le matchmaking sur l'ensemble d'une description, nous allons l'appliquer sur les différentes sous-parties de la description, et nous servir de l'agrégation des différents résultats pour classer les réponses possibles à la demande, de la plus proche à la plus éloignée. Il existe déjà plusieurs techniques de matchmaking, proposées notamment par Colucci et al. (2004), Noia et al. (2011), Noia et al. (2003), Benatallah et al. (2003). Néanmoins, certain de ces procédés ne sont pas applicables dans notre cas (limitation liée à la logique de description utilisée) ou ne possèdent pas le degré de précision que nous aimerions avoir, car ils ne traitent notamment pas des informations superflues lors du classement des réponses potentielles. Seule une application du principe des meilleures couvertures (voir Benatallah et al. (2003)) est en mesure de nous offrir cette précision, en mettant en exergue les informations manquantes et superflues d'une réponse par rapport à une demande utilisateur, nous adaptons donc celui-ci à notre besoin.

Exemple 1. *Soit une ontologie de métrologie contenant les connaissances suivantes exprimées avec la LD $\mathcal{EL}$ et formant l'ensemble $\mathcal{T}$ qu'on appelle une TBox :*

$\mathcal{T} =$

$$
\begin{array}{lcl}
\{Bois & \sqsubseteq & Matériau, \\
Chêne & \sqsubseteq & Bois, \\
Métal & \sqsubseteq & Matériau, \\
Plastique & \sqsubseteq & Matériau, \\
Fer & \sqsubseteq & Métal, \\
Acier & \sqsubseteq & Métal, \\
PiedaCoulisse & \sqsubseteq & Type, \\
Micromètre & \sqsubseteq & PiedaCoulisse, \\
Règle & \sqsubseteq & Type, \\
Fabricant - Européen & \sqsubseteq & Fabricant, \\
Fabricant - Américain & \sqsubseteq & Fabricant,
\end{array}
$$

$$
\begin{array}{lcl}
Fabricant - Français & \sqsubseteq & Fabricant - Européen, \\
Fabricant - Allemand & \sqsubseteq & Fabricant - Européen, \\
Fabricant - Canadien & \sqsubseteq & Fabricant - Américain, \\
Mm & \sqsubseteq & Unité, \\
Cm & \sqsubseteq & Unité\ , \\
Analogique & \sqsubseteq & ModeLecture, \\
Numérique & \sqsubseteq & ModeLecture\}.
\end{array}
$$

$\mathcal{T}$ *indique que le Chêne est une sorte de Bois qui est un type de Matériau (même chose pour Acier, Fer, Métal et Plastique). PiedaCoulisse (abrégé PaC) est un type d'Instrument (il permet de mesurer des longueurs). Un Micromètre est un type de PaC. La Règle est également un Type d'instrument. Fabricant−Européen et Fabricant−Américain sont des Fabricant de différentes zones géographiques. Fabriquant−Français et Fabricant − Allemand sont des fabricants européens et Fabricant − Canadien un fabricant américain (Fabricant sera abrégé Fab−). Le Mm et le Cm sont des Unités de mesure. Analogique et Numérique sont les deux ModeLecture possible d'un instrument. L'ontologie est illustrée à la figure 1 (p.5). Le concept Instrument est défini par cinq liens : ses matériaux, son type, son unité, son mode de lecture et son fabricant. Ces liens sont les composantes à partir desquelles la recherche multicritère se fait. Leur nom est indiqué à côté des pointillés sur la figure 1 (p.5) : aMatériau, aType, aMode, aUnité, aFabricant. L'ontologie permet à un utilisateur d'exprimer des demandes, par exemple d'instruments, décrits par leur cinq composantes : leur type, leur mode de lecture, le(s) matériau(x) qui le composent, leur unité de mesure et leur fabricant.*

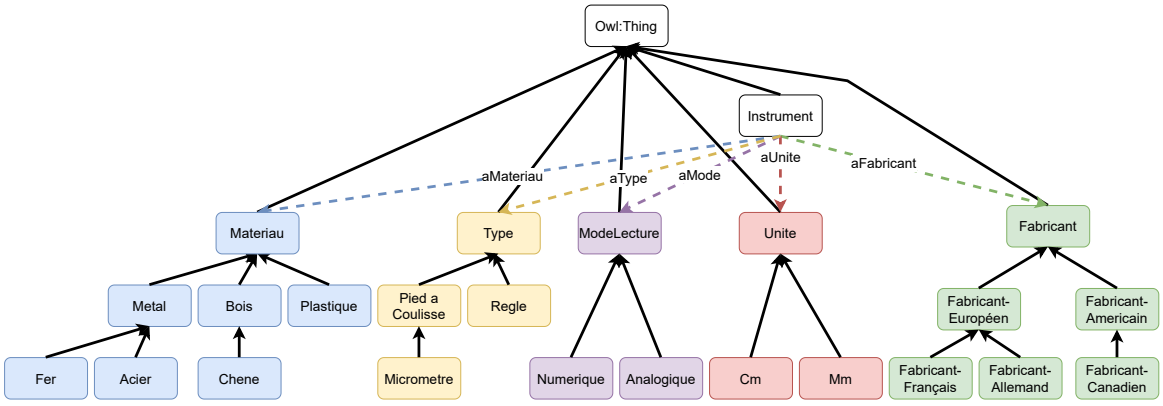


FIGURE 1 – Ontologie de l'exemple 1 (p.4).

Imaginons les deux demandes suivantes :

$Dem_1 \equiv \exists aType.PaC \sqcap \exists aMatériau.Acier \sqcap \exists aFabricant.Fab - Français \sqcap \exists aUnité.Mm,$

$Dem_2 \equiv \exists aType.Micromètre \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne) \sqcap \exists aFabricant.Fab - Européen.$

Puisque ce sont des demandes d'instruments, Dem_1 et Dem_2 précisent tout ou parties des cinq composantes définissant un instrument. Dem_1 demande des ressources à propos

d'un PaC fait en Acier fabriqué par un $Fab-Français$ et qui mesure en Mm. Dem_2 demande des ressources à propos d'un Micromètre Analogique fait en Plastique, en Métal et en Chêne et fabriqué par un $Fab-Européen$, sans précision sur l'unité de mesure. Supposons maintenant quatre réponses aux demandes avec les descriptions de ressource suivantes :

$Rep_1 \equiv \exists aType.PaC \sqcap \exists aMode.Analogique \sqcap \exists aUnité.Cm \sqcap \exists aFabricant.Fab - Allemand$

qui décrit un PaC Analogique qui mesure en Cm et est fabriqué par un $Fab-Allemand$.

$Rep_2 \equiv \exists aType.PaC \sqcap \exists aMatériau.Plastique \sqcap \exists aMode.Numérique \sqcap \exists aFabricant.Fab - Français \sqcap \exists aUnité.Mm.$

qui décrit un PaC Numérique fait en Plastique, qui est fabriqué par un $Fab-Français$, et qui mesure en Mm.

$Rep_3 \equiv \exists aType.Micromètre \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Acier) \sqcap \exists aFabricant.Fab - Français$

qui décrit un Micromètre Analogique fait en Plastique et en Acier et fabriqué par un $Fab - Français$.

$Rep_4 \equiv \exists aType.Règle \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne) \sqcap \exists aUnité.Mm \sqcap \exists aFabricant.Fab - Européen$

qui décrit une Règle Analogique fait en Métal, en Chêne et en Plastique, qui mesure en Mm et qui est fabriquée par un $Fab - Européen$.

On peut alors identifier les informations manquantes et superflues de chaque réponse par rapport aux demandes. Dans la suite de l'exemple, les connaissances soulignées correspondent aux informations manquantes et les connaissances en gras correspondent aux informations superflues.

Pour la demande Dem_1 :

$Dem_1 \equiv \exists aType.PaC \sqcap \exists aMatériau.\underline{Acier} \sqcap \exists aFabricant.\underline{Fab - Français} \sqcap \exists aUnité.\underline{Mm}$

$Rep_1 \equiv \exists aType.PaC \sqcap \exists aMode.\boldsymbol{Analogique} \sqcap \exists aUnité.\boldsymbol{Cm} \sqcap \exists aFabricant.\boldsymbol{Fab\text{-}Allemand}$

Rep_1 ne répond pas à Mm, $Fab - Français$ et Acier : ce sont les informations manquantes. De plus, elle apporte Cm, $Fab-Allemand$ et Analogique qui ne sont pas dans la demande : ce sont les informations superflues (voir figure 1 (p.7)). Tels qu'ils sont représentés dans l'ontologie, il n'y a pas de relation entre les concepts Cm et Mm.

$Dem_1 \equiv \exists aType.PaC \sqcap \exists aMatériau.\underline{Acier} \sqcap \exists aFabricant.Fab-Français \sqcap \exists aUnité.Mm$

$Rep_2 \equiv \exists aType.PaC \sqcap \exists aMatériau.\boldsymbol{Plastique} \sqcap \exists aMode.\boldsymbol{Numérique} \sqcap \exists aFabricant.Fab - Français \sqcap \exists aUnité.Mm.$

Rep_2 ne répond pas à Acier, c'est la seule information manquante.
Plastique et Numérique sont les informations superflues.

$Dem_1 \equiv \exists aType.PaC \sqcap \exists aMatériau.Acier \sqcap \exists aFabricant.Fab-Français \sqcap \exists aUnité.\underline{Mm}$

$Rep_3 \equiv \exists aType.\boldsymbol{Micromètre} \sqcap \exists aMode.\boldsymbol{Analogique} \sqcap \exists aMatériau.(\boldsymbol{Plastique} \sqcap Acier) \sqcap \exists aFabricant.Fab - Français$

Rep_3 ne répond pas à Mm, c'est la seule information manquante. Comme Micromètre est un type de pied à coulisse, Rep_3 répond bien à la composante aType de Dem_1. Cependant, Micromètre étant plus spécifique que PaC, Micromètre représente une information superflue de Rep_3 par rapport à Dem_1. De plus, elle apporte Plastique et

Analogique, ce sont les informations superflues.
$Dem_1 \equiv \exists aType.\underline{PaC} \sqcap \exists aMatériau.\underline{Acier} \sqcap \exists aFabricant.\underline{Fab-Français} \sqcap \exists aUnité.Mm$
$Rep_4 \equiv \exists aType.\boldsymbol{Règle} \sqcap \exists aMode.\boldsymbol{Analogique} \sqcap \exists aMatériau.(\boldsymbol{Plastique} \sqcap Métal \sqcap \boldsymbol{Chêne}) \sqcap \equiv \exists aUnité.Mm \sqcap \exists aFabricant.Fab - Européen$
Rep_4 ne répond pas à $Fab-Français$, Acier, PaC, ce sont les informations manquantes. De plus, elle apporte Règle, Analogique, Chêne et Plastique, ce sont les informations superflues. La $Fab - Français$ étant un $Fab - Européen$, le concept $Fab - Européen$ n'est pas considéré comme une information superflue. Il en va de même pour Métal, puisque l'acier en est un.

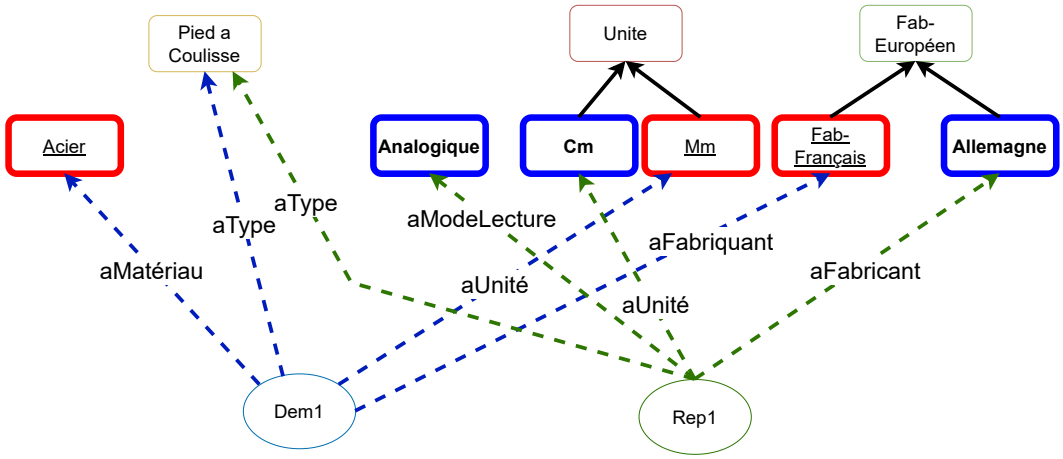


FIGURE 2 – Un exemple d'une demande Dem_1 et d'une réponse Rep_1, exprimées avec l'ontologie de la figure 1 (p.5). Les informations manquantes sont soulignées et les informations superflues sont en gras .

Pour la demande Dem_2 :
$Dem_2 \equiv \exists aType.\underline{Micromètre} \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(\underline{Plastique} \sqcap \underline{Métal} \sqcap \underline{Chêne}) \sqcap \exists aFabricant.Fab - Européen.$
$Rep_1 \equiv \exists aType.PaC \sqcap \exists aMode.Analogique \sqcap \exists aUnité.\boldsymbol{Cm} \sqcap \exists aFabricant.\boldsymbol{Fab\text{-}Allemand}$
Rep_1 ne répond pas à Plastique, Métal, Chêne et Micromètre : ce sont les informations manquantes. De plus, elle apporte $Fab - Allemand$ et Cm qui ne sont pas dans la demande : ce sont des informations superflues.
$Dem_2 \equiv \exists aType.\underline{Micromètre} \sqcap \exists aMode.\underline{Analogique} \sqcap \exists aMatériau.(Plastique \sqcap \underline{Métal} \sqcap \underline{Chêne}) \sqcap \exists aFabricant.Fab - Européen.$
$Rep_2 \equiv \exists aType.PaC \sqcap \exists aMatériau.Plastique \sqcap \exists aMode.\boldsymbol{Numérique} \sqcap \exists aFabricant.\boldsymbol{Fab\text{-}Français} \sqcap \exists aUnité.\boldsymbol{Mm}.$
Rep_2 ne répond pas à Métal, Chêne, Analogique, Micromètre : ce sont les informations manquantes. De plus, elle apporte Mm, Numérique et $Fab - Français$ qui sont des informations superflues.
$Dem_2 \equiv \exists aType.Micromètre \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap$

$\underline{Chêne}) \sqcap \exists aFabricant.Fab - Européen.$
$Rep_3 \equiv \exists aType.Micromètre \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap \boldsymbol{Acier}) \sqcap \exists aFabricant.\boldsymbol{Fab\text{-}Français}$
Rep_3 ne répond pas à Chêne : c'est l'information manquante. De plus, elle apporte Acier et $Fab - Français$ qui sont des informations superflues.
$Dem_2 \equiv \exists aType.\underline{Micromètre} \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne) \sqcap \exists aFabricant.Fab - Européen.$
$Rep_4 \equiv \exists aType.\boldsymbol{Règle} \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne) \sqcap \exists aUnité.\boldsymbol{Mm} \sqcap \exists aFabricant.Fab - Européen$
Rep_4 ne répond pas à Micromètre : c'est l'information manquante. De plus, elle apporte Règle et Mm qui sont des informations superflues.

Si l'on regarde uniquement le nombre d'informations manquantes entre la demande et une réponse, on obtient les ordres suivants (le signe = signifie qu'il y a autant d'informations manquantes dans les deux opérandes et le signe > signifie que l'opérande de gauche a moins d'informations manquantes que l'opérande de droite, et donc qu'il est meilleur) :
Pour Dem_1 : $Rep_2 = Rep_3 > Rep_1 = Rep_4$, Rep_2 et Rep_3 sont meilleures que Rep_1 et Rep_4.
Pour Dem_2 : $Rep_4 = Rep_3 > Rep_2 = Rep_1$, Rep_4 et Rep_3 sont meilleures que Rep_2 et Rep_1.

Pour affiner le résultat, on peut prendre en compte les informations superflues, dans le cas des égalités. On obtient alors les ordres suivants :
Pour Dem_1 : $Rep_2 > Rep_3 > Rep_4 = Rep_1$, Rep_2 est meilleure que Rep_3 qui est meilleure que Rep_4 et Rep_1.
Pour Dem_2 : $Rep_4 = Rep_3 > Rep_1 > Rep_2$, Rep_4 et Rep_3 sont meilleures que Rep_1 qui est meilleure que Rep_3.
On peut alors s'interroger sur la pertinence de certains des résultats. En effet, on propose une règle dans Rep_4 à un utilisateur ayant demandé un micromètre. Or ce sont deux instruments très différents et il est raisonnable de penser que renvoyer Rep_4 à l'utilisateur ne va pas le satisfaire dans le cas de Dem_2. Le problème vient du fait qu'on a classé les réponses sans tenir compte des composantes définissant un instrument.

Pour obtenir de meilleurs classements des réponses, on compare les différentes composantes entre elles pour déterminer la meilleure ressource, en prenant en compte l'information manquante et superflue uniquement pour cette composante :

- *Composante type d'instrument :*
  Rep_2 et Rep_1 répondent à Dem_1 sans apporter d'information superflue, elles sont donc équivalentes. Rep_3 répond à Dem_1 mais apporte une information superflue (Micromètre), elle est donc moins bien classée. Enfin, Rep_4 ne répond pas à Dem_1 alors :
  Pour Dem_1 : $Rep_2 = Rep_1 > Rep_3 > Rep_4$,
  De la même manière :

Pour Dem_2 : $Rep_3 > Rep_2 = Rep_1 > Rep_4$

— *Composante matériau :*
Pour Dem_1 : $Rep_3 > Rep_1 > Rep_2 > Rep_4$
Pour Dem_2 : $Rep_4 > Rep_3 > Rep_2 > Rep_1$

— *Composante lieu de Fabrication :*
Pour Dem_1 : $Rep_2 = Rep_3 > Rep_4 > Rep_1$
Pour Dem_2 : $Rep_4 > Rep_1 = Rep_2 = Rep_3$

— *Composante unité :*
Pour Dem_1 : $Rep_2 = Rep_4 > Rep_3 > Rep_1$
Pour Dem_2 : $Rep_3 > Rep_4 = Rep_1 = Rep_2$

— *Composante mode de Lecture :*
Pour Dem_1 : $Rep_2 = Rep_1 = Rep_3 = Rep_4$
Pour Dem_2 : $Rep_4 = Rep_3 = Rep_1 > Rep_2$

On donne ensuite un score à chaque réponse en fonction des scores intermédiaires. Si une réponse est meilleure qu'une autre pour une composante, son score sur cette composante augmente de 1. Le score augmente de 0 en cas d'égalité et de -1 si une réponse est moins bonne qu'une autre. Par exemple, pour la composante type d'instrument : pour Dem_1 : $Rep_2 = Rep_1 > Rep_3 > Rep_4$, Rep_2 est à égalité avec Rep_1 et est meilleure que Rep_3 et Rep_4, son score de composante est donc de 2. Rep_1 est à égalité avec Rep_2 et est meilleure que Rep_3 et Rep_4, son score de composante est donc de 2. Rep_3 est meilleure que Rep_4 et est moins bonne que Rep_2 et Rep_1, son score de composante est donc de -1. Rep_4 est moins bonne que les autres réponses, son score de composante est donc de -3
On obtient alors les scores suivants pour chaque réponse en additionnant les scores des 5 composantes d'un instrument :

$Score_{Dem_1}(Rep_1)$= 2 + 1 - 3 - 3 + 0 = -3
$Score_{Dem_1}(Rep_2)$= 2 - 1 + 2 + 2 + 0 = 5
$Score_{Dem_1}(Rep_3)$=-1 + 3 + 2 - 1 + 0 = 3
$Score_{Dem_1}(Rep_4)$=-3 - 3 - 1 + 2 + 0 = -5

$Score_{Dem_2}(Rep_1)$= 0 - 3 - 1 - 1 + 1 = -4
$Score_{Dem_2}(Rep_2)$= 0 - 1 - 1 - 1 - 3 = -6
$Score_{Dem_2}(Rep_3)$= 3 + 1 - 1 + 3 + 1 = 7
$Score_{Dem_2}(Rep_4)$=-3 + 3 + 3 - 1 + 1 = 3

La somme de ces scores de composante donne l'ordre global suivant :
Pour Dem_1 : Rep_2 (5) > Rep_3 (3)> Rep_1 (-3)> Rep_4 (-5)
Pour Dem_2 : Rep_3(7) > Rep_4(3) > Rep_1(-4) > Rep_2(-6)

Cette fois, Rep_4 n'est pas équivalent à Rep_3 pour Dem_2. Le problème a été résolu par la prise en compte des composantes. Par la suite, comme pour l'exemple précédent, nous utilisons demande pour parler de la demande de l'utilisateur à la base de connaissances. De la même manière, nous utilisons réponse pour parler de toute description de ressource pouvant constituer une réponse possible à la demande.

Nous traitons donc dans ce document d'un problème de matchmaking pour la recommandation. Plus précisément, nous étudions une méthode de matchmaking sémantique

multicritères. Nos contributions sont :

- la définition et l'étude d'un nouvel opérateur de différence appelé CSO (Component Subtraction Operator) pour la LD $\mathcal{EL}$ permettant de définir les notions d'informations manquantes et superflues.
- la définition et l'étude d'une nouvelle approche de matchmaking sémantique, appelée CCO (Component Comparison Operator), intégrant un principe de comparaison multicritère
- l'implémentation et le test des CSO et CCO en lien avec la plateforme Perferct Memory de gestion de ressources.

Ce document comporte une première partie théorique traitant de notre approche. Au chapitre 1 des rappels théoriques sur les Logiques de Descriptions (section 1.1) et un état de l'art sur le matchmaking (section 1.2) et la différence sémantique dans les logiques de description (section 1.3) sont proposés. On y trouve également les explications du principe de comparaison multicritère (section 1.4). Dans le Chapitre 2, nous présentons notre approche de matchmaking multicritère (section 2.2. Le CCO nécessitant un opérateur de différence, nous proposons dans le Chapitre 3 un opérateur pour $\mathcal{EL}$, le CSO, étudié dans deux cas : l'un pour des terminologies acycliques (section 3.1) et définitionnelles, et un second pour les terminologies acycliques et générales (section 3.2).

Vient ensuite une seconde partie traitant de l'implémentation pratique du CCO, en lien avec le projet STAM dans laquelle cette thèse s'inscrit. Dans le chapitre 4 nous présentons Perfect Memory partenaire du projet STAM et responsable de la partie information et ontologie du projet, sa plateforme de recherche et comment nous proposons d'intégrer le CCO à celle-ci . Le chapitre 5 porte sur l'implémentation du CCO réalisée en Ruby. Enfin le chapitre 6 présente les protocoles et résultats des tests réalisés sur l'implémentation.

Partie un : Matchmaking multicritère et différence sémantique

Chapitre 1

Rappels théoriques

Dans ce chapitre, nous présentons des rappels sur les logiques de description à la section 1.1 ainsi qu'un état de l'art des différents mécanismes utilisés dans notre processus de matchmaking : les raisonnements de matchmaking dans les LD à la section 1.2, les raisonnements de différences à la section 1.3 et à la section 1.4 les principes de comparaison multicritère.

Sommaire

1.1 Logiques de description

Les logiques de description (LD) sont un formalisme de représentation des connaissances et de raisonnement. Plus précisément, elles constituent une famille de sous-langages de la logique du premier ordre munie de la sémantique classique basée sur la théorie des modèles, mais aussi d'une syntaxe particulière basée sur les notions de concept, de rôle et d'individu.

Intuitivement, on considère que l'on a un ensemble d'éléments appelés domaine ou univers. Un concept est identifié par un nom comme *Instrument* ou *Matériau*. Un concept définit un ensemble d'éléments du domaine. Un rôle est identifié par un nom comme $aMatériau$ ou $aType$. Un rôle définit un ensemble de couples d'éléments du

domaine. Par exemple, le rôle $aMatériau$ définit un ensemble de couples d'éléments dont le premier est un instrument et le second est le matériau constituant cet instrument. Un individu est identifié par un nom comme $RègleBois2$ ou $Acier1$. Un individu correspond à un élément du domaine.

Plus formellement, on suppose avoir deux ensembles dénombrables infinis : **C** pour les noms de concept et **r** pour les noms de rôles. Avec l'aide de constructeurs, des descriptions de concept complexes peuvent être construites. Chaque logique de description est définie par une liste de constructeurs. La table 1.1 (p.15) regroupe la syntaxe et la sémantique des constructeurs de la logique de description $\mathcal{EL}$. Par la suite, on emploiera le terme "concept" en lieu et place de l'expression "description de concept $\mathcal{EL}$". Par exemple, $\exists aMatériau.Acier \sqcap \exists aType.PaC$ est le concept qui décrit l'ensemble des éléments du domaine reliés à un élément du concept $Acier$, par le rôle $aMatériau$ et à un élément du concept PaC par le rôle $aType$.

Pour un concept C on peut définir sa taille et sa profondeur.

Definition 1 (Taille (voir Baader et al. (2017)))**.** *Pour un concept C, sa taille, notée* $\mathsf{size}(C)$*, est définie par récurrence sur sa propre structure :*
- *si* $C \in \mathbf{C} \cup \top$ *alors* $\mathsf{size}(C) = 1$.
- *si* $C = C_1 \sqcap C_2$*, alors* $\mathsf{size}(C) = 1 + \mathsf{size}(C_1) + \mathsf{size}(C_2)$.
- *si* $C = \exists r.D$ *alors* $\mathsf{size}(C) = 1 + \mathsf{size}(D)$.

Definition 2 (Profondeur(voir Baader et al. (2017)))**.** *Pour un concept C, sa profondeur, notée* $\mathsf{rd}(C)$*, est définie par récurrence sur sa propre structure :*
- *si* $C \in \mathbf{C} \cup \top$ *alors* $\mathsf{rd}(C) = 0$.
- *si* $C = C_1 \sqcap C_2$*, alors* $\mathsf{rd}(C) = Max(\mathsf{rd}(C_1), \mathsf{rd}(C_2))$.
- *si* $C = \exists r.D$ *alors* $\mathsf{rd}(C) = 1 + rd(D)$.

Par exemple, $\mathsf{size}(\exists aMatériau.Acier \sqcap \exists aType.PaC) = 5$ et $\mathsf{rd}(\exists aMatériau.Acier \sqcap \exists aType.PaC) = 1$.

Nous proposons également la définition de la largeur d'un concept :

Definition 3 (Largeur d'un concept)**.** *Pour un concept C, sa largeur, notée* $\mathsf{wid}(C)$*, est définie par récurrence :*
- *si* $C \in \mathbf{C} \cup \top$ *alors* $\mathsf{wid}(C) = 1$.
- *si* $C = \prod_{i=1}^{n} C_i$*, avec* $n \geq 2$*, alors* $\mathsf{wid}(C) = Max(n, Max(\mathsf{wid}(C_i), 1 \leq i \leq n))$.
- *si* $C = \exists r.D$ *alors* $\mathsf{wid}(C) = \mathsf{wid}(D)$.

Par exemple, $\mathsf{wid}(\exists aMatériau.Acier \sqcap \exists aType.PaC) = 2$.

On donne aux concepts une sémantique basée sur la notion d'interprétation. Une interprétation est une paire $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, où $\Delta^{\mathcal{I}}$ est le domaine qui regroupe l'ensemble des individus étudiés et $.^{\mathcal{I}}$ la fonction d'interprétation qui relie tous les concepts à une partie de $\Delta^{\mathcal{I}}$, et chaque rôle à une partie de $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Les concepts peuvent être reliés entre eux par des axiomes dont la sémantique est donnée dans le tableau 1.1 (p.15). Il en existe trois types :

Constructeurs/Axiomes	Syntaxe	Sémantique
Top	$\top$	$\Delta^{\mathcal{I}}$
Nom de concept $\in \mathbf{C}$	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Rôle $\in \mathbf{r}$	r	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Conjonction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Restriction existentielle	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in C^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}}\}$
General Concept Inclusion (GCI)	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Définition de concept	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$
Définition de concept primitif	$A \sqsubseteq C$	$A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$

TABLE 1.1 – Constructeurs et axiomes pour $\mathcal{EL}$. $A \in \mathbf{C}$, $r \in \mathbf{r}$, et C et D sont des concepts. On considère que les conjonctions ne contiennent pas plusieurs fois le même conjonct, et qu'écrire $\bigsqcap_{i=1}^{n} C_i$ signifie que les C_is ne sont pas eux-mêmes des conjonctions.

— Les general concept inclusions (GCI), notés $C \sqsubseteq D$, où C et D sont des descriptions de concept spécifiant que tous les individus de C sont aussi des individus de D. On lit ainsi "C est subsumé par D".
— Les axiomes de définition de concept sont notés $A \equiv C$ et permettent de nommer le concept C par le nom de concept A. Ainsi les ensembles d'individus correspondants à A et C sont les mêmes. On lit ainsi "A est défini par C".
— Les axiomes de définition de concept primitif sont notés $A \sqsubseteq C$. Ils permettent d'indiquer que A est un sous-concept de C. On lit ainsi "A est subsumé par C".

La taille d'un axiome est la somme des tailles de la partie droite et gauche de cet axiome.

Un ensemble fini d'axiomes est appelé Terminological Box (TBox). Une interprétation $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ est un modèle d'une TBox $\mathcal{T}$ si, pour chaque axiome de $\mathcal{T}$, la condition donnée dans la troisième colonne du tableau 1.1 (p.15) est satisfaite. Un concept C est subsumé par un autre concept D par rapport à une TBox $\mathcal{T}$, noté $C \sqsubseteq_{\mathcal{T}} D$ (or $\mathcal{T} \models C \sqsubseteq D$), si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ pour tous les modèles de $\mathcal{T}$.

On appelle TBox générale une TBox composée de GCIs (ce qui inclut les définitions de concept et les définitions de concept primitif). On appelle TBox définitionnelle une TBox uniquement composée de définitions de concept. Dans ces TBox, les noms de concept apparaissant à gauche d'une définition sont appelés *concepts définis* et définissent l'ensemble noté $\mathsf{def}_{\mathcal{T}} \subseteq \mathbf{C}_{\mathcal{T}}$. Un concept défini ne peut apparaître qu'une seule fois à gauche d'une définition de concept. Les autres noms de concept sont appelés *concepts primitifs* et forment l'ensemble $\mathsf{prim}_{\mathcal{T}} \subseteq \mathbf{C}_{\mathcal{T}}$. La taille d'une TBox est la somme des tailles de ses axiomes.

Une TBox contenant uniquement des définitions de concept et des définitions de concept primitif peut être transformée en TBox définitionnelle en temps linéaire par rapport à $\mathsf{size}(\mathcal{T})$ (voir Baader et al. (2017)) puisque (i) chaque définition de concept primitif $A \sqsubseteq C$ peut être transformé en la définition de concept $A \equiv C \sqcap \overline{A}$, avec $\overline{A}$ un nouveau nom de concept non utilisé dans un autre axiome, et (ii) deux définitions de concept primitif $A \sqsubseteq B$ et $A \sqsubseteq C$ peuvent être rassemblées en un axiome de type

$A \sqsubseteq B \sqcap C$. Une TBox définitionnelle obtenue depuis une TBox non définitionnelle $\mathcal{T}$ est notée $\overline{\mathcal{T}}$.

En étendant une définition de Baader et al. (2017), pour A, B, B' des noms de concept, on dit que A *utilise directement* B s'il existe une GCI $A \sqsubseteq C$ dans $\mathcal{T}$ tel que B apparaît dans C. On dit que A *utilise* B si A utilise directement B ou s'il existe un B' tel que A utilise B' et B' utilise directement B. Une TBox contient un cycle quand un nom de concept s'utilise lui-même. Par la suite, les TBox sont considérées acycliques sauf mention contraire.

La *signature d'une TBox* $\mathcal{T}$, notée, $\mathsf{sig}_\mathcal{T}$, est l'ensemble de tous les noms de concept et de rôle apparaissant dans $\mathcal{T}$. On note $\mathbf{C}_\mathcal{T} = \mathbf{C} \cap \mathsf{sig}_\mathcal{T}$, $\mathbf{r}_\mathcal{T} = \mathbf{r} \cap \mathsf{sig}_\mathcal{T}$, et $\mathcal{T}_{\mathcal{EL}}$ l'ensemble de tous les $\mathcal{EL}$ concepts qui peuvent être construits à partir de $\mathsf{sig}_\mathcal{T}$ et $\top$. Si la LD étudiée n'est pas $\mathcal{EL}$ mais $\mathcal{L}$, l'ensemble de tous les concepts pouvant être construits à partir de $\mathsf{sig}_\mathcal{T}$ et $\top$ est noté $\mathcal{T}_\mathcal{L}$.

Dans les logiques de descriptions, on considère en général les problèmes suivants :

Definition 4. *Voir Baader et al. (2017) Soit une TBox $\mathcal{T}$, C et D des concepts, on dit que :*

1. *C est satisfiable par rapport à $\mathcal{T}$ s'il existe un modèle $\mathcal{I}$ de $\mathcal{T}$ et un $d \in \Delta^\mathcal{I}$ quelconque tel que $d \in C^\mathcal{I}$.*
2. *C est subsumé par D par rapport à $\mathcal{T}$, écrit $\mathcal{T} \models C \sqsubseteq D$, si $C^\mathcal{I} \subseteq D^\mathcal{I}$ pour tous les modèles $\mathcal{I}$ de $\mathcal{T}$.*
3. *C est équivalent par D par rapport à $\mathcal{T}$, écrit $\mathcal{T} \models C \equiv D$, si $C^\mathcal{I} = D^\mathcal{I}$ pour tous les modèles $\mathcal{I}$ de $\mathcal{T}$.*

On utilise ici $\models$ car l'implication dans les DL coïncide avec la sémantique de l'implication dans la Logique du Premier Ordre. Quand la TBox $\mathcal{T}$ est définie, la subsomption et l'équivalence par rapport à $\mathcal{T}$ sont des relations binaires entre deux concepts et on peut les écrire ainsi : $C \sqsubseteq_\mathcal{T} D$ pour $\mathcal{T} \models C \sqsubseteq D$ et $C \equiv_\mathcal{T} D$ pour $\mathcal{T} \models C \equiv D$.

Dans Baader et al. (2005, 2017) est proposée une normalisation de TBox, s'effectuant en temps linéaire par rapport à la taille de celle-ci, pour une TBox générale $\mathcal{T}$ cyclique ou acyclique. La normalisation génère $\mathcal{T}_{Bnf}$ qui ne contient que des axiomes de la forme $A \sqsubseteq B$, $A_1 \sqcap A_2 \sqsubseteq B$, $A \sqsubseteq \exists r.B$ ou $\exists r.A \sqsubseteq B$, avec A, B, A_1 et A_2 des noms de concept. Cette normalisation assure, avec $(A, B) \in \mathbf{C}^2_\mathcal{T}$ que $\forall (C, D) \in \mathcal{T}^2_{\mathcal{EL}}, C \sqsubseteq_\mathcal{T} D \Leftrightarrow C \sqsubseteq_{\mathcal{T}_{Bnf}} D$. La normalisation peut générer de nouveaux noms de concept qui n'étaient pas dans $\mathcal{T}$. Les règles de normalisations sont rappelées dans la table 5.3 (p.78).

À partir de cette TBox $\mathcal{T}_{Bnf}$ normalisée, Baader et al. (2005, 2017) définissent également la classification d'une TBox, qui calcule toutes les relations de subsomption entre les couples de noms de concept de $\mathcal{T}_{Bnf}$. À partir de règles de classification, il est possible d'effectuer cette classification en saturant la TBox avec chaque axiome de subsomption jusqu'à ce que toutes les relations soient explicitées (voir Baader et al. (2017)). Une seconde méthode (voir Baader et al. (2005)), utilisant le même type de règles, effectue la classification $Cl_{\mathcal{T}_{Bnf}}$ de $\mathcal{T}_{Bnf}$ qui contient les ensembles $\mathbf{S}(A), \forall A \in \mathbf{C}_{\mathcal{T}_{Bnf}} \cup \{\top\}$ tels que : $B \in \mathbf{S}(A)$ si et seulement si $A \sqsubseteq_{\mathcal{T}_{Bnf}} B$ et quelques ensembles $\mathbf{R}(r)$, avec

$r \in \mathbf{r}_{\mathcal{T}_{Bnf}}$ tels que : $(A,\ B) \in \mathbf{R}(r)$ implique $A \sqsubseteq_{\mathcal{T}_{Bnf}} \exists r.B$. Les règles de classification sont rappelées dans la table 5.4 (p.81). La classification s'effectue en temps polynomial par rapport à la taille de $\mathcal{T}_{Bnf}$. Enfin, nous aurons besoin de l'expansion complète $\mathcal{T}^*$ (voir Nebel (1990); Baader et al. (2017)) d'une TBox $\mathcal{T}$ acyclique et définitionnelle, qui réécrit chaque définition de concept de $\mathcal{T}$ en une définition équivalente ne contenant que des concepts primitifs dans sa partie droite. Pour un concept C, $\mathcal{T}^*(C)$ est l'expansion complète de C par rapport à $\mathcal{T}$. Si $C = \mathcal{T}^*(C)$, on dira alors que C est entièrement étendu. Ce processus se fait en temps exponentiel par rapport à la taille de $\mathcal{T}$.

1.2 Matchmaking dans les logiques de description

Dans les logiques de description, nous proposons de définir un problème de matchmaking de la manière suivante :

Definition 5. *Soit $\mathcal{L}$ une logique de description, $\mathcal{T}$ une TBox dans $\mathcal{L}$, $\mathcal{R}$ un ensemble fini de concepts de $\mathcal{T}_\mathcal{L}$ et Dem un concept de $\mathcal{T}_\mathcal{L}$. Soit par ailleurs une relation binaire $\rho \subseteq \mathcal{T}_\mathcal{L} \times \mathcal{T}_\mathcal{L}$, appelée critère de proximité sémantique, et un ordre total ou partiel $\ll$ entre concepts de $\mathcal{T}_\mathcal{L}$.*

Un problème de matchmaking est décrit par un sextuple $\{\mathcal{L}, \mathcal{T}, \mathcal{R}, Dem, \rho, \ll\}$ et a comme solutions tous les concepts $Rep \in \mathcal{R}$ vérifiant :

1. *$\rho(Dem, Rep)$, et*
2. *Rep est minimal par rapport $\ll$ tel que 1. est vrai.*

On peut maintenant définir ce qu'est une information manquante et superflue dans le cadre du matchmaking. Pour ce faire, on supposera qu'on dispose :

— d'un opérateur binaire $\ominus_\mathcal{T} : \mathcal{T}_\mathcal{L} \times \mathcal{T}_\mathcal{L} \to \mathcal{T}_\mathcal{L}$ de soustraction de concepts produisant un résultat unique modulo $\equiv_\mathcal{T}$. Informellement, un tel opérateur contient les informations du premier opérande et pas celles du second. On étudie un tel opérateur en détail à la section 3 (p.45), qui donne par exemple : $(\exists r.(A \sqcap B)) \ominus_\mathcal{T} (\exists r.B) = \exists r.A$.

— d'un opérateur binaire $\oplus_\mathcal{T} : \mathcal{T}_\mathcal{L} \times \mathcal{T}_\mathcal{L} \to \mathcal{T}_\mathcal{L}$ d'addition de concepts produisant un résultat unique modulo $\equiv_\mathcal{T}$. Informellement, un tel opérateur contient toutes les informations contenues dans ses deux opérandes. Par exemple $\sqcap$ est un tel opérateur : $C \oplus_\mathcal{T} D = C \sqcap D$.

Definition 6 (Information manquante et superflue)**.** *Soit Rep une solution du problème de matchmaking $\{\mathcal{L}, \mathcal{T}, \mathcal{R}, Dem, \rho, \ll\}$. Soient $\oplus_\mathcal{T}$ et $\ominus_\mathcal{T}$ deux opérateurs d'addition et de soustraction dans $\mathcal{L}$.*

On appelle information manquante de Rep par rapport à Dem tout concept H tel que Rep n'est pas solution de $\{\mathcal{L}, \mathcal{T}, \mathcal{R} \cup \{Rep \oplus_\mathcal{T} H\}, Dem, \rho, \ll\}$, mais $Rep \oplus_\mathcal{T} H$ l'est. On appelle information superflue de Rep par rapport à Dem tout concept G tel que Rep n'est pas solution de $\{\mathcal{L}, \mathcal{T}, \mathcal{R} \cup \{Rep \ominus_\mathcal{T} G\}, Dem, \rho, \ll\}$, mais $Rep \ominus_\mathcal{T} G$ l'est.

Grâce aux définitions 5 et 6, nous pouvons maintenant présenter les approches existantes de matchmaking dans les LD dans un même cadre. Au préalable, nous rappelons

les différentes classes de correspondance entre une demande Dem et une réponse Rep, définies par Noia et al. (2011), dans l'ordre décroissant de leur intérêt pour la demande, d'une réponse la plus proche de la demande à la plus éloignée :

- Correspondance exacte (exact match) : $\rho(Dem, Rep)$ ssi $Dem \equiv_{\mathcal{T}} Rep$. C'est la correspondance parfaite entre la demande et la réponse, qui sont sémantiquement identiques.
- Correspondance totale (full match) : $\rho(Dem, Rep)$ ssi $Rep \sqsubseteq_{\mathcal{T}} Dem$. La demande est entièrement satisfaite par la réponse. C'est-à-dire que chaque partie de la demande est satisfaite par la réponse, sans que l'inverse soit vrai. Cela correspond à une réponse plus précise que la demande. En effet, si un utilisateur recherche un "métal", lui offrir "fer" est une réponse adéquate.
- Correspondance connectée (plug-in match) : $\rho(Dem, Rep)$ ssi $Dem \sqsubseteq_{\mathcal{T}} Rep$. La demande est un sous-concept de la réponse. On offre à l'utilisateur une réponse plus générale que ce qu'il recherche.
- Correspondance potentielle (potential match) : $\rho(Dem, Rep)$ ssi $Rep \sqcap Dem \not\sqsubseteq_{\mathcal{T}} \bot$. La demande et la réponse possèdent des caractéristiques communes et aucun sous-ensemble de concepts qui s'excluent (quand la logique de description le permet).
- Correspondance disjointe (partielle) (partial match) : $\rho(Dem, Rep)$ ssi $Rep \sqcap Dem \sqsubseteq_{\mathcal{T}} \bot$. Un conflit existe entre la demande et la réponse (quand la logique de description le permet).

S'il est facile de comparer deux réponses possibles à un problème de matchmaking lorsque leurs classes de correspondance sont différentes, il est nécessaire de pouvoir ordonner deux réponses appartenant à la même classe. Une des premières méthodes utilisées dans le matchmaking utilise le principe de la contraction (voir Colucci et al. (2004) et Noia et al. (2011)). L'idée derrière la contraction est l'affaiblissement de la réponse ou de la demande jusqu'à obtenir une compatibilité. Ainsi la contraction vise à passer d'un cas de correspondance disjointe à un cas de correspondance potentielle en retirant des sous-concepts de la description. La demande est divisée en deux ensembles, K (Keep) et G (give-up), avec G optimal selon un critère choisi. Il est alors possible d'ordonner les différentes réponses en fonction de leur G. Par exemple, plus il est grand en taille, moins la proposition est proche.

Definition 7. *Voir Colucci et al. (2004) Soit $\mathcal{L}$ une logique de description, $\mathcal{T}$ une TBox dans $\mathcal{L}$, et Rep et Dem deux concepts $\in \mathcal{T}_{\mathcal{L}}$, où Rep et Dem sont tous deux satisfiables par rapport à $\mathcal{T}$. En supposant que $Rep \sqcap Dem \sqsubseteq_{\mathcal{T}} \bot$, un Problème de Contraction de Concept (Concept Contraction Problem CCP), identifié par $\{\mathcal{L}$, Dem, Rep, $\mathcal{T}\}$ consiste à trouver une paire de concepts $\{G, K\} \in \mathcal{T}_{\mathcal{L}} \times \mathcal{T}_{\mathcal{L}}$ tel que $Rep \equiv_{\mathcal{T}} G \sqcap K$ avec $K \sqcap Dem$ satisfiable dans $\mathcal{T}$. On appelle K une contraction de Rep par rapport à Dem et $\mathcal{T}$. Les meilleures contractions sont celles minimisant la taille de G.*

Ainsi, par rapport à la définition d'un problème de matchmaking, on a :

1. *$\rho(Dem, Rep)$ ssi il existe $(G, K) \in \mathcal{T}_{\mathcal{L}}^2$ tels que $Rep \equiv_T G \sqcap K$ et $Dem \sqcap K \not\sqsubseteq_T \bot$, et*
2. *Rep minimal par rapport à $\ll$ tel que 1. ssi la taille de G est minimale*

On remarque qu'il peut exister plusieurs solutions possibles à un problème de contraction de concept. Par ailleurs, en prenant un opérateur de soustraction $\ominus_{\mathcal{T}}$ tel que $(G \sqcap K) \ominus_{\mathcal{T}} (G) = K$, on peut voir que G représente l'information superflue de *Rep* par rapport à *Dem*. L'information manquante n'est pas explicitée ni exploitée.

Exemple 2. *On considère une logique de description dans laquelle il est possible d'exprimer les axiomes suivants :*

$Acier \sqcap Bois \sqsubseteq \bot$

$PaC \sqsubseteq Instrument$

$Analogique \sqcap Numérique \sqsubseteq \bot$

Si on considère la demande Dem et les deux réponses Rep_1 *et* Rep_2 *suivantes :*

$Dem \equiv Instrument \sqcap Bois \sqcap Analogique$

$Rep_1 \equiv PaC \sqcap Numérique \sqcap Acier$

$Rep_2 \equiv PaC \sqcap Analogique \sqcap Acier$

La réponse au CPP{$\mathcal{L}$*, Dem,* Rep_1*,* $\mathcal{T}$*} est* $G_1 \equiv Acier \sqcap Numérique$ *et* $K_1 \equiv PaC$

La réponse au CPP{$\mathcal{L}$*, Dem,* Rep_2*,* $\mathcal{T}$*} est* $G_2 \equiv Acier$ *et* $K_2 \equiv PaC \sqcap Analogique$

Les conjonctions $Dem \sqcap Rep_1$ *et* $Dem \sqcap Rep_2$ *sont toutes deux en correspondance disjointe. Si on retire de* Rep_1 *Acier et Numérique, la demande est en correspondance potentielle avec* Rep_1 *(*G_1*=Acier* $\sqcap$ *Numérique et* K_1*=PaC). De la même manière, si on retire de Acier de* Rep_2*, la demande est correspondance potentielle avec* Rep_2 *(*G_2*=Acier et* K_2*=PaC* $\sqcap$ *Analogique).*

Comme on a $\mathsf{size}(G_1) > \mathsf{size}(G_2)$*, on privilégie* Rep_2 *à* Rep_1 *comme réponse potentielle de Dem.*

La contraction de concept étend la satisfiabilité, en faisant passer une réponse de la classe "correspondance partielle" à la classe "correspondance potentielle". Cette méthode implique cependant de pouvoir exprimer l'inconsistance, ce qui n'est pas le cas de $\mathcal{EL}$, cette méthode n'est donc pas adaptée à nos besoins.

Une seconde méthode, basée sur le principe de l'abduction et proposée par Noia et al. (2003), Colucci et al. (2004) et Noia et al. (2011), permet d'ordonner cette fois des couples de concepts appartenant au moins à la classe "correspondance potentielle" et "correspondance connectée", ce qui est donc utilisable dans $\mathcal{EL}$ (où tous les concepts appartiennent au minimum à la classe "correspondance potentielle"). La méthode propose d'ordonner les réponses possibles à une demande en fonction du nombre de concepts qu'il faut ajouter à la réponse pour que celle-ci soit subsumée par la demande, en maximisant par rapport à $\sqsubseteq_{\mathcal{T}}$ ou en minimisant par rapport à la taille des concepts ajoutés. Au sein d'une même classe, on peut alors ordonner les réponses de la plus proche à la plus éloignée de la demande. Il est possible d'utiliser l'abduction sur les classes "correspondance totale" et "correspondance exacte", mais le résultat sera systématiquement $\top$, car la réponse est déjà subsumée par la demande dans ces cas-là.

Definition 8 (Voir Colucci et al. (2004))**.** *Soit* $\mathcal{L}$ *une logique de description,* $\mathcal{T}$ *une TBox dans* $\mathcal{L}$*, et Rep et Dem deux concepts* $\in \mathcal{T}_{\mathcal{L}}$*, où Rep et Dem sont tous deux satisfiables par rapport à* $\mathcal{T}$*. Un Problème d'Abduction de Concept (Concept Abduction*

Problem CAP), identifié par {$\mathcal{L}$, Dem, Rep, $\mathcal{T}$} consiste à trouver le concept $H \in \mathcal{T}_{\mathcal{L}}$ tel que $Rep \sqcap H \sqsubseteq_{\mathcal{T}} Dem$ avec $Rep \sqcap H$ satisfiable dans $\mathcal{T}$. On appelle H une hypothèse de Rep par rapport à Dem et $\mathcal{T}$. Les meilleures hypothèses H sont celles qui sont soit maximales par rapport à $\sqsubseteq_{\mathcal{T}}$, soit minimales par rapport à la taille.

Ainsi, par rapport à la définition d'un problème de matchmaking, on a :

1. *$\rho(Dem, Rep)$ ssi il existe $H \in \mathcal{T}_{\mathcal{L}}$ tel que $Rep \sqcap H \sqsubseteq_{\mathcal{T}} Dem$, et*
2. *Rep minimal par rapport à $\ll$ tel que 1. ssi H est maximal par rapport à $\sqsubseteq_{\mathcal{T}}$ ou minimal par rapport à la taille tel que 1.*

Si on prend l'opérateur d'addition $\oplus_{\mathcal{T}} = \sqcap$, on peut voir que H représente l'information manquante de *Rep* par rapport à *Dem* et que l'information superflue n'est ni explicitée ni exploitée.

Exemple 3. *Soit les axiomes d'une TBox $\mathcal{T}$ suivants :*
$PaC \sqsubseteq Instrument$
$Chêne \sqsubseteq Bois$
Soit la demande Dem et les réponses Rep_i suivante :
$Dem \equiv Instrument \sqcap Bois$
$Rep_1 \equiv PaC \sqcap Chêne$
$Rep_2 \equiv PaC \sqcap Chêne \sqcap Analogique$
$Rep_3 \equiv PaC \sqcap Analogique$

La réponse au CAP{$\mathcal{L}$, Dem, Rep_1, $\mathcal{T}$} est $H_1 \equiv \top$
La réponse au CAP{$\mathcal{L}$, Dem, Rep_2, $\mathcal{T}$} est $H_2 \equiv \top$
La réponse au CAP{$\mathcal{L}$, Dem, Rep_3, $\mathcal{T}$} est $H_3 \equiv Bois$

Ainsi, Rep_1 et Rep_2 sont de meilleures réponses que Rep_3. Néanmoins, Rep_2, qui possède de l'information non demandée par la demande (Analogique) n'est pas discriminée par rapport à Rep_1.

L'abduction étend la subsomption, en faisant passer une réponse des classes "correspondance potentielle" et "correspondance connectée" à la classe "correspondance totale". Si elle permet de comparer deux réponses en fonction de leur information manquante, l'abduction ne prend cependant pas en compte les informations superflues.

Enfin la détermination des meilleures couvertures est une méthode de matchmaking proposée dans Benatallah et al. (2003). Elle effectue la réécriture d'une demande *Dem* en la description la plus proche *Rep*, qui est une conjonction de différentes réponses. Ce système de matchmaking s'éloigne de la simple subsomption. Le concept *Rep* s'obtient par l'intermédiaire de deux valeurs à optimiser : le rest et le miss, qui se déterminent par un calcul de différence entre concepts $\ominus_{\mathcal{T}}$. Le rest représente alors les informations manquantes et le miss les informations superflues.

Definition 9 (Rest et miss)**.** *Soit $\mathcal{L}$ une logique de description, $\mathcal{T}$ une TBox dans $\mathcal{L}$, et Rep et Dem deux concepts $\in \mathcal{T}_\mathcal{L}$. On suppose qu'on dispose d'un opérateur $\ominus_\mathcal{T}$: $\mathcal{T}_\mathcal{L} \times \mathcal{T}_\mathcal{L} \to \mathcal{T}_\mathcal{L}$ de différence entre concepts de $\mathcal{L}$. On définit le rest et miss ainsi :*

$rest_{Dem}(Rep) = Dem \ominus_\mathcal{T} Rep$

$miss_{Dem}(Rep) = Rep \ominus_\mathcal{T} Dem$

Une réponse *Rep* à une demande *Dem* par rapport à $\mathcal{T}$ est appelée une couverture de *Dem*. C'est une conjonction de concepts de $\mathcal{T}$ telle que $rest_{Dem}(Rep) \not\equiv_\mathcal{T} Dem$. Une meilleure couverture est alors une couverture qui minimise l'information d'abord dans le rest, puis dans le miss, en maximisant par rapport à $\sqsubseteq_\mathcal{T}$ ou en minimisant par rapport à la taille.

Definition 10 (Meilleure couverture)**.** *Soient $\mathcal{L}$ une logique de description, $\mathcal{T}$ une TBox dans $\mathcal{L}$, $Dem \in \mathcal{T}_\mathcal{L}$ et $\{Rep_i, 1 \leq i \leq n\}$ un ensemble de n concepts de $\mathcal{T}_\mathcal{L}$, $n \geq 1$. Une meilleure couverture de Dem par rapport à $\mathcal{T}$ et $\{Rep_i, 1 \leq i \leq n\}$ est un concept Rep défini par un sous-ensemble couv de $\{Rep_i, 1 \leq i \leq n\}$ tel que $Rep = \bigsqcap_{Rep_j \in couv} Rep_j$, et qui respecte les contraintes suivantes :*

a. $Rep \not\equiv_\mathcal{T} \bot$

b. $rest_{Dem}(Rep) \not\equiv_\mathcal{T} Dem$.

c. $rest_{Dem}(Rep)$ est maximal par rapport à $\sqsubseteq_\mathcal{T}$ ou minimal par rapport à la taille tel que a. et b.

d. $miss_{Dem}(Rep)$ est maximal par rapport à $\sqsubseteq_\mathcal{T}$ ou minimal par rapport à la taille tel que a., b., et c.

Dans le contexte d'un problème de matchmaking $\{\mathcal{L}, \mathcal{T}, \mathcal{R}, Dem, \rho, \ll\}$, on a :

1. $\mathcal{R} = \{\bigsqcap_{Rep_j \in couv} Rep_j | couv \subseteq \{Rep_i, 1 \geq i \geq n\}\}$

2. $\rho(Dem, Rep)$ ssi a. et b.

3. Rep minimal par rapport à $\ll$ tel que 1. ssi c. puis d.

Dans cette approche de matchmaking, si les opérateurs $\oplus_\mathcal{T}$ et $\ominus_\mathcal{T}$ sont bien choisis, on vérifie facilement que :

— $Rep \oplus_\mathcal{T} rest_{Dem}(Rep) = Rep \oplus_\mathcal{T} (Dem \ominus_\mathcal{T} Rep)$ est meilleure que *Rep* par rapport au critère c. On a même que $Rep \oplus_\mathcal{T} (Dem \ominus_\mathcal{T} Rep)$ ne contient pas d'information manquante.

— $Rep \ominus_\mathcal{T} miss_{Dem}(Rep) = Rep \ominus_\mathcal{T} (Rep \ominus_\mathcal{T} Dem)$ est meilleure que *Rep* par rapport au critère d. On a même que $Rep \ominus_\mathcal{T} (Rep \ominus_\mathcal{T} Dem)$ ne contient pas d'information superflue.

Cette méthode de matchmaking offre donc l'avantage de comparer à la fois l'information manquante (le rest) et l'information superflue (le miss), ce qui permet un ordonnancement très précis des réponses possibles. Dans $\mathcal{FL}_0$, $\oplus_\mathcal{T}$ est l'intersection $\sqcap$. Pour $\ominus_\mathcal{T}$, $A \ominus_\mathcal{T} B$ est $A - LCS(A, B)$, où $-$ est la différence de Teege (voir 1.3.1 (p.24)) et $LCS(A, B)$ le plus petit subsumant commun entre A et B (Least Common Subsummer, défini dans l'annexe A.6 (p.124)). Dans l'exemple suivant, nous utilisons notre propre opérateur de différence, présenté au chapitre 3 (p.45).

Exemple 4. *L'opérateur $\ominus_{\mathcal{T}}$ utilisé ici est celui défini dans le chapitre 3 (p.45). Informellement, pour $C \ominus_{\mathcal{T}} D$, cet opérateur retire de C les sous-descriptions qui subsument D. Le critère $\ll$ utilisé ici est la maximalité de la subsomption. Soient les axiomes d'une TBox suivants, avec la demande Dem et les réponses Rep_i :*

$PaC \sqsubseteq Instrument$

$Chêne \sqsubseteq Bois$

$Dem \equiv Instrument \sqcap Bois \sqcap Acier$

$Rep_1 \equiv PaC \sqcap Chêne$

$Rep_2 \equiv PaC \sqcap Chêne \sqcap Analogique$

$Rep_3 \equiv PaC \sqcap Acier$

$rest_{Dem}(Rep_1) = Acier$ car Rep_1 répond à la partie Instrument et Bois, mais pas Acier. $rest_{Dem}(Rep_1) = Dem \ominus_{\mathcal{T}} Rep_1 = Dem$ privé des sous-descriptions Instrument et Bois car toutes deux subsument Rep_1 par rapport à $\mathcal{T}$

$miss_{Dem}(Rep_1) = Chêne \sqcap PaC$ car ces concepts sont plus précis que ceux de la demande.

$rest_{Dem}(Rep_2) = Acier$ car Rep_2 répond à la partie Instrument et Bois mais pas Acier.

$miss_{Dem}(Rep_2) = PaC \sqcap Chêne \sqcap Analogique$ car ces concepts sont plus précis que ceux de la demande ou en sont absents.

$rest_{Dem}(Rep_3) = Bois$ car Rep_1 répond à la partie Instrument mais pas Bois.

$miss_{Dem}(Rep_3) = PaC$ car ce concept est plus précis que celui de la demande.

Si l'on considère les réponses individuellement, les rest de Rep_1 et Rep_2 sont égaux et sont incomparables avec le rest de Rep_3. Donc Rep_1, Rep_2et Rep_3 sont indistinguables par rapport au seul rest. Cependant Rep_3 est meilleure que les deux autres par rapport au miss car Rep_3 a le miss le plus grand par rapport à la subsomption. Les meilleures couvertures considèrent cependant des combinaisons de réponses (par conjonction) :

$rest_{Dem}(Rep_1 \sqcap Rep_2) = Acier$.

$miss_{Dem}(Rep_1 \sqcap Rep_2) = Chêne \sqcap PaC \sqcap Analogique$.

$rest_{Dem}(Rep_3 \sqcap Rep_2) = \top$ car la combinaison répond à toutes les parties de Dem.

$miss_{Dem}(Rep_3 \sqcap Rep_2) = PaC \sqcap Chêne \sqcap Analogique$.

$rest_{Dem}(Rep_1 \sqcap Rep_3) = \top$.

$miss_{Dem}(Rep_1 \sqcap Rep_3) = PaC \sqcap Chêne$.

Ainsi, les combinaisons $(Rep_1 \sqcap Rep_3)$ et $(Rep_2 \sqcap Rep_3)$ sont meilleures que les réponses prises individuellement et meilleures que la combinaison $(Rep_1 \sqcap Rep_2)$, car leur rest est maximal par rapport à la subsomption. Cependant, la combinaison $(Rep_1 \sqcap Rep_3)$ a un miss plus grand que $(Rep_2 \sqcap Rep_3)$ par rapport à la subsomption, c'est donc la meilleure couverture possible à la demande Dem. Si cet exemple illustre le fonctionnement des meilleures couvertures, la combinaison de réponse n'est pas pertinente pour notre domaine d'étude, étant donné que chaque Rep_i décrit une ressource. Initialement, les meilleures couvertures étaient appliquées pour trouver des combinaisons de services web. Dans le domaine de la métrologie, trouver des combinaisons de ressources n'est pas très pertinent.

Les processus de matchmaking présentés ne sont pas adaptés à notre contexte de

recherche dans un domaine précis. En effet, la contraction ne peut pas s'utiliser dans la logique de description souhaitée ($\mathcal{EL}$) car on doit pouvoir exprimer les inconsistances dans la LD choisie, ce qui n'est pas possible avec $\mathcal{EL}$. De plus, elle ne prend pas en compte l'information manquante. À contrario, l'abduction n'est pas en mesure de traiter l'information superflue d'une réponse potentielle, mais seulement l'information manquante. Enfin, les meilleures couvertures donnent des combinaisons de réponses, ce qui n'est pas très pertinent par rapport à notre application. Cependant, nous avons souhaité reprendre le principe de détermination du rest et du miss de Benatallah et al. (2003), qui offre à la fois un classement sur l'information manquante et sur l'information superflue, permettant un classement plus fin. Ce choix nécessite un opérateur de différence pour la LD $\mathcal{EL}$. Les opérateurs de différence existants sont présentés dans la section qui suit.

1.3 Opérateurs de différence dans les logiques de description

Cette section présente les travaux en relation avec la notion de différence entre concepts dans les LD. Les différentes méthodes sont comparées et il est expliqué pourquoi les opérateurs de différence existants ne sont pas satisfaisants dans le cadre de notre méthode de matchmaking. Dans la suite, nous définirons chaque opérateur comme $\ominus_{au}$, où au représente les deux premières lettres du premier auteur de l'article de référence sur l'opérateur en question. De plus, sauf précision contraire, C, D et E sont des concepts de $\mathcal{EL}$ et r un rôle de $\mathcal{EL}$.

Nous prendrons comme exemple les deux concepts suivants :
$Dem = PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$
$Rep = PaC \sqcap Numérique \sqcap \exists aMatériau.Acier$
pour lesquels nous souhaiterions obtenir, à l'image de notre exemple d'introduction :
$Dem \ominus Rep = \exists aMatériau.Plastique$
$Rep \ominus Dem = \top$
Informellement, on souhaite retirer une partie du diminuende (la partie gauche de l'opération de différence) si celle-ci subsume le diminuteur (la partie droite de l'opération de différence). En effet, on considère qu'on répond à une partie de la requête si la réponse apporte un concept équivalent à ou plus précis que cette partie de requête. Par exemple, si un utilisateur demande un outil en métal, lui proposer un outil en fer est une réponse acceptable. Par contre, si un utilisateur demande un outil en fer, lui proposer un outil dans un autre métal ne sera pas une réponse satisfaisante. C'est ce qu'on appelle le critère de subsomption inverse, qui correspond à une situation de correspondance totale dans les classes de correspondance des approches de matchmaking (voir 1.2 (p.18)). À noter qu'en général, les autres opérateurs de différence correspondent à une situation de correspondance connectée.

Après avoir identifié une partie du diminuende qui subsume le diminuteur, on souhaite pouvoir la retirer du diminuende. Cette partie peut être une partie d'une restriction existentielle. On souhaite donc la retirer sans supprimer le reste de la restriction existentielle. Dans l'exemple, c'est la capacité à retirer $\exists aMatériau.Acier$ de Dem sans retirer

$\exists aMatériau.Plastique$: la réponse *Rep*, qui décrit un PaC en acier, répond en partie à la demande. C'est le critère de fine granularité, qui permet de ne supprimer que ce qui doit l'être, de manière plus intuitive pour l'utilisateur.

On va maintenant examiner les approches existantes en essayant notamment de voir si les deux critères de subsomption inverse et de fine granularité sont proposés par ces approches.

1.3.1 Approche de Teege (1994)

Teege (1994) propose un opérateur sémantique dont le résultat, pour $C \ominus_{Te} D$, est le concept le plus général, qui en conjonction avec D, est équivalent à C. L'opérateur n'est pas défini si $C \not\sqsubseteq D$ et ne prend en compte aucune TBox. Une définition plus formelle se trouve dans l'annexe A.1 (p.119).

Exemple 5. *On rappelle que :*
$Dem = PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$.
$Rep = PaC \sqcap Numérique \sqcap \exists aMatériau.Acier$.
$Rep \ominus_{Te} Dem$ *n'est pas défini car* $Rep \not\sqsubseteq Dem$.
$Dem \ominus_{Te} Rep = \exists aMatériau.(Acier \sqcap Plastique)$.

Avec $\ominus_{Te}$, il n'est pas possible de retirer des concepts à l'intérieur de restrictions existentielles. Cette limitation vient d'une propriété des restrictions existentielles : $\exists r.C \sqcap \exists r.D \not\equiv \exists r.(C \sqcap D)$.

Exemple 6. *Considérons un autre concept* $Rep2 = \exists aMatériau.Acier$. *Avec le calcul* $Dem \ominus_{Te} Rep2$, *on souhaite retirer Acier de Dem. On aimerait donc obtenir* $PaC \sqcap Numérique \sqcap \exists aMatériau.Plastique$ *comme résultat. Or,* $Dem \ominus_{Te} Rep2 = Dem$. *En effet, il n'est pas possible de trouver un concept* $C \not\equiv Dem$ *tel que* $C \sqcap \exists aMatériau.Acier \equiv Dem$.

Par rapport à notre exemple, l'opérateur de Teege ne semble pas être adapté.

1.3.2 Approche de Brandt et al. (2002)

Brandt et al. (2002) propose un opérateur sémantique qui ne prend en compte aucune TBox et qui nécessite au préalable la normalisation des concepts consister à retirer les redondances. L'opérateur est défini pour un C dans $\mathcal{ALC}$ et un D dans $\mathcal{ALE}$. Ces deux logiques de descriptions sont présentées dans Brandt et al. (2002). $\mathcal{EL}$ est contenue dans ces deux logiques de description. Il est donc possible d'utiliser cet opérateur dans $\mathcal{EL}$. L'utilisation de cet opérateur dans $\mathcal{EL}$ retire la nécessité de normaliser les concepts étudiés, cette nécessité venant de la présence du constructeur $\forall$. Il est cependant nécessaire d'appliquer un traitement pour retirer les redondances (voir la section annexeA.2 (p.119)). L'opérateur cherche ensuite le résultat E de $C \ominus_{Br} D$, qui en conjonction avec D, est équivalent à $C \sqcap D$. Le résultat de $C \ominus_{Br} D$ est un concept obtenu en retirant de C tous les sous-concepts redondants ou présents dans D. Une définition plus formelle se trouve dans l'annexe A.2 (p.119).

Exemple 7. *On rappelle que*
$Dem = PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$.
$Rep = PaC \sqcap Numérique \sqcap \exists aMatériau.Acier$.
On a alors :
$Dem \ominus_{Br} Rep = \exists aMatériau.(Acier \sqcap Plastique)$.
$Rep \ominus_{Br} Dem = \top$.

Il n'est encore une fois pas possible de retirer des concepts à l'intérieur de restrictions existentielles. C'est-à-dire que $Dem \ominus_{Br} Rep$ ne donne pas $\exists aMatériau.Plastique$ comme on le souhaiterait.

1.3.3 Approche de Suchanek et al. (2016)

Suchanek et al. (2016) définit un opérateur qui nécessite également une normalisation et un ordre syntaxique total $\prec$, ainsi que la suppression des redondances (voir annexe A.3 (p.120) pour plus de détail). Informellement, l'opérateur retire de C le premier conjonct (de la gauche vers la droite) qui est subsumé par le premier conjonct de D, les conjoncts de C et de D étant ordonnés selon $\prec$. L'opération se répète ensuite avec le second conjonct de D et ainsi de suite jusqu'à avoir examiné chaque conjonct de D dans l'ordre. Une définition plus formelle se trouve dans l'annexe A.3 (p.120).

Exemple 8. *Avec l'ordre $\prec$ définit dans A.3 (p.120) et avec $\prec_{N_C}$ et $\prec_{N_R}$ l'ordre alphabétique. On rappelle que*
$Dem = PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$
$Rep = PaC \sqcap Numérique \sqcap \exists aMatériau.Acier$
Leur normalisation est alors :
$norm(Dem) = Numérique \sqcap PaC \sqcap \exists aMatériau.(Acier \sqcap Plastique)$
$norm(Rep) = Numérique \sqcap PaC \sqcap \exists aMatériau.Acier$
Le résultat des opérations de différence est alors :
$Dem \ominus_{Su} Rep = \top$
$Rep \ominus_{Su} Dem = \exists aMatériau.Acier$

Il n'est encore une fois pas possible de retirer des concepts à l'intérieur de restrictions existentielles. C'est-à-dire que $Dem \ominus_{Su} Rep$ ne donne pas $\exists aMatériau.Plastique$ De plus, l'opérateur effectue la différence dans le sens inverse de celui souhaité, c'est-à-dire qu'il retire des concepts subsumés par D et non pas subsumant D, dans $C \ominus_{Su} D$. C'est-à-dire que $Rep \ominus_{Su} Dem$ ne donne pas $\top$.

1.3.4 Approche de Heinz (2018)

Heinz (2018) propose un opérateur plus complexe qui nécessite également une normalisation. Les deux concepts étudiés sont rangés dans un ordre total $\prec$, défini à partir de deux ordres $\prec_{N_C}$ et $\prec_{N_\nabla}$ qui définissent l'importance de l'information et qui sont donnés par un expert. L'opérateur de différence de Heinz étant défini par rapport à une TBox définitionnelle et acyclique, chaque concept est entièrement étendu. L'opération

$C \ominus_{He} D$ consiste à retirer de C le plus petit conjonct présent à la fois dans C et dans D, à la condition que $C \sqsubseteq D$. L'opérateur est également capable, si nécessaire, de retirer de l'information à l'intérieur des rôles. Une définition plus formelle se trouve dans l'annexe A.4 (p.120).

Exemple 9. *On suppose qu'on a l'ordre $\prec$ défini dans A.4 (p.120) et $\prec_{N_C}$ et $\prec_{N_\nabla}$ l'ordre alphabétique. On rappelle que :*
$Dem = PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$.
$Rep = PaC \sqcap Numérique \sqcap \exists aMatériau.Acier$.
Leur normalisation est alors :
$norm(Dem) = \exists aMatériau.(Acier \sqcap Plastique) \sqcap Numérique \sqcap PaC$.
$norm(Rep) = \exists aMatériau.Acier \sqcap Numérique \sqcap PaC$.
Le résultat des opérations de différence est alors :
$Dem \ominus_{He} Rep = \exists aMatériau.Plastique \sqcap Numérique \sqcap PaC$
$Rep \ominus_{He} Dem = \exists aMatériau.Acier \sqcap Numérique \sqcap PaC$

Exemple 10. *Un second exemple est réalisé pour montrer l'application de l'opérateur sur des conjonctions de concepts primitifs A, B et F. Avec l'ordre $\prec$ définit dans l'annexe A.4 (p.120) et avec $\prec_{N_C}$ et $\prec_{N_\nabla}$ l'ordre alphabétique. Soit $norm(C) = A \sqcap B \sqcap F$ et $norm(D) = B \sqcap F$, alors $C \ominus_{He} D = A \sqcap F$.*

Si l'opérateur est bien capable de retirer des concepts à l'intérieur des restrictions existentielles, il n'est pas capable d'effectuer de différence entre deux concepts n'ayant pas de relation de subsomption. Par exemple, $\exists aMatériau.Acier \ominus_{He} \exists aMatériau.(Acier \sqcap Plastique) = \exists aMatériau.Acier$. De plus, il ne retire du diminuende qu'un seul conjonct du diminuteur. Dans l'exemple 10, $A \sqcap B \sqcap F \ominus_{He} B \sqcap F = A \sqcap F$ et non pas seulement A .

1.3.5 Approche de Rienstra et al. (2020)

Rienstra et al. (2020) proposent un opérateur $\ominus_{Ri}$ de contraction dans $\mathcal{EL}$ qui généralise A, lorsqu'on réalise $C \ominus_{Ri} D$. Le résultat est le concept le plus proche de C possible, par rapport à une fonction de sélection, tel qu'il ne soit pas subsumé par D. Si $C \not\sqsubseteq_{\mathcal{T}} D$, $C \ominus_{Ri} D = C$. La TBox est acyclique et définitionnelle. Les résultats sont appelés résidus (remainder) de C par rapport à D. Une définition plus formelle se trouve dans l'annexe A.5 (p.121).

Exemple 11. *On rappelle que :*
$Dem = PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$.
$Rep = PaC \sqcap Numérique \sqcap \exists aMatériau.Acier$.
et on souhaiterait obtenir :
$Dem \ominus Rep = \exists aMatériau.Plastique$.
$Rep \ominus Dem = \top$
Parmi les fonctions de sélections σ proposées dans Rienstra et al. (2020), celle donnant

les résultats les plus proches de ce que nous recherchons est le full meet. Voir A.5 (p.121) pour plus de précision sur la détermination des résultats.

$Dem \ominus_{Ri} Rep = \exists aMatériau.Plastique.$

$Rep \ominus_{Ri} Dem = \exists aMatériau.Acier \sqcap Numérique \sqcap PaC.$

L'opérateur ne retire pas de concept de A dans $A \ominus_{Ri} B$ si $A \not\sqsubseteq_{\mathcal{T}} B$. De plus, s'il est en mesure de retirer des concepts à l'intérieur des rôles, il ne le fait pas de la manière souhaitée, voir l'exemple si dessous.

Exemple 12. $\exists aMatériau.(Acier \sqcap Plastique \sqcap Aluminium) \ominus_{Ri} \exists aMatériau.(Acier \sqcap Plastique) = \exists aMatériau.(Acier \sqcap Aluminium) \sqcap \exists aMatériau.(Plastique \sqcap Aluminium)$
Le résultat souhaité étant : $\exists aMatériau.Aluminium$

Enfin, l'opérateur effectue la différence dans le sens inverse de celui souhaité, c'est-à-dire que dans $A \ominus_{Ri} B$, il retire des sous-descriptions subsumées par B et non pas subsumant B.

Exemple 13. *Avec* $Acier \sqsubseteq Métal$

$Acier \ominus_{Ri} Métal = \top$

$Métal \ominus_{Ri} Acier = Métal$

Le résultat souhaité étant :

$Acier \ominus Métal = Acier$

$Métal \ominus Acier = \top$

1.3.6 Synthèse

On vient de voir que les opérateurs de différences existants ne nous permettent pas d'obtenir le résultat souhaité. Pour une vision synthétique, on propose les 4 critères suivants, nous permettant de comparer les opérateurs dans le tableau 1.2 (p.28) :

- Leur type ① : S pour une soustraction (on retire du diminuende) ou C pour une complétion (on ajoute au diminuteur)
- Nature de la définition ② : T pour syntaxique et M pour sémantique, B pour les deux
- Leur critère d'optimalité ③ : T pour syntaxique et M pour sémantique ou − si non applicable
- La capacité à retirer des concepts des restrictions existentielles ④ : O pour Oui et N pour Non

La table en annexe 1.2 (p.28) fait une synthèse similaire avec les définitions formelles des opérateurs.

L'opérateur CSO que nous définissons à la section 3.1.1 (p.46) présente les propriétés suivantes :

- ① : Basé sur la soustraction
- ② : Sémantique et Syntaxique
- ③ : Pas besoin de critère d'optimalité
- ④ : L'opérateur est capable d'effectuer des opérations à l'intérieur de restrictions existentielles.

TABLE 1.2 – Comparaison des opérateurs de différence dans les LD.

Référence et principe informel	①	②	③	④	Remarques et complexité
Teege (1994) : $C \ominus_{Te} D$ est le concept maximal par rapport à $\sqsubseteq$ qui, si ajouté à D par conjonction, donne un concept équivalent à C.	C	M	M	N	• Défini pour toute LD $\mathcal{L}$. • Non défini si $C \not\sqsubseteq D$. • Non étudié en présence de TBox. • La complexité n'est pas indiquée.
Brandt et al. (2002) : $C \ominus_{Br} D$ est l'ensemble minimal des concepts par rapport à un ordre des sous-descriptions tel que, si ajoutés à D par conjonction, donne un concept équivalent à $C \sqcap D$.	C	M	T	N	• C est dans $\mathcal{ALC}$, D est dans $\mathcal{ALE}$. • Non étudié en présence de TBox. • PTIME par rapport aux tailles de C et D, avec un oracle pour la détermination de la subsomption.
Suchanek et al. (2016) : $C \ominus_{Su} D$ retire les conjoncts minimaux selon un ordre syntaxique sur les concepts $\mathcal{EL}$, qui sont subsumés par des conjoncts de D (on retire un conjonct de C pour un conjonct D).	S	B	–	N	• Défini pour $\mathcal{EL}$. • $\mathcal{T}$ est acyclique. • EXPTIME par rapport aux tailles de $\mathcal{T}$, C et D et PTIME par rapport aux tailles de $\mathcal{T}^*(C)$ et $\mathcal{T}^*(D)$.
Heinz (2018) : $C \ominus_{He} D$ retire les conjoncts de C qui sont subsumés par le plus petit conjonct de D, par rapport à un ordre syntaxique total. Le procédé est récursif à l'intérieur des restrictions existentielles.	S	B	–	Y	• Défini pour $\mathcal{EL}$. • $\mathcal{T}$ est définitionnelle et acyclique. • $C \ominus_{He} D = C$ si $C \not\sqsubseteq D$. • La complexité n'est pas indiquée.
Rienstra et al. (2020) : En étendant la notion de sous-description pour inclure les concepts obtenus après avoir remplacé un nom de concept par $\top$, $C \ominus_{Ri} D$ est l'unique sous-description (mode full meet) ou l'ensemble des sous-descriptions (mode maxi-choice) S de C qui sont minimales par rapport à $\sqsubseteq$ tel que $S \not\sqsubseteq D$.	S	M	–	Y	• Défini pour $\mathcal{EL}$. • $\mathcal{T}$ est définitionnelle et acyclique. • $C \ominus_{Ri} D = C$ si $C \not\sqsubseteq D$. • La complexité n'est pas indiquée.
L'opérateur proposé dans la section 3 : $C \ominus_{\mathcal{T}} D$ retire les descriptions communes par rapport à $\mathcal{T}$ de C vers D, C'est-à-dire les branches caractéristiques de C qui subsument D.	S	B	M	Y	• Défini pour $\mathcal{EL}$. • $\mathcal{T}$ est définitionnelle et acyclique. • EXPTIME par rapport aux tailles de $\mathcal{T}$, C et D et par rapport aux tailles de $\mathcal{T}^*(C)$ et $\mathcal{T}^*(D)$.

1.4 Principe de base de comparaison multicritère

Dans cette section, nous présentons les principes de la comparaison multicritère, les deux différentes approches utilisées pour la comparaison et les raisons que nous avons d'en choisir une plutôt que l'autre.

Le découpage des demandes et réponses en différentes composantes, que nous définissons dans la section 2 (p.35) permet de voir notre problème de recommandation comme une approche de matchmaking multicritère. C'est-à-dire que nous cherchons une solution optimum en fonction de différents objectifs. Ici, notre objectif est le même pour toutes les composantes : obtenir la plus grande proximité sémantique entre une composante de la demande et la réponse.

Pour rappel, selon Marquis et al. (2014), un problème de décision multicritère implique la prise en compte de plusieurs objectifs à optimiser simultanément afin d'arriver à une solution optimale (pour nous, la proximité avec la demande), choisie parmi un ensemble A d'alternatives (pour nous, les réponses). On définit ensuite un ensemble fini de critères (pour nous, les composantes) $N = \{1, ..., n\}$ prenant la forme de fonctions objectifs $f_i, i \in N$, avec $f_i : A \to X_i$. Pour tout $x \in A$ et tout $i \in N$, on appelle performance de x sur le critère i la quantité $f_i(x)$ reflétant la valeur de x par rapport au critère i, avec $x \in X_i$. L'ensemble $\mathcal{X} = X_1 \times ... \times X_n$ constitue un espace de description des alternatives, appelé espace des critères. La solution x est donc représentée dans $\mathcal{X}$ par le vecteur $(x_1, ..., x_n)$, avec $x_i = f_i(x)$.

Ainsi, un problème de décision multicritère est un triplet de la forme :

$(A, \{f_1, ...f_n\}, Q)$ où $Q \in$ {choix, rangement, tri} est la question formulée.

Q représente la problématique d'analyse multicritère :

- Choix consiste à trouver la solution qui optimise au mieux les différents critères, ou un sous-ensemble de solutions aussi réduit que possible contenant les meilleures solutions.
- Rangement consiste à ordonner, au moins partiellement, l'ensemble des solutions selon leur performance sur les critères étudiés.
- Tri consiste à placer chaque solution dans des catégories prédéfinies.

La comparaison des différentes alternatives passe alors par l'utilisation d'une règle d'agrégation. C'est une fonction qui, pour chaque paire d'alternatives (x, y) de $A \times A$ définit la préférence $x \succsim y$ (x est meilleure que y) sur la base des vecteurs de performances $(x_1, ..., x_n)$ et $(y_1, ..., y_n)$:

$x \succsim y$ ssi $h(x_1, ..., x_n, y_1, ..., y_n) \geq 0$ où h est une fonction à valeurs réelles définie sur $\mathbb{R}^{2n}$ croissante des n premiers arguments et décroissante des n derniers arguments et telle que $h(x, x) \geq 0$ pour tout $x \in \mathbb{R}^n$.

La fonction h réalise les deux opérations cruciales des problèmes multicritères : l'agrégation et la comparaison des solutions. En général, ces deux étapes sont clairement distinguées et h est alors la combinaison d'une fonction d'agrégation, qui permet de synthétiser un vecteur de n performances en un scalaire et une fonction de comparaison

qui permet de comparer deux performances données. On distingue alors deux modes opératoires différents :

- Agréger puis Comparer (AC), qui agrège d'abord les performances de chaque alternative pour ensuite comparer les résultats pour définir la ou les solutions.
- Comparer puis agréger (CA), qui compare chaque performance individuellement pour chaque paire d'alternatives avant d'agréger les résultats pour définir la ou les solutions.

Plus précisément, l'approche AC consiste à résumer toute solution x par une note globale $f(x)$ calculée à partir de son vecteur de performances. Cette note résume la valeur globale de l'alternative et sert de base à la comparaison avec les autres solutions. C'est, par exemple, la manière de comparer deux élèves avec leur moyenne générale. L'approche CA consiste à comparer, critère par critère, les performances des alternatives deux à deux. On construit donc autant de relations de préférences que de critère pour chaque paire, puis on agrège ces résultats pour obtenir un ordre général. C'est, par exemple, le procédé utilisé dans plusieurs méthodes de détermination d'un meilleur candidat lors d'un vote dans la théorie du choix social.

Nous avons choisi d'utiliser l'approche CA. En effet, la méthode AC nous impose de transformer en valeur numérique la proximité sémantique d'une solution qu'on a obtenue sous la forme de concepts. Il serait dommage de faire cela, car nous perdrions la capacité de comparer sémantiquement les réponses entre elles. L'approche CA nous offre cette possibilité, par exemple au travers de la Concordance Relative. Ainsi, pour la composante $aMatériau$, si on compare une réponse Rep_{Fer} dont le matériau est le Fer et une réponse Rep_{Bois} dont le matériau est le $Bois$ par rapport à une demande dont le matériau est le $Métal$, on peut utiliser le fait que $Fer \sqsubseteq_{\mathcal{T}} Métal$ pour dire que Rep_{Fer} est meilleure que Rep_{Bois} pour le critère $aMatériau$. Ce n'est qu'après l'ensemble des comparaisons sémantiques réalisées qu'on agrège les résultats à l'aide de coefficients numériques.

Concordance Relative Pour chaque paire d'alternatives $(x, y) \in A \times A$ et chaque critère i, on définit un indice binaire de préférence partielle $\phi_i(x, y)$, où ϕ_i est une fonction croissante des x_i, décroissante des y_i. La préférence $x \succsim y$ est alors définie par l'agrégation des ϕ_i. Puisque $\phi_i(x, y)$ ne dépend que de x_i et y_i, alors elle définit une relation de préférence nette, c'est-à-dire qu'on peut dire que x est meilleure que y sur le critère i, qui permet d'ordonner les solutions d'après i uniquement. On obtient des pré-ordres complets pour chaque critère i en posant $x \succsim_i y$ si et seulement si $\phi_i(x, y) \geq 0$. Par exemple, c'est ce qu'on obtient avec :

$$\phi_i(x, y) = \left\{ \begin{array}{lll} 1 & si & x_i > y_i \\ 0 & si & x_i = y_i \\ -1 & si & x_i < y_i \end{array} \right\} \tag{1.1}$$

Il suffit ensuite d'agréger les scores de chaque $\phi_i(x, y)$ pour obtenir un score relatif $c(x, y)$ de x par rapport y pour tous les n critères. Par exemple en faisant la somme des $\phi_i(x, y)$:

$$c(x,y) = \sum_{i=0}^{n} \phi_i(x,y) \tag{1.2}$$

Dans le cas (1.2), on a clairement $c(x,y) = -c(y,x)$. Et on pourra alors dire que x est meilleure que y ssi $c(x,y) \geq c(y,x)$. Pour obtenir le score global de x par rapport à toutes les autres alternatives, on fait par exemple la somme pour toutes les autres alternatives y de $c(x,y)$:

$$score(x) = \sum_{y \in A, y \neq x} c(x,y) \tag{1.3}$$

Le score nous sert alors à choisir, ranger ou trier les solutions entre elles : $x \succsim y$ ssi score$(x) \geq$ score(y). Dans notre contexte, les critères i sont nos composantes. Dans l'exemple 1 (p.4), on retrouve la concordance relative dans le troisième calcul de score où les composantes sont prises en compte. Chaque composante est considérée comme un critère, chaque réponse est une alternative. Les ϕ_i sont déterminés en fonction du nombre d'informations manquantes et superflues de chaque réponse. Dans l'exemple, on avait obtenu l'ordre suivant pour la composante type d'instrument et Dem_1 :

— Composante type d'instrument :
Pour Dem_1 : $Rep_2 = Rep_1 > Rep_3 > Rep_4$,

À partir de cet ordre on peut déterminer les ϕ_i, par exemple $\phi_{Type}(Rep_1, Rep_3) = 1$ car $Rep_1 >_{Type} Rep_3$.

Pour mettre en avant l'importance de l'ordre des réponses pour chaque critères, on peut calculer les scores par critères de chaque alternative :

$$score_i(x) = \sum_{y \in A, y \neq x} \phi_i(x,y) \tag{1.4}$$

Le score global d'une alternative x devient :

$$score(x) = \sum_{i=0}^{n} score_i(x) \tag{1.5}$$

Les résultats obtenus avec les formules 1.3 et 1.5 sont les mêmes, mais nous préférons le calcul des $score_i(x)$ pour étudier plus facilement les différences sur les différentes composantes dans les exemples qui suivent.

Enfin, l'utilisation de composante donne à l'expert la possibilité de donner des coefficients à une composante, en fonction de l'importance de celle-ci dans la TBox. Ainsi, l'équation de comparaison d'une composante avec une autre devient, avec $coef$ le coefficient de composante :

$$\phi_i(x,y) = \left\{ \begin{array}{lll} 1 \times coef & si & x_i > y_i \\ 0 & si & x_i = y_i \\ -1 \times coef & si & x_i < y_i \end{array} \right\} \tag{1.6}$$

L'exemple 14 reprend l'exemple 1 (p.4) et le classement final des différentes réponses pour la demande dem_2. L'utilisation de la concordance relative avec et sans coefficient y est présenté :

Exemple 14. *On rappelle que dans l'exemple 1 (p.4) :*

$Dem_2 \equiv \exists aType.Micromètre \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne) \sqcap \exists aFabricant.Fab - Européen.$

$Rep_1 \equiv \exists aType.PaC \sqcap \exists aMode.Analogique \sqcap \exists aUnité.Cm \sqcap \exists aFabricant.Fab - Allemand$

$Rep_2 \equiv \exists aType.PaC \sqcap \exists aMatériau.Plastique \sqcap \exists aMode.Numérique \sqcap \exists aFabricant.Fab - Français \sqcap \exists aUnité.Mm.$

$Rep_3 \equiv \exists aType.Micromètre \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Acier) \sqcap \exists aFabricant.Fab - Français$

$Rep_4 \equiv \exists aType.Règle \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne) \sqcap \exists aUnité.Mm \sqcap \exists aFabricant.Fab - Européen$

En reprenant le dernier classement (prenant en compte les informations manquantes, puis affinant avec les informations superflues) de l'exemple 1 on a :

- *Composante type d'instrument :*
  Pour Dem_2 *:* $Rep_3 >_{Type} Rep_2 =_{Type} Rep_1 >_{Type} Rep_4$

- *Composante matériau :*
  Pour Dem_2 *:* $Rep_4 >_{Matériau} Rep_3 >_{Matériau} Rep_2 >_{Matériau} Rep_1$

- *Composante lieu de Fabrication :*
  Pour Dem_2 *:* $Rep_4 >_{Fabricant} Rep_1 =_{Fabricant} Rep_2 =_{Fabricant} Rep_3$

- *Composante unité :*
  Pour Dem_2 *:* $Rep_3 >_{Unité} Rep_4 =_{Unité} Rep_1 =_{Unité} Rep_2$

- *Composante mode de Lecture :*
  Pour Dem_2 *:* $Rep_4 =_{ModeLecture} Rep_3 =_{ModeLecture} Rep_1 >_{ModeLecture} Rep_2$

En donnant un score en fonction de chaque ordre selon la méthode de la concordance relative. Par exemple pour Rep_1 *pour la composante type d'instrument et* Dem_2 *:* $\phi_{Type}(Rep_1, Rep_2)$ *= 0 car* $Rep_1 =_{Type} Rep_2$*.* $\phi_{Type}(Rep_1, Rep_3)$ *= -1 car* $Rep_1 <_{Type} Rep_3$*.* $\phi_{Type}(Rep_1, Rep_4)$ *= 1 car* $Rep_1 >_{Type} Rep_4$*. et son score relatif pour la composante type d'instrument est donc de* $score_i(Rep_1) = 0$*.*

On effectue ce traitement pour chaque réponse et chaque composante et on obtient alors les scores suivants :

$Score_{Dem_2}(Rep_1)$*= 0 - 3 - 1 - 1 + 1 = -4*
$Score_{Dem_2}(Rep_2)$*= 0 - 1 - 1 - 1 - 3 = -6*
$Score_{Dem_2}(Rep_3)$*= 3 + 1 - 1 + 3 + 1 = 7*
$Score_{Dem_2}(Rep_4)$*=-3 + 3 + 3 - 1 + 1 = 3*
L'ordre global sans coefficient est alors :

Pour Dem_2 *:* $Rep_3 \succsim Rep_4 \succsim Rep_1 \succsim Rep_2$

Avec un coefficient de 4 pour le type d'instrument, et de 2 pour le mode de lecture, on obtient les scores suivants :

$Score_{Dem_2}(Rep_1)$*= 0 - 3 - 1 + 0 + 2 = -2*

$Score_{Dem_2}(Rep_2)$*= 0 - 1 - 1 - 3 - 6 = -11*

$Score_{Dem_2}(Rep_3)$*= 12 + 1 - 1 + 3 + 2 = 17*

$Score_{Dem_2}(Rep_4)$*=-12 + 3 + 3 + 0 + 2 = -3*

L'ordre global est alors :

Pour Dem_2 *:* $Rep_3 \succsim Rep_1 \succsim Rep_4 \succsim Rep_2$

L'introduction de coefficients pour les composantes les plus importantes permet d'éviter ici de proposer un instrument trop différent de celui demandé par l'utilisateur.

Chapitre 2

Component Comparison Operator : matchmaking multicritère dans $\mathcal{EL}$

Notre objectif est de classer des réponses par rapport à une demande utilisateur, avec les réponses et la demande qui sont structurées en différentes composantes. Comme évoqué dans l'exemple de l'introduction et dans le chapitre précédent, les approches existantes de matchmaking sémantique ne sont pas totalement adaptées, au contraire de la méthode de concordance relative en comparaison multicritère (avec chaque composante qui est un critère). Nous voyons donc dans ce chapitre comment adapter la concordance relative en un processus de matchmaking sémantique.

Dans un premier temps, nous justifions la structure de l'information en composantes dans la section 2.1. Nous donnons ensuite notre approche multicritère sous la forme d'un opérateur appelé Component Comparison Operator dans la section 2.2. Nous donnons son algorithme et les propriétés associées.

Dans ce chapitre, on suppose que l'on a deux opérateurs de concepts définis par rapport à une TBox $\mathcal{T}$, $\ominus_{\mathcal{T}}$ et $\oplus_{\mathcal{T}}$. L'opérateur $\ominus_{\mathcal{T}}$ a comme résultat un concept unique. On prendra $\oplus_{\mathcal{T}} = \sqcap$ et $\ominus_{\mathcal{T}}$ sera défini et étudié dans le chapitre 3 (p.45).

2.1 Composante

La structuration en composantes esquissée en introduction permet une comparaison des réponses par rapport à une demande, équilibrée et ajustable en fonction des besoins du domaine. Il est facile de le constater avec l'exemple 15. Dans cette section, on utilisera

- $x \succsim_{composante} y$ pour signifier que $\phi_{composante}(x, y) = 1$, c'est-à-dire que x est meilleure que y sur cette composante,
- $x =_{composante} y$ pour signifier que $\phi_{composante}(x, y) = 0$, c'est-à-dire que x est équivalente à y sur cette composante,
- $x \precsim_{composante} y$ pour signifier que $\phi_{composante}(x, y) = -1$, c'est-à-dire que x est

moins bonne que y sur cette composante.

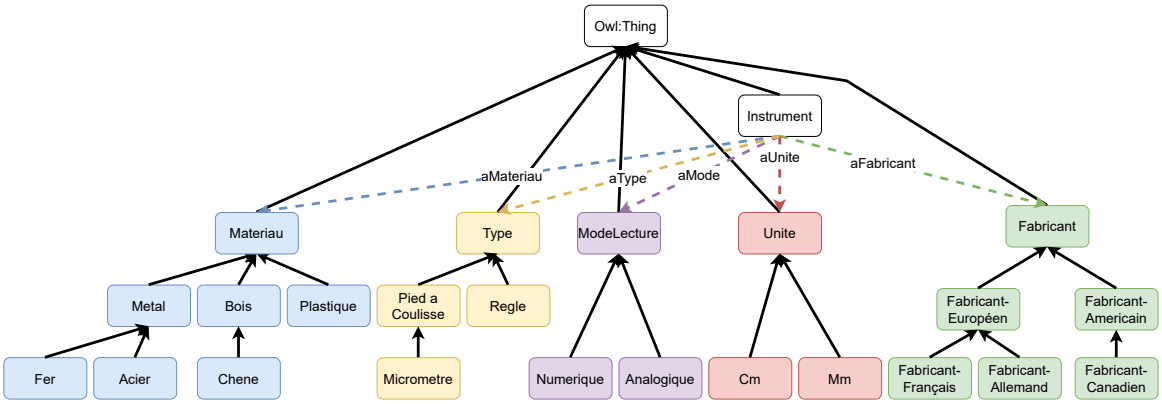


FIGURE 2.1 – Rappel de la TBox $\mathcal{T}_{ex}$ de l'exemple 1 (p.4).

Exemple 15. *Soit la TBox $\mathcal{T}_{ex}$ de l'exemple donné en introduction. Considérons :*

- *la TBox $\mathcal{T}_{ex2} = \mathcal{T}_{ex} \cup \{Aluminium \sqsubseteq Métal, Titane \sqsubseteq Métal\}$*
- *la demande $Dem_3 = \exists aType.Micromètre \sqcap \exists aMode.Numérique \sqcap \exists aMatériau.(Plastique \sqcap Acier \sqcap Aluminium \sqcap Titane \sqcap Chêne)$*
- *les réponses :*
  - *$Rep_5 = \exists aType.Micromètre \sqcap \exists aMode.Numérique$*
  - *$Rep_6 = \exists aMatériau.(Plastique \sqcap Acier \sqcap Aluminium \sqcap Titane \sqcap Chêne)$*

Supposons dans un premier temps que l'on compare les deux réponses comme un problème de meilleure couverture (voir Benatallah et al. (2003)). L'information manquante de Rep_6 est $\exists aType.Micromètre \sqcap \exists aMode.Numérique$ et elle est de taille 4 : $\mathsf{size}(\exists aType.Micromètre \sqcap \exists aMode.Numérique) = 4$. L'information manquante de Rep_5 est $\exists aMatériau.(Plastique \sqcap Acier \sqcap Aluminium \sqcap Titane \sqcap Chêne)$ et est de taille 6 : $\mathsf{size}(\exists aMatériau.(Plastique \sqcap Acier \sqcap Aluminium \sqcap Titane \sqcap Chêne)) = 6$. Rep_6 est donc meilleure que Rep_5.

Dans un second temps on peut structurer nos réponses en trois composantes, selon les rôles $aType$, $aMatériau$ et $aMode$. Chaque composante devient un critère de comparaison dans le but d'obtenir un rangement multicritère selon la méthode de la concordance relative. Pour chaque composante, on considère le problème de matchmaking restreint au concept qui suit le rôle de la composante correspondante. Dans l'exemple suivant, on considère que $Rep_i \succsim_r Rep_j$ si $\mathsf{size}_{Rep_i}(informationManquante) < \mathsf{size}_{Rep_j}(informationManquante)$. On obtient les préférences suivantes :

- *Composante $aType$: $Rep_5 \succsim_{aType} Rep_6$ car*
  - *Rep_5 n'a pas d'information manquante, pour la composante $aType$ car $\exists aType.Micromètre$ est présent dans $Dem3$ et $Rep5$.*
  - *L'information manquante de Rep_6 dans la composante $aType$ est $Micromètre$ qui est de taille 1.*

— *Composante aMatériau : $Rep_6 \succsim_{aMatériau} Rep_5$ car*
  — *L'information manquante de Rep_5 dans la composante $aMatériau$ est de taille 9 :* $\mathsf{size}(Plastique \sqcap Acier \sqcap Aluminium \sqcap Titane \sqcap Chêne) = 5$
  — *Rep_6 n'a pas d'information manquante.*
— *Composante aMode : $Rep_5 \succsim_{aMode} Rep_6$ car*
  — *Rep_5 n'a pas d'information manquante.*
  — *L'information manquante de Rep_6 dans la composante $aMode$ est de taille 1 :* $\mathsf{size}(Numérique) = 1$

Comme Rep_5 est préférable à Rep_6 pour deux composantes sur trois, Rep_5 est préférable à Rep_6. L'utilisation de composantes permet d'affiner la comparaison pour ne pas favoriser une partie de réponse par rapport à une autre. De plus, on pourrait facilement affiner encore plus en pondérant chaque composante par un coefficient (voir l'exemple 14 (p.32)).

Les composantes sont obtenues en structurant la TBox avec l'expert du domaine. On définie la notion de composante ainsi :

Definition 11 (Composante)**.** *Étant donnée une TBox $\mathcal{T}$ et E un nom de concept de $\mathcal{T}$, la composante $\mathcal{C}_E$ est l'ensemble des concepts qui sont subsumés par E dans $\mathcal{T}$. E est appelée le top concept de $\mathcal{C}_E$. Pour chaque composante $\mathcal{C}_E$ d'une TBox $\mathcal{T}$, il existe un rôle $r_{\mathcal{C}_E}$ dit "rôle de composante" associé à $\mathcal{C}_E$. On dira aussi que $\mathcal{C}_E$ est associée à $r_{\mathcal{C}_E}$. La portée (range) d'un rôle de composante $r_{\mathcal{C}_E}$ est un concept subsumée par E par rapport à $\mathcal{T}$.*

Exemple 16. *Dans la TBox $\mathcal{T}_{ex}$ de l'exemple 1 (p.5) à la page 5, on peut définir cinq composantes :*

— *$\mathcal{C}_{Matériau}$, ayant pour top concept $Matériau$ et pour rôle de composante $aMatériau$, qui traite des matériaux d'un instrument dans la TBox,*
— *$\mathcal{C}_{Type}$, ayant pour top concept $Type$ et pour rôle de composante $aType$, qui traite du type d'un instrument dans la TBox,*
— *$\mathcal{C}_{Unité}$, ayant pour top concept $Unité$ et pour rôle de composante $aUnité$, qui traite de l'unité de mesure d'un instrument dans la TBox,*
— *$\mathcal{C}_{ModeLecture}$, ayant pour top concept $ModeLecture$ et pour rôle de composante $aMode$, qui traite du mode de lecture d'un instrument dans la TBox,*
— *$\mathcal{C}_{Fabricant}$, ayant pour top concept $Fabricant$ pour rôle de composante $aFabricant$, qui traite de la Fabricant de fabrication d'un instrument dans la TBox.*

L'ensemble des noms de concept concernant la composante $\mathcal{C}_{Matériau}$ est : $\{Fer$, $Acier$, $Metal$, $Chêne$, $Bois$, $Plastique$, $Materiau\ \}$. Les concepts suivant (ensemble non exhaustif) concernent également la composante $\mathcal{C}_{Matériau}$: $\{Fer \sqcap Bois$, $Acier \sqcap Fer \sqcap Bois$, $...\}$.

Par commodité de langage et quand le contexte sera clair, on pourra employer le terme composante soit dans le sens exact de sa définition, soit pour évoquer son top concept, soit pour évoquer son rôle de composante associé.

Definition 12 (Réponse (resp. demande))**.** *Étant donnée une TBox $\mathcal{T}$ et $\mathcal{C}_{E_1}$, ..., $\mathcal{C}_{E_n}$ les n composantes de $\mathcal{T}$ (données arbitrairement), avec r_1, ..., r_n les rôles de composantes*

associés, une réponse O (resp. une demande D) est un concept qui s'écrit $O = \exists r_1.C_1 \sqcap \ldots \sqcap \exists r_n.C_n$ où chaque C_i est un concept concernant $\mathcal{C}_{E_i}$, $\forall i \in \{1, \ldots, n\}$.

On suppose que chacune des réponses (resp. demande) est exprimée en fonction de chaque composante. Si on ne veut pas exprimer une réponse par rapport à une composante $\mathcal{C}_{E_i}$, on a $C_i = \top$. Nous définissons maintenant la projection sur une composante, qui correspond au concept contenant uniquement les concepts de la composante.

Definition 13 (Projection sur une composante)**.** *Soient une TBox $\mathcal{T}$, $\mathcal{C}_{E_1}$, ..., $\mathcal{C}_{E_n}$ les n composantes de $\mathcal{T}$ (données arbitrairement), avec r_1, ..., r_n les rôles de composantes associés, et un concept $O = \exists r_1.C_1 \sqcap \ldots \sqcap \exists r_n.C_n$. La projection de O sur $\mathcal{C}_{E_i}$ est le concept C_i, $\forall i \in \{1, \ldots, n\}$. On la note O^{E_i}.*

Au sein d'une demande ou d'une réponse, selon la projection correspondante, les composantes sont considérées inexistantes ou existantes, selon la définition suivante.

Definition 14 (Composante existante et inexistante)**.** *Soient une TBox $\mathcal{T}$, $\mathcal{C}_E$ une composante et un concept O. On dit que $\mathcal{C}_E$ est existante dans O si $O^E \neq \top$, et inexistante sinon.*

Definition 15 (Concept et axiome concernant une composante)**.** *Étant donnés une TBox $\mathcal{T}$, $\mathcal{C}_E$ une composante, E le top concept de cette composante, un nom de concept B et un axiome Ax. On dit que B est un concept de la composante $\mathcal{C}_E$ s'il est subsumé par le top concept de $\mathcal{C}_E$. On a alors $B \in \mathcal{C}_E$ et $B \sqsubseteq_{\mathcal{T}} E$ par rapport à $\mathcal{T}$.*

On dit que Ax concerne la composante $\mathcal{C}_E$ si tous les noms de concept de Ax sont des concepts de la composante. C'est à dire que les n noms de concept D_i de l'axiome Ax sont des concepts de la composante $\mathcal{C}_E$. On a alors $D_i \in \mathcal{C}_E$ et $D_i \sqsubseteq_{\mathcal{T}} E$.

2.2 Component Comparison Operator

Dans cette section, on définit le principe de comparaison multicritère sémantique, par la notion de problème de comparaison de composantes. Nous précisons ensuite sur quelle instance de ce problème on travaille.

Definition 16 (Component Comparison Problem)**.** *Soient $\mathcal{T}$ une TBox dans $\mathcal{EL}$, $Dem \in \mathcal{T}_{\mathcal{EL}}$, $\{Rep_i, 1 \leq i \leq n\}$ un ensemble de n concepts de $\mathcal{T}_{\mathcal{EL}}$, $n \geq 1$, $\{\mathcal{C}_{E_k}, 1 \leq k \leq m\}$ un ensemble de m composantes dans $\mathcal{T}$. Soit $\{\phi_k, 1 \leq k \leq m\}$ un ensemble de m indices binaires de préférence partielle pour les composantes. Un problème de comparaison de composantes, identifié par $\{\mathcal{EL}, \mathcal{T}, \{Rep_i, 1 \leq i \leq n\}, Dem, \{\mathcal{C}_{E_k}, 1 \leq k \leq m\}, \{\phi_k, 1 \leq k \leq m\}\}$, consiste à ordonner tous les Rep_i selon leur*
$score_{Dem}(Rep_i) = \sum_{j=1, j\neq i}^{n} \sum_{k=1}^{m} \phi_k(Rep_i, Rep_j)$, avec

$$\phi_k(Rep_i, Rep_j) = \left\{ \begin{array}{lll} 1 & si & Rep_i \succsim_{E_k} Rep_j \\ 0 & si & Rep_i =_{E_k} Rep_j \\ -1 & si & Rep_i \precsim_{E_k} Rep_j \end{array} \right\}$$

Un problème de comparaison de composantes peut être vu comme un problème de matchmaking où, avec un ensemble de composantes et d'indices de préférence partielle, on a :

1. $\rho(Dem, Rep)$ *ssi* $Rep \in \{Rep_i, 1 \geq i \geq n\}$ *(chaque* Rep_i *peut être une solution)*
2. *et Rep minimal par rapport à* $\ll$ *tel que 1. ssi* $score(Rep_i)$ *est maximal.*

Pour définir une instance particulière du problème de comparaison de composantes, il faut définir $\succsim_{E_k}$. Le principe de comparaison, pour une composante, est le suivant :

1. On cherche les réponses de rest maximal (par rapport à $\sqsubseteq_{\mathcal{T}}$).
2. Si deux réponses ont des rest équivalents ou incomparables (par rapport à $\sqsubseteq_{\mathcal{T}}$), on préfère celle avec le miss maximal (par rapport à $\sqsubseteq_{\mathcal{T}}$).
3. Quand rest et miss sont équivalents ou incomparables, on cherche à minimiser la taille du rest, puis la taille du miss.

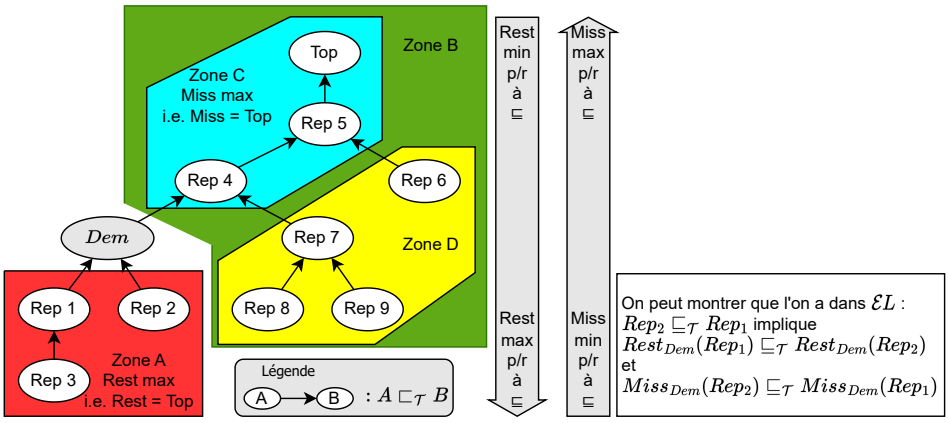


FIGURE 2.2 – Principe de la comparaison des réponses par rapport à une demande, pour une composante.

La figure 2.2 (p.39) illustre les différents cas possibles qui peuvent survenir entre une demande et une réponse, pour une composante. Dans les ellipses, on trouve la projection de la demande et des offres sur la composante étudiée. Les projections des réponses sur la composante sont classées dans différentes zones :

— Zone A, les réponses 1, 2 et 3 sont subsumées par la demande, elles sont plus précises que celle-ci. Leur rest est équivalent à $\top$. Elles correspondent à la classe "Correspondance totale" (voir section 1.2 (p.18)).

— Zone B, réponses 4 à 9 sont caractérisées par un rest $\sqsubset_{\mathcal{T}} \top$. Elles sont moins bonnes que les réponses de la zone A.
— Zone C, les réponses 4 et 5 subsument la demande. Elles correspondent à la classe "Correspondance connectée" (voir section 1.2 (p.18)).
— Zone D, les réponses 6, 7, 8 et 9 n'ont pas de relation de subsomption avec la demande. Elles correspondent à la classe "Correspondance potentielle" ou la classe "Correspondance disjointe" (voir section 1.2 (p.18)) .

Dans la figure 2.2 (p.39), si on suppose que :
— l'absence de flèche entre 2 offres signifie qu'elles sont incomparables par rapport à $\sqsubseteq_{\mathcal{T}}$,
— les Rest des offres 4 à 9 sont tous non équivalents 2 à 2,
— les Miss des offres 4 à 9 sont tous non equivalents 2 à 2 (sauf ceux des offres 4 et 5 qui valent $\top$)
— les offres qui sont incomparables 2 à 2 par rapport à $\sqsubseteq_{\mathcal{T}}$ ont des Rest et des Miss qui sont incomparables 2 à 2 par rapport à $\sqsubseteq_{\mathcal{T}}$,

on obtient les classements des réponses suivantes (des meilleures aux moins bonnes) :
Selon le rest uniquement : Rep 1, 2 et 3 > Rep 6, 8 et 9 > Rep 7 > Rep 4 > Rep 5.
Selon le miss uniquement : Rep 4 et 5 > Rep 1, 2, 6 et 7 > Rep 3, 8 et 9.
Selon le rest puis le miss : Rep 1 et 2 > Rep 3 > Rep 6, 8 et 9 > Rep 7 > Rep 4 > Rep 5.

On pourrait encore affiner le classement en comparant les offres selon la taille de leur Rest, puis selon la taille de leur Miss. Sans les hypothèses précédentes, pour classer les offres des zones B et C qui ne sont pas de Rest maximal, il faut calculer les Rest et Miss de chacune d'entre elles et les comparer par rapport à $\sqsubseteq_{\mathcal{T}}$.

Dans la comparaison des réponses 2 à 2 par rapport à une demande, comme dit précédemment, les réponses de la Zone A sont meilleures que les réponses de la zone B, parce que les rest des réponses de la zone A sont équivalents à $\top$. Si les deux réponses appartiennent toutes deux à la zone A, il faut calculer et comparer leur miss. Si les deux réponses appartiennent toutes deux à la zone B, il faut calculer et comparer leur rest, puis si nécessaire, leur miss. On peut donc maintenant définir l'ordre de préférence $\succsim_{E_k}$

Definition 17. *Soient $\mathcal{T}$ une TBox $\mathcal{EL}$, une demande $Dem \in \mathcal{T}_{\mathcal{EL}}$, deux réponses Rep_1 et $Rep_2 \in \mathcal{T}_{\mathcal{EL}}$, et une composante $\mathcal{C}_E$ de $\mathcal{T}$. On définit $\succsim_E$ tel que : $Rep_1 \succsim_E Rep_2$, qu'on lit Rep_1 est meilleure que Rep_2 pour la composante $\mathcal{C}_E$ par rapport à Dem et $\mathcal{T}$, si*

1. $rest^E_{Dem}(Rep_2) \sqsubseteq_{\mathcal{T}} rest^E_{Dem}(Rep_1)$,
2. $miss^E_{Dem}(Rep_2) \sqsubseteq_{\mathcal{T}} miss^E_{Dem}(Rep_1)$ *si les deux rest sont incomparables ou équivalents,*
3. $\mathsf{size}(rest^E_{Dem}(Rep_2)) \geq \mathsf{size}(rest^E_{Dem}(Rep_1))$, *si les deux rest sont équivalents ou incomparables par rapport à $\sqsubseteq_{\mathcal{T}}$ et si les deux rest et les deux miss sont équivalents ou incomparables par rapport à $\sqsubseteq_{\mathcal{T}}$,*

4. $\mathsf{size}(miss^E_{Dem}(Rep_2)) \geq \mathsf{size}(miss^E_{Dem}(Rep_1))$, *si les deux rest sont équivalents ou incomparables par rapport à* $\sqsubseteq_\mathcal{T}$, *et si les deux rest et les deux miss sont équivalents ou incomparables par rapport à* $\sqsubseteq_\mathcal{T}$ *et si les tailles des rest sont identiques.*

L'algorithme 1 (p.41)) est celui effectuant le tri de l'ensemble des réponses à une demande. Il prend en entrée la TBox utilisée par la base de connaissances, la liste des composantes de celle-ci et des rôles qui y sont associés, la demande de l'utilisateur et les réponses à comparer. Dans l'algorithme, la demande Dem et la TBox $\mathcal{T}$ par rapport auxquelles on calcule l'ordre de préférence ne sont pas précisées pour ne pas alourdir l'écriture. Elles seront cependant rappelées si le contexte n'est pas clair.

L'algorithme initialise un score pour chaque réponse à zéro, puis effectue la comparaison de chaque réponse deux à deux. C'est-à-dire que pour chaque réponse, il va comparer leurs différentes composantes, en utilisant l'algorithme 2 (section 2 (p.42)), ce qui va modifier le score de la réponse. Une fois l'ensemble des réponses comparées entre elles, l'algorithme effectue un classement des réponses de la meilleure à la plus mauvaise (du score le plus haut au score le plus bas). L'algorithme 2 suppose que l'on dispose d'un autre algorithme de calcul de subsomption par rapport à une TBox.

Algorithm 1 Tri des réponses.

Require: Une TBox $\mathcal{EL}$ $\mathcal{T}$ avec $\mathcal{C}_{E_1}$, ..., $\mathcal{C}_{E_n}$ les n composantes de $\mathcal{T}$ (données arbitrairement), un concept $\mathcal{EL}$ Dem (la demande), et m concepts $\mathcal{EL}$ $Rep_1, \ldots, Rep_m$ (les réponses).

Ensure: L'ensemble $e_{Rep} = \{(Rep_j, score_j) |\ j \in \{1, .., m\}\}$ des couples composés de Rep_j et du score associé $score_j$ par rapport à Dem et $\mathcal{T}$, classés du plus grand score au plus petit.

1: $e_{Rep} := \emptyset$
2: Initialisation de $score_1$ à $score_m$ à 0
3: **for all** $(Rep_i, Rep_j) \in \{Rep_1, \ldots, Rep_m\}^2$ avec $j > i$ **do**
4: **for all** composante $C_{E_k} \in \{C_{E_1}, \ldots, C_{E_n}\}$ **do**
5: $score_i := score_i + \phi_k(Rep_i, Rep_j)$ { //cf. algo. 2 (p.42) pour le calcul de ϕ_k}
6: $score_j := score_j - \phi_k(Rep_i, Rep_j)$
7: **end for**
8: **end for**
9: **for all** Rep_j **do**
10: $e_{Rep} := e_{Rep} \cup (Rep_j, score_j)$
11: **end for**
12: Tri des couples de e_{Rep} par score décroissant.
13: Renvoyer e_{Rep}

Proposition 1 (Propriétés du CCO et des algorithmes 1 (p.41) et 2 (p.42))**.** *Soit* $\mathcal{T}$ *une TBox* $\mathcal{EL}$ *définitionnelle et acyclique, Dem une demande utilisateur, l'ensemble des réponses possibles* Rep_i $(i > 0)$ *et un opérateur de différence entre concept par rapport à une TBox définitionnelle et acyclique, dont la complexité est dans EXPTIME, en fonction des tailles de* $\mathcal{T}, Dem$ *et des* Rep_i *et dans PTIME en fonction des tailles de* $\mathcal{T}^*(Dem)$ *et des* $\mathcal{T}^*(Rep_i)$*. On a :*

a. Les algorithmes 1 et 2 terminent et donnent un résultat unique (modulo l'ordonnancement des réponses de score égal).

b. Les algorithmes 1 et 2 donnent le résultat de la concordance relative par rapport à la proximité sémantique considérée.

c. Les algorithmes 1 et 2 sont dans EXPTIME, en fonction des tailles de $\mathcal{T}, Dem$ et des Rep_i. Les algorithmes sont dans PTIME en fonction des tailles de $\mathcal{T}^*(Dem)$ et des $\mathcal{T}^*(Rep_i)$.

Preuves. a. Les deux algorithmes étant itératifs, ils se terminent à condition que l'algorithme utilisé pour calculer la différence entre deux concepts (lors du calcul du rest et du miss) termine également. L'unicité du résultat est garantie par l'unicité du résultat de l'algorithme de différence. La terminaison et l'unicité du résultat de l'algorithme de différence sont abordés dans la section 3.1.1 (p.46)

b. L'algorithme 2 (p.42) est la traduction algorithmique de la fonction de comparaison ϕ par rapport à $\succsim_{E_k}$ (dont le principe est donnée au début de la section 2.2). L'algorithme

Algorithm 2 Fonction $\phi_{R,\mathcal{T}}(Dem)(Rep_1, Rep_2)$ d'ordonnancement sémantique de deux réponses Rep_1 et Rep_2 pour une demande Dem par rapport à une composante $\mathcal{C}_{E_i}$ de $\mathcal{T}$.

Require: Une TBox $\mathcal{EL}$ $\mathcal{T}$, une composante $\mathcal{C}_{E_i}$ de $\mathcal{T}$, la description Dem d'une demande, et les concepts Rep_1 et Rep_2, deux réponses.

Ensure: 1 si Rep_1 est meilleure que Rep_2 pour Dem dans $\mathcal{T}$ par rapport à $\mathcal{C}_{E_i}$, -1 si Rep_2 est meilleure, et 0 si Rep_1 et Rep_2 sont équivalentes.

if $Rep_1^{E_i} \equiv_{\mathcal{T}} Dem^{E_i}$ **then**
  if $Rep_2^{E_i} \equiv_{\mathcal{T}} Dem^{E_i}$ **then**
    Renvoyer 0
  else
    Renvoyer 1
  end if
else if $Rep_1^{E_i} \sqsubset_{\mathcal{T}} Dem^{E_i}$ **then**
  if $Rep_2^{E_i} \equiv Dem^{E_i}$ **then**
    Renvoyer -1
  else if $Rep_2^{E_i} \sqsubset_{\mathcal{T}} Dem^{E_i}$ **then**
    if $miss_{Dem^{E_i}}(Rep_1^{E_i}) \sqsubset_{\mathcal{T}} miss_{Dem^{E_i}}(Rep_2^{E_i})$ **then**
      Renvoyer -1
    else if $miss_{Dem^{E_i}}(Rep_1^{E_i}) \sqsupset_{\mathcal{T}} miss_{Dem^{E_i}}(Rep_2^{E_i})$ **then**
      Renvoyer 1
    else
      if $\mathsf{size}(miss_{Dem^{E_i}}(Rep_1^{E_i})) > \mathsf{size}(miss_{Dem^{E_i}}(Rep_2^{E_i}))$ **then**
        Renvoyer -1
      else if $\mathsf{size}(miss_{Dem^{E_i}}(Rep_1^{E_i})) < \mathsf{size}(miss_{Dem^{E_i}}(Rep_2^{E_i}))$ **then**
        Renvoyer 1
      else
        Renvoyer 0
      end if
    end if
  else
    Renvoyer 1
  end if

```
27: else
28:   if Rep_2^{E_i} ⊑_T Dem^{E_i} then
29:     Renvoyer -1
30:   else
31:     if rest_{Dem^{E_i}}(Rep_1^{E_i}) ⊏_T rest_{Dem^{E_i}}(Rep_2^{E_i}) then
32:       Renvoyer -1
33:     else if rest_{Dem^{E_i}}(Rep_1^{E_i}) ⊐_T rest_{Dem^{E_i}}(Rep_2^{E_i}) then
34:       Renvoyer 1
35:     else
36:       if miss_{Dem^{E_i}}(Rep_1^{E_i}) ⊏_T miss_{Dem^{E_i}}(Rep_2^{E_i}) then
37:         Renvoyer -1
38:       else if miss_{Dem^{E_i}}(Rep_1^{E_i}) ⊐_T miss_{Dem^{E_i}}(Rep_2^{E_i}) then
39:         Renvoyer 1
40:       else
41:         if size(rest_{Dem^{E_i}}(Rep_1^{E_i})) > size(rest_{Dem^{E_i}}(Rep_2^{E_i})) then
42:           Renvoyer -1
43:         else if size(rest_{Dem^{E_i}}(Rep_1^{E_i})) < size(rest_{Dem^{E_i}}(Rep_2^{E_i})) then
44:           Renvoyer 1
45:         else
46:           if size(miss_{Dem^{E_i}}(Rep_1^{E_i})) > size(miss_{Dem^{E_i}}(Rep_2^{E_i})) then
47:             Renvoyer -1
48:           else if size(miss_{Dem^{E_i}}(Rep_1^{E_i})) < size(miss_{Dem^{E_i}}(Rep_2^{E_i})) then
49:             Renvoyer 1
50:           else
51:             Renvoyer 0
52:           end if
53:         end if
54:       end if
55:     end if
56:   end if
57: end if
```

1 (p.41) est la traduction algorithmique de la concordance relative (voir 1.4).

c. L'algorithme 2 (p.42) effectue, dans le pire des cas, deux calculs de différence et deux calculs de subsomption (entre les rest et entre les miss). L'algorithme 1 (p.41) effectue une comparaison entre chaque paire des m réponses possibles, pour chacune des n composantes. Les deux algorithmes effectuent donc, au pire, $n(m^2 - m)$ calculs de différence et de subsomption. Le calcul de subsomption étant polynomial, la complexité du CCO est bornée par celle de l'opérateur de différence. Avec un opérateur de différence qui est dans EXPTIME en fonction des tailles de $\mathcal{T}, Dem$ et des Rep_i et dans PTIME en fonction des tailles de $\mathcal{T}^*(Dem)$ et des $\mathcal{T}^*(Rep_i)$ de complexité exponentielle par rapport à la taille de la TBox, de la demande et des réponses alors, le CCO est dans EXPTIME en fonction des tailles de $\mathcal{T}, Dem$ et des Rep_i et dans PTIME en fonction des tailles de $\mathcal{T}^*(Dem)$ et des $\mathcal{T}^*(Rep_i)$.

□

Calculer le rest et le miss nécessite d'utiliser un opérateur de différence dans la logique

de description utilisée, ce qui en fait le sujet du chapitre suivant.

Chapitre 3

Component Subtraction Operator : opérateur de différence pour $\mathcal{EL}$

Pour calculer le rest et le miss, nous avons besoin d'un opérateur de différence. Les opérateurs existants, listés dans la partie 1.3 (p.23), ne correspondent pas aux deux critères que l'on souhaite assurer pour le CCO, qui sont illustrés dans l'exemple 20 (p.52) et définis dans l'introduction du chapitre 1.3 :

— critère de subsomption inverse, c'est-à-dire que l'opérateur retire du diminuende une partie de concept si celle-ci subsume le diminuteur.

— critère de fine granularité, c'est-à-dire que l'opérateur est en mesure de retirer des parties de concepts à l'intérieur de restrictions existentielles.

Dans ce chapitre, nous proposons un opérateur de différence qui respecte les critères précédents pour des TBox acycliques et définitionnelles, dans la section 3.1 : le Component Subtraction Operator. Dans la section 3.2, nous proposons une heuristique capable de fonctionner avec des TBox acycliques générales. Dans les deux sections, nous donnons les propriétés des algorithmes correspondants aux opérateurs.

3.1 Cas d'une TBox acyclique et définitionnelle

Dans la section 3.1.1 (p.46), nous définissons le CSO, d'abord informellement, en présentant la notion de similarité descriptionnelle, qui va s'appuyer sur celle des branches caractéristiques, puis formellement. Dans la section suivante 3.1.2 (p.49), nous proposons un nouvel opérateur syntaxique appelé "tree subtraction operator" (TSO) et montrons comment l'utiliser pour calculer le CSO dans la section 3.1.3 (p.51). Nous donnons également les propriétés principales du CSO et du TSO : existence, unicité, terminaison, justesse et complexité des algorithmes associés. Nous utiliserons l'exemple 17 dans cette section :

Exemple 17. *Soit la TBox définitionnelle et acyclique :* $\mathcal{T} = \{A \equiv B \sqcap C, C \equiv D \sqcap \exists r.B, D \equiv E, F \equiv \exists r.D\}$. *On a alors* $\mathbf{C}_\mathcal{T} = \{A,\ B,\ C,\ D,\ E,\ F\}$, $\mathbf{r}_\mathcal{T} = \{r\}$, $\mathsf{prim}_\mathcal{T} = \{B, E\}$ *et* $\mathsf{def}_\mathcal{T} = \{A, C, D, F\}$.
Son expansion complète (définie précédemment à la section 1.1 (p.17)) $\mathcal{T}^*$ est
$\mathcal{T}^* = \{A \equiv B \sqcap E \sqcap \exists r.B,\ C \equiv E \sqcap \exists r.B,\ D \equiv E,\ F \equiv \exists r.E\}$.
Soient la demande Dem et la réponse Rep suivantes :
$Rep = A \sqcap F \sqcap \exists r.\top \sqcap \exists s.\top$
$Dem = B \sqcap \exists r.D \sqcap \exists s.E$

Les résultats attendus, respectant le critère de subsomption inverse et de fine granularité, sont :
$Rep \ominus_\mathcal{T} Dem = E \sqcap \exists r.B$
$Dem \ominus_\mathcal{T} Rep = \exists s.E$

3.1.1 Définition du CSO

L'opérateur CSO $C \ominus_\mathcal{T} D$ cherche à retirer du diminuende C toutes les parties de concept partagées avec le diminuteur D par rapport à une TBox $\mathcal{T}$. Nous appelons ces parties partagées des similarités descriptionnelles de C vers D.

Syntaxiquement, nous voulons pouvoir retirer des similarités descriptionnelles du diminuende sans affecter ses autres parties. Il faut donc opérer de manière atomique. Comme les concepts $\mathcal{EL}$ ont une structure arborescente, on peut définir la notion de partie atomique d'un concept comme les branches de sa structure arborescente, comme dans la figure 3.1. La définition d'une structure arborescente est rappelée dans l'annexe 35 (p.174).

Sémantiquement, une similarité descriptionnelle de C vers D par rapport à $\mathcal{T}$ signifie qu'une partie de C est reliée à D (dans la structure arborescente). Nous proposons qu'une similarité descriptionnelle de C vers D par rapport à $\mathcal{T}$ soit définie en tant que branche caractéristique qui subsume D par rapport à $\mathcal{T}$:

- Une branche caractéristique de C par rapport à $\mathcal{T}$ est une branche primitive de C par rapport à $\mathcal{T}$ qui ne peut pas être syntaxiquement ôtée de C sans changer sa sémantique. Une branche primitive possède un concept primitif ou $\top$ comme feuille. En se focalisant sur les branches primitives (plutôt que les branches du concept initial), on impose à l'opérateur de différence de s'appliquer aux expansions complètes de ses opérandes. On prend ainsi en compte la TBox dans la différence. Travailler au niveau des branches permet d'implémenter le critère de fine granularité.
- Imposer que D soit subsumé par toute similarité descriptionnelle de C vers D nous permet d'implémenter le critère de subsomption inverse.

On propose ensuite de définir $C \ominus_\mathcal{T} D$ comme le concept minimal E qui subsume C tel qu'il n'y ait aucune similarité descriptionnelle de E vers D par rapport à $\mathcal{T}$. C'est-à-dire que toutes les similarités descriptionnelles de C vers D par rapport à $\mathcal{T}$ ont été retirées.

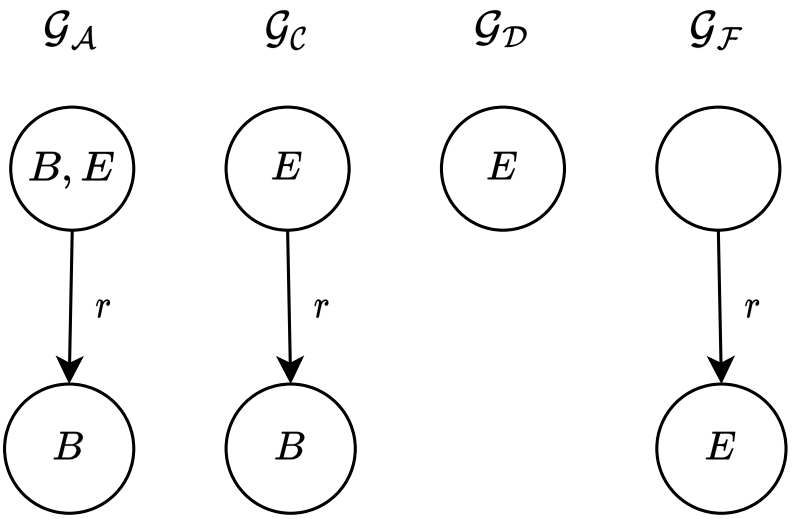


FIGURE 3.1 – Représentations arborescentes $\mathcal{G}$ des concepts définis A, C, D et F de l'exemple 17. Les concepts dans un nœud sont en conjonction. Les arrêtes indiquent une conjonction avec une restriction existentielle. A possède trois branches : B, E et $\exists r.B$ et on peut retrouver la définition de A a partir de cette représentation : $A \equiv B \sqcap E \sqcap \exists r.B.$, C possède deux branches : E et $\exists r.B$, D possède une branche : E et F possède une branche : $\exists r.E$.

On va maintenant formaliser toutes ces notions.

Une sous-description d'un concept C est obtenue en retirant n'importe où dans C, zéro ou plusieurs conjoncts, tant que C reste syntaxiquement correct. La définition suivante formalise cette idée, elle est équivalente à la définition donnée par Brandt et al. (2002) (restreinte à $\mathcal{EL}$).

Definition 18 (Sous-description d'un concept)**.** *Avec C un concept et $n \geq 2$, l'ensemble des sous-descriptions de C, noté subd_C, prend pour valeur :*

- $\{C\}$ *si* $C \in \mathbf{C} \cup \{\top\}$
- $\{\exists r.E \mid E \in \mathsf{subd}_D\}$ *si* $C = \exists r.D$
- $\bigcup_{\substack{\mathcal{C} \subseteq \{C_i | 1 \leq i \leq n\} \\ avec\ |\mathcal{C}| = m \neq 0}} \left(\bigcup_{\langle S_1, \ldots, S_m \rangle \in \prod_{C_i \in \mathcal{C}} \mathsf{subd}_{C_i}} \left(\prod_{j=1}^{m} S_j \right) \right)$ *si* $C = \prod_{i=1}^{n} C_i$.

On note $\mathsf{subd}^{\mathsf{prim}}_{C,\mathcal{T}} = \mathsf{subd}_C \cap \mathcal{T}^{\mathsf{prim}}_{\mathcal{EL}}$ l'ensemble des sous-descriptions de C où les noms de concept appartiennent à $\mathcal{T}^{\mathsf{prim}}_{\mathcal{EL}}$. L'ensemble $\mathcal{T}^{\mathsf{prim}}_{\mathcal{EL}}$ est l'ensemble des concept $\mathcal{EL}$ qui peuvent être construit à partir des concepts primitifs de $\mathcal{T}$ et de $\top$.

La notion de sous-description est proche, mais différente de celle de la notion de sous-concept défini par Baader et al. (2017). Informellement, un sous-concept est n'importe quelle conjonction prise dans le concept original dans laquelle zéro ou plusieurs conjoncts ont été retirés (en en gardant au moins un).

La notion de similarité descriptionnelle est basée sur le raffinement de la notion de sous-description : la notion de branche. Une branche est un concept ayant une largeur d'au plus 1.

Definition 19 (Branche)**.** *Soit une TBox $\mathcal{T}$ et $C \in \mathcal{T}_{\mathcal{EL}}$. L'ensemble des branches de la TBox $\mathcal{T}$ est noté $\mathsf{br}_{\mathcal{T}}$. L'ensemble des branches primitives de la TBox $\mathcal{T}$ est noté $\mathsf{br}^{\mathsf{prim}}_{\mathcal{T}}$. L'ensemble des branches de C est noté br_C. Et l'ensemble des branches de C qui sont primitives dans la TBox $\mathcal{T}$ est noté $\mathsf{br}^{\mathsf{prim}}_{C,\mathcal{T}}$. Ces ensembles sont définis de la manière suivante :*
$\mathsf{br}_{\mathcal{T}} = \{S \in \mathcal{T}_{\mathcal{EL}} \mid \mathsf{wid}(S) = 1\}$
$\mathsf{br}^{\mathsf{prim}}_{\mathcal{T}} = \{S \in \mathcal{T}^{\mathsf{prim}}_{\mathcal{EL}} \mid \mathsf{wid}(S) = 1\}$
$\mathsf{br}_C = \{S \in \mathsf{subd}_C \mid \mathsf{wid}(S) = 1\}$
$\mathsf{br}^{\mathsf{prim}}_{C,\mathcal{T}} = \{S \in \mathsf{subd}^{\mathsf{prim}}_{C,\mathcal{T}} \mid \mathsf{wid}(S) = 1\}$

Exemple 18. *En utilisant la TBox $\mathcal{T}$ de l'exemple 17 (p.45), soit G le concept suivant : $G = C \sqcap \exists r_1.(\exists r_2.\exists r_3.\top \sqcap D \sqcap \exists r_4.B)$. On a alors :*
$\mathsf{br}_G = \{C, \exists r_1.\exists r_2.\exists r_3.\top, \exists r_1.D, \exists r_1.\exists r_4.B\}$ *et* $\mathsf{br}^{\mathsf{prim}}_{G,\mathcal{T}} = \{\exists r_1.\exists r_2.\exists r_3.\top, \exists r_1.\exists r_4.B\}$.

Une branche caractéristique de C par rapport à $\mathcal{T}$ est une branche primitive de C par rapport à $\mathcal{T}$ qui ne peut être syntaxiquement ôtée de C sans changer sa sémantique.

Definition 20 (Branche caractéristique)**.** *Soit $\mathcal{T}$ une TBox et un concept $C \in \mathcal{T}_{\mathcal{EL}}$. L'ensemble $\mathsf{char}^{\mathcal{T}}_C$ des branches caractéristiques de C par rapport à $\mathcal{T}$ est défini ainsi :*
$\mathsf{char}^{\mathcal{T}}_C = \{S \in \mathsf{br}^{\mathsf{prim}}_{\mathcal{T}} \mid \exists C' \in \mathcal{T}_{\mathcal{EL}}$ *tel que*
$\big(C \equiv_{\mathcal{T}} C'$ *et* $S \in \mathsf{br}_{C'}$ *et* $\forall C'' \in \mathcal{T}_{\mathcal{EL}}\ (\mathsf{br}_{C''} = \mathsf{br}_{C'} \setminus \{S\}) \Rightarrow (C'' \not\equiv_{\mathcal{T}} C))\}$

Les $\mathcal{T}$-similarités de C vers D par rapport à $\mathcal{T}$ sont définies comme des branches caractéristiques de C qui subsument D. Notons que $\mathcal{T}_{\mathcal{EL}}$ peut être infini. Cependant, C' doit rester équivalent à C par rapport à $\mathcal{T}$, et si nous n'envisageons les descriptions avec des conjoncts redondants, il n'existe pas une infinité de C' dans $\mathcal{T}_{\mathcal{EL}}$ tel que $C' \equiv C$.

Definition 21 (Similarité descriptionnelle)**.** *Soit $\mathcal{T}$ une TBox et une paire de concepts $(C, D) \in (\mathcal{T}_{\mathcal{EL}})^2$. L'ensemble $\mathsf{dcom}^{\mathcal{T}}_{C,D}$ des $\mathcal{T}$-similarités descriptionnelles de C vers D est défini ainsi :*
$\mathsf{dcom}^{\mathcal{T}}_{C,D} = \{S \in \mathsf{char}^{\mathcal{T}}_C \mid D \sqsubseteq_{\mathcal{T}} S\}$

Enfin, $C \ominus_{\mathcal{T}} D$ est défini comme le concept minimal E (w.r.t. $\sqsubseteq_{\mathcal{T}}$) qui subsume C et n'ayant aucune similarité de E vers D (l'unicité est montrée dans la proposition 3 (p.51)). Quand toutes les branches caractéristiques sont des similarités (et doivent donc être toutes retirées de C), le résultat est $\top$.

Definition 22 (CSO)**.** *Soient $\mathcal{T}$ une TBox et une paire de concepts $(C, D) \in (\mathcal{T}_{\mathcal{EL}})^2$. L'opérateur binaire $\ominus_{\mathcal{T}}$, nommé "Commonality Subtraction Operator" (CSO) pour $\mathcal{T}$, est défini ainsi :*

$$C \ominus_{\mathcal{T}} D = \begin{cases} Min_{\sqsubseteq_{\mathcal{T}}}\{E \in \mathcal{T}_{\mathcal{EL}} \mid C \sqsubseteq_{\mathcal{T}} E \text{ et } \mathsf{dcom}^{\mathcal{T}}_{E,D} = \emptyset\} & \text{si } \mathsf{char}^{\mathcal{T}}_C \not\subseteq \mathsf{dcom}^{\mathcal{T}}_{C,D} \\ \top & \text{si } \mathsf{char}^{\mathcal{T}}_C = \mathsf{dcom}^{\mathcal{T}}_{C,D} \end{cases}$$

L'exemple 19 donne la résolution de l'exemple 17 (p.45) par le CSO de la définition 22.

Exemple 19. *Soient la TBox $\mathcal{T}$ de l'exemple 17 (p.45), la réponse Rep et la demande Dem suivantes :*

$\mathcal{T} = \{A \equiv B \sqcap C, C \equiv D \sqcap \exists r.B, D \equiv E, F \equiv \exists r.D\}$

$Rep = A \sqcap F \sqcap \exists r.\top \sqcap \exists s.\top$

$Dem = B \sqcap \exists r.D \sqcap \exists s.E.$

On a les expansions complètes suivantes :

$\mathcal{T}^*(Rep) = B \sqcap E \sqcap \exists r.B \sqcap \exists r.E \sqcap \exists r.\top \sqcap \exists s.\top$

$\mathcal{T}^*(Dem) = B \sqcap \exists r.E \sqcap \exists s.E$

On a les branches caractéristiques suivantes :

$\mathsf{char}^{\mathcal{T}}_{Rep} = \{B, E, \exists r.B, \exists r.E, \exists s.\top\}$

$\mathsf{char}^{\mathcal{T}}_{Dem} = \{B, \exists r.E, \exists s.E\}$

On a les similarités descriptionnelles suivantes :

$\mathsf{dcom}^{\mathcal{T}}_{Rep,Dem} = \{B, \exists r.E, \exists s.\top\}$

$\mathsf{dcom}^{\mathcal{T}}_{Dem,Rep} = \{B, \exists r.E\}$

On a alors comme résultats :

$Rep \ominus_{\mathcal{T}} Dem = E \sqcap \exists r.B$

$Dem \ominus_{\mathcal{T}} Rep = \exists s.E$

3.1.2 Tree Subtraction Operator (TSO)

Pour pouvoir calculer $C \ominus_{\mathcal{T}} D$, nous proposons une approche basée sur un opérateur de différence syntaxique qui s'applique aux branches des expansions complètes de C et D. Cet opérateur n'est pas une différence ensembliste classique des ensembles de branches, car il prend en compte les relations de subsomption entre les branches (et en particulier celles incluant le concept $\top$.). De plus, il ne modifie pas la structure arborescente originale du diminuende. Nous appelons cet opérateur le Tree Subtraction Operator (TSO), noté $\triangle$. Informellement, $C \triangle D$, qui est lu "C privé de D", est le concept minimal par rapport à la subsomption qui subsume C et tel que toutes les branches de br_C qui subsument une branche de br_D ont été retirées de br_C. Cela signifie que l'on retire les branches de br_C qui sont aussi des branches de br_D, mais également les branches de br_C qui se terminent par $\top$ quand il existe une branche dans br_D qui commence par les mêmes restrictions existentielles. Si toutes les branches de C sont retirées, alors le résultat est $\top$.

Definition 23 (TSO)**.** *Soient C et D deux concepts. L'opérateur binaire $\triangle$, appelé tree subtraction operator (TSO), est défini ainsi :*

$$C \triangle D = \begin{cases} Min_{\sqsubseteq}\{E \mid C \sqsubseteq E \text{ et } \mathsf{br}_E = \mathsf{br}\} & \text{si } \mathsf{br} \neq \emptyset \\ \top & \text{si } \mathsf{br} = \emptyset \end{cases}$$

avec $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

et P un nom de concept quelconque ou $\top$ (P ne peut pas etre $\top$ quand $m = 0$).

Il n'est pas difficile de voir que la définition 23 (p.49) assure que $C \triangle D$ garde la structure arborescente de C, c'est-à-dire que $C \triangle D$ est une sous-description de C. Cela vient du fait que $C \triangle D$ est minimal par rapport à $\sqsubseteq$, que $C \sqsubseteq C \triangle D$ et que l'ensemble de branches du résultat est un sous-ensemble de l'ensemble des branches de C. On peut illustrer cela avec des concepts non équivalents mais possédant le même ensemble de branches : $C_1 = \exists r.A \sqcap \exists r.B$ et $C_2 = \exists r.(A \sqcap B)$. Leurs branches sont $\mathsf{br}_{C_1} = \mathsf{br}_{C_2} = \{\exists r.A, \exists r.B\}$. Si l'on retire de C_1 un concept D qui n'a pas de similarité descriptionnelle avec C_1, le résultat est C_1. Si l'on retire de C_2 un concept D qui n'a pas de similarité descriptionnelle avec C_2, le résultat est C_2. On a alors $C_1 \triangle D \neq C_2 \triangle D$ car la structure arborescente initiale de C_1 et de C_2 est conservée par le TSO, même si leurs branches sont identiques.

Algorithm 3 tso(C, D).

Require: C et D deux concepts $\mathcal{EL}$.
Ensure: $C \triangle D$ (voir définition 23 (p.49))

if $C = D$ ou $C = \top$ **then**
  $Result := \top$
else
  if $C = C_1 \sqcap \ldots \sqcap C_n$ avec $n \geq 2$ **then**
    $Result1 := \text{tso}(C_1, D) \sqcap \ldots \sqcap \text{tso}(C_n, D)$
    if il y a au moins un conjonct $\neq \top$ dans $Result1$ **then**
      $Result :=$ $Result1$ sans aucun conjonct $\top$.
    else
      $Result := \top$
    end if
  else if $D = D_1 \sqcap \ldots \sqcap D_m$ avec $m \geq 2$ **then**
    $Result := \text{tso}(\ldots(\text{tso}(\text{tso}(C, D_1), D_2), \ldots), D_m)$
  else if $C = \exists r.C'$ et $D = \exists r.D'$ **then**
    $Result1 := \text{tso}(C', D')$
    if $Result1 = \top$ **then**
      $Result := \top$
    else
      $Result := \exists r.Result1$
    end if
  else
    $Result := C$
  end if
end if
Retourner $Result$

TSO est implémenté par l'algorithme 3 (p.50). Il parcourt en même temps la structure arborescente de C et de D, enlevant de C les branches qui subsument, par rapport à une TBox vide, une branche de D. Les propriétés du TSO et de son algorithme 3 (p.50)

sont données dans la proposition 2 (p.51). Les lignes 5 et 12 peuvent être écrite à l'aide d'une récurrence sur n et m, mais nous conservons l'écriture avec des boucles implicites pour faciliter la lecture.

Proposition 2. *Soient C et D deux concepts. On a :*
a. $C \vartriangle D$ est unique et existe toujours.
b. L'algorithme 3 (p.50) se termine et produit un résultat unique.
c. $C \vartriangle D = tso(C, D)$ (justesse de l'algorithme 3 (p.50)).
d. Calculer $tso(C, D)$ est PTIME par rapport aux tailles de C et D.

Ebauche de preuves. a. L'existence vient de la définition de br qui correspond toujours à au moins un concept, ou $\top$ quand br est vide. L'unicité provient de la minimalité par rapport à la subsomption.

b. On montre par récurrence sur les tailles de C et D que $\text{tso}(C, D)$ se termine toujours et génère un résultat qui est toujours strictement plus petit que $\mathsf{size}(C) + \mathsf{size}(D)$.

c. La justesse est montrée en 3 étapes : (i) de la caractérisation de la subsomption dans $\mathcal{EL}$ en l'absence de TBox de Baader et al. (2017), on dérive une caractérisation de la subsomption dans $\mathcal{EL}$ en termes de sous-descriptions, (ii) on dérive ensuite une caractérisation du TSO en termes de sous-descriptions, (iii) une preuve par récurrence de la justesse est obtenue à partir de la caractérisation obtenue à l'étape (ii).

d. L'efficacité est démontrée en exhibant un pire cas et en étudiant la complexité dans ce cas précis.

□

La preuve complète se trouve à l'annexe C.1 (p.171).

3.1.3 Calculer le CSO à l'aide du TSO

On peut maintenant utiliser le TSO pour calculer le CSO : $C \ominus_{\mathcal{T}} D$ est obtenu en calculant $\mathcal{T}^*(C) \vartriangle \mathcal{T}^*(D)$ (voir algorithme 4 (p.51)). La proposition 3 (p.51) montre les propriétés associées au CSO et la complexité de l'algorithme 4 (p.51).

Algorithm 4 $\text{cso}(\mathcal{T}, C, D)$.

Require: • $\mathcal{T}$ est une TBox $\mathcal{EL}$ acyclique et définitionnelle.
• $(C, D) \in (\mathcal{T}_{\mathcal{EL}})^2$
Ensure: $C \ominus_{\mathcal{T}} D$ (voir la définition 22 (p.48))
1: Retourner $\text{tso}(\mathcal{T}^*(C), \mathcal{T}^*(D))$

Proposition 3. *Soient une TBox $\mathcal{T}$ et une paire de concepts $(C, D) \in (\mathcal{T}_{\mathcal{EL}})^2$. On a :*
a. $C \ominus_{\mathcal{T}} D$ existe et est unique (modulo $\equiv_{\mathcal{T}}$).
b. $cso(\mathcal{T}, C, D)$ se termine.
c. $C \ominus_{\mathcal{T}} D = \mathcal{T}^*(C) \vartriangle \mathcal{T}^*(D) = cso(\mathcal{T}, C, D)$ (Justesse de l'algorithme 4 (p.51)).
d. Calculer le $cso(\mathcal{T}, C, D)$ est EXPTIME par rapport aux tailles de $\mathcal{T}$ (car la complexité de l'expansion complète est EXPTIME (Voir Nebel (1990))), C et D et PTIME par rapport aux tailles de $\mathcal{T}^*(C)$ et $\mathcal{T}^*(D)$.

Ebauche de preuve. a. L'existence et l'unicité du CSO découlent de la définition 22 (p.48).

b. La terminaison est justifiée car l'expansion complète et l'algorithme 3 (p.50) se terminent également.

c. La justesse de $\text{cso}(\mathcal{T}, C, D)$ provient de : (i) la caractérisation de la subsomption en termes de sous description, déjà utilisée dans les preuves de la proposition 2 (p.51), et de (ii) le lemme disant que $\mathsf{br}_{C \triangle D}$ est l'ensemble des branches de C qui ne subsume pas D.

d. Le résultat est facilement obtenu de la complexité de l'expansion complète (qui est EXPTIME) et de l'algorithme 3 (p.50). □

La preuve complète se trouve à l'annexe C.2 (p.186).

3.1.4 Cas d'une TBox avec définitions de concepts primitifs

À la section 1.1 (p.13), on a rappelé que l'on pouvait facilement transformer une TBox contenant des définitions de concepts et des définitions de concepts primitifs en une TBox définitionnelle (ne contenant pas de définition de concept primitif). Si $\mathcal{T}$ est une telle TBox, le résultat du calcul de $C \ominus_{\mathcal{T}} D$ devra être "nettoyé" pour obtenir le CSO : il faudra remplacer les concepts $\overline{C}$ par les concepts d'origine C et, dans chaque conjonction, ne garder que les plus petits conjoncts par rapport à $\sqsubseteq_{\mathcal{T}}$. L'exemple 20 (p.52) illustre cette situation :

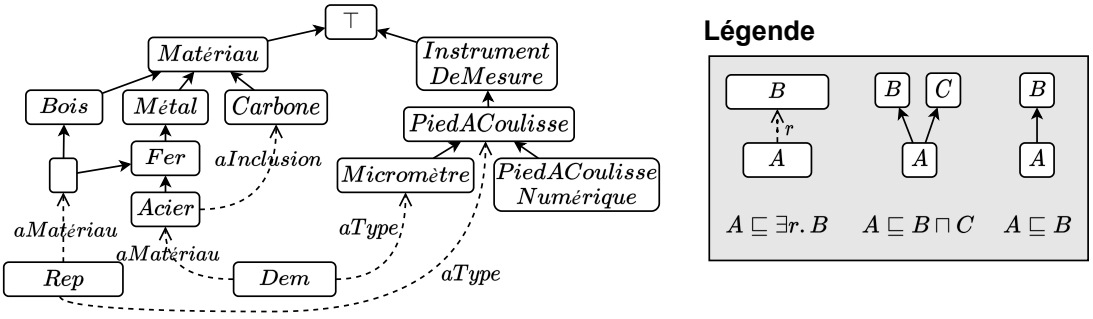


FIGURE 3.2 – TBox de métrologie acyclique

Exemple 20. *Dans le contexte de la métrologie, on a la TBox $\mathcal{EL}$, les concepts réponses et requête suivants (voir figure 3.2 (p.52))) :*
$\mathcal{T} = \{Acier \equiv fer \sqcap \exists aInclusion.Carbone,\ Fer \sqsubseteq Métal,\ Métal \sqsubseteq Matériau,\ Bois \sqsubseteq Matériau,\ Carbone \sqsubseteq Matériau,\ Micromètre \sqsubseteq PaC,\ PaCNumérique \sqsubseteq PaC,\ PaC \sqsubseteq InstrumentDeMesure\}$.
$Rep = \exists aType.PaC \sqcap \exists aMatériau.(Fer \sqcap Bois)$
$Dem = \exists aType.Micromètre \sqcap \exists aMatériau.Acier.$

Acier est défini comme du Fer dans lequel est inclus du Carbone, Fer est un Métal qui est un Matériau, comme Bois et Carbone. Micromètre est un PaC, comme

$PaCNumérique$, et PaC est un $InstrumentDeMesure$. Rep décrit une réponse à propos d'un PaC comme type d'instrument, fait de Fer et de $Bois$. Avec Dem, un utilisateur recherche un $Micromètre$ fait en $Acier$. On aimerait avoir : (i) $Rep \ominus_{\mathcal{T}} Dem = \exists aMatériau.Bois$, qui signifie que Dem partage toutes les informations de Rep à part le fait que Rep décrit un instrument fait de bois, et (ii) $Dem \ominus_{\mathcal{T}} Rep = \exists aType.Micromètre \sqcap \exists aMatériau.\exists aInclusion.Carbone$, qui signifie que Rep partage avec Dem que l'instrument décrit est fait en Fer, mais pas les autres aspects de Dem (que l'instrument est un $Micromètre$ et que le matériau à des inclusions de $Carbone$).

On explicite maintenant les différentes étapes dans les calculs $Rep \ominus_{\mathcal{T}} Dem$ et $Dem \ominus_{\mathcal{T}} Rep$, et notamment en montrant l'impact de la transformation de $\mathcal{T}$ en son expansion complète (puisque $\mathcal{T}$ contient des définitions de concepts primitifs).
La version définitionnelle $\overline{\mathcal{T}}$ de $\mathcal{T}$ est :
$\overline{\mathcal{T}} = \{Acier \equiv Fer \sqcap \exists contient.Carbone,$
$Fer \equiv Métal \sqcap \overline{Fer},$
$Métal \equiv Matériau \sqcap \overline{Métal},$
$Bois \equiv Matériau \sqcap \overline{Bois},$
$Carbone \equiv Matériau \sqcap \overline{Carbone},$
$Micromètre \equiv PiedACoulisse \sqcap \overline{Micromètre},$
$PiedACoulisseNumérique \equiv PiedACoulisse \sqcap \overline{PiedACoulisseNumérique},$
$PiedACoulisse \equiv InstrumentDeMesure \sqcap \overline{PiedACoulisse}\}$
Les expansions complètes de Rep et Dem sont :
$\overline{\mathcal{T}}^*(Rep) \equiv$
$\exists aType.(InstrumentDeMesure \sqcap \overline{PiedACoulisse}) \sqcap$
$\exists faitDe.(Matériau \sqcap \overline{Métal} \sqcap \overline{Fer} \sqcap \overline{Bois})$
$\overline{\mathcal{T}}^*(Dem) \equiv \exists aType.(InstrumentDeMesure \sqcap \overline{PiedACoulisse} \sqcap \overline{Micromètre}) \sqcap$
$\exists faitDe.(Matériau \sqcap \overline{Métal} \sqcap \overline{Fer} \sqcap \exists contient.(Matériau \sqcap \overline{Carbone}))$
Les résultats du TSO sont :
$\overline{\mathcal{T}}^*(Rep) \vartriangle \overline{\mathcal{T}}^*(Dem) = \exists faitDe.\overline{Bois}$
$\overline{\mathcal{T}}^*(Dem) \vartriangle \overline{\mathcal{T}}^*(Rep) = \exists faitDe.(\exists contient.(Matériau \sqcap \overline{Carbone})) \sqcap$
$\exists aType.\overline{Micromètre}$
Après nettoyage, on obtient bien les résultats recherchés :
$Rep \ominus_{\mathcal{T}} Dem = \exists faitDe.Bois$
$Dem \ominus_{\mathcal{T}} Rep = \exists faitDe.(\exists contient.Carbone) \sqcap \exists aType.Micromètre$

Pour faciliter la compréhension de cet exemple, la figure 3.3 (p.54) montre les arbres de description des concepts associés.

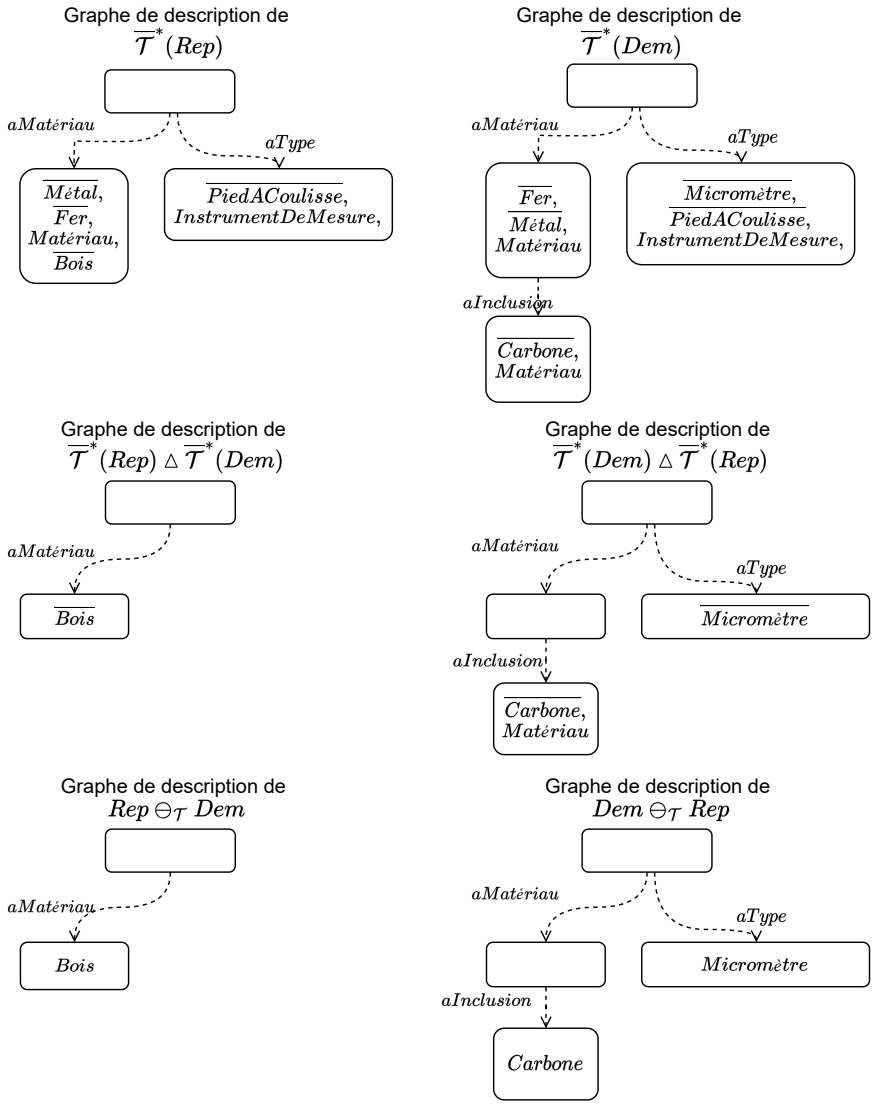


FIGURE 3.3 – Exemple de CSO avec une TBox contenant des définitions de concept et des définitions de concept primitif. On montre également les arbres de description des concepts impliqués.

3.2 Heuristique pour TBox générale

Nous proposons maintenant une approche heuristique qui permet d'étendre l'approche du CSO aux TBox acycliques générales. L'expansion complète n'étant pas définie en présence de GCI, nous proposons un procédé de saturation remplissant le même rôle que l'expansion complète pour des TBox acycliques générales et ainsi utiliser le CSO en présence de GCI. On l'appelle le "fill-up" et on le note $Fup_{\mathcal{T}}(A)$. Ainsi, le CSO $A \ominus_{\mathcal{T}} B$

de deux concepts A et B, devient avec cette heuristique : $Fup_{\mathcal{T}}(A) \vartriangle Fup_{\mathcal{T}}(B)$.

On présente dans la section 3.2.1 la notion de fill-up et nous donnons ses propriétés dans la section 3.2.2. Nous proposons ensuite une approximation du CSO utilisant le fill-up dans la section 3.2.3.

3.2.1 Notion de fill-up

Informellement, le fill-up d'un concept C est un concept équivalent à C composé de toutes les branches possibles qui subsument C.

Definition 24. *Soit C un concept $\mathcal{EL}$ et $\mathcal{T}$ une TBox générale et acyclique, $Fup_{\mathcal{T}}(C)$ est un concept tel que*

1. $C \equiv_{\mathcal{T}} Fup_{\mathcal{T}}(C)$
2. $\mathsf{br}_{Fup_{\mathcal{T}}(C)} = \{S \in \mathsf{br}_{\mathcal{T}} \mid C \sqsubseteq_{\mathcal{T}} S\}$

En général, il existe plusieurs concepts qui sont les $Fup_{\mathcal{T}}(C)$, qui ont tous le même ensemble de branches. Les algorithmes 5 (p.56) et 6 (p.56) nous en donnent un. Dans la suite, quand nous évoquerons le fill-up, ce sera par référence à ce dernier.

Exemple 21. *Avec la TBox de de l'exemple 1 (p.5), pour les concepts*
$Dem_1 \equiv \exists aType.PaC \sqcap \exists aMatériau.Acier \sqcap \exists aFabricant.Fab{-}Français \sqcap \exists aUnité.Mm$,
$Dem_2 \equiv \exists aType.Micromètre \sqcap \exists aMode.Analogique \sqcap \exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne) \sqcap \exists aFabricant.Fab - Européen$ *et*
$Rep_1 \equiv \exists aType.PaC \sqcap \exists aMode.Analogique \sqcap \exists aUnité.Cm \sqcap \exists aFabricant.Fab{-}Allemand$
on a les fill-up suivants :

$Fup_{\mathcal{T}}(Dem_1) = \exists aType.(PaC \sqcap Type \sqcap \top) \sqcap$
$\exists aMatériau.(Acier \sqcap Métal \sqcap Matériau \sqcap \top) \sqcap$
$\exists aFabricant.(Fab - Français \sqcap Fab - Européen \sqcap Region \sqcap \top) \sqcap$
$\exists aUnité.(Mm \sqcap Unité \sqcap \top) \sqcap$
$\top$

$Fup_{\mathcal{T}}(Dem_2) = \exists aType.(Micromètre \sqcap Pac \sqcap Type \sqcap \top) \sqcap$
$\exists aMode.(Analogique \sqcap ModeLecture \sqcap \top) \sqcap$
$\exists aMatériau.(Plastique \sqcap Métal \sqcap Chêne \sqcap Bois \sqcap Matériau \sqcap \top) \sqcap$
$\exists aFabricant.(Fab - Européen \sqcap Region \sqcap \top) \sqcap$
$\top$

$Fup_{\mathcal{T}}(Rep_1) = \exists aType.(PaC \sqcap Type \sqcap \top) \sqcap$
$\exists aMode.(Analogique \sqcap ModeLecture \sqcap \top) \sqcap$
$\exists aUnité.(Cm \sqcap Unité \sqcap \top) \sqcap$
$\exists aFabricant.(Fab - Allemand \sqcap Fab - Européen \sqcap Region \sqcap \top) \sqcap$
$\top$

Nous proposons l'algorithme 5 pour calculer ce fill-up. Son principe est le suivant : après avoir calculé la classification de la TBox, on parcourt le concept C dont on veut calculer fill-up$(\mathcal{T}, C)$. On ajoute alors chaque nom de concept qui subsume C en tant que conjonct à C, sauf les noms de concepts équivalents à une restriction existentielle $\exists r.D$

pour lesquels fill-up($\mathcal{T}, D$) et on ajoute ensuite à C le conjonct $\exists r$.fill-up($\mathcal{T}, D$). Comme on veut ajouter à C toutes les branches qui subsument C, on est obligé de renommer toutes les restrictions existentielles et les conjonctions que l'on trouve dans $\mathcal{T}$ afin de les prendre en compte dans la classification. Pour ce faire, on a besoin des fonctions rename-input() et rename-existential(). On a enfin besoin de la fonction clean-up() pour présenter le résultat sans les noms de concepts générés pour le calcul du fill-up. Une trace d'une exécution de l'algorithme est présentée dans l'annexe C.4 (p.192). Nous présentons maintenant les détails de ces trois fonctions.

Algorithm 5 fill-up($\mathcal{T}, C$).

Require: • une TBox $\mathcal{EL}$ acyclique normalisée $\mathcal{T}$.
• un concept $\mathcal{EL}$ C.

Ensure: Un $Fup_{\mathcal{T}}(C)$ (voir définition 24 (p.55)).

1: **if** $C \notin \mathbf{C}$ (c'est-à-dire $C = ... \sqcap ...$ ou $C = \exists ...$) **then**
2: $(\mathcal{T}^0, C^0)$:= rename-input($\mathcal{T}, C$)
3: $C' := C^0$
4: **else**
5: $\mathcal{T}^0 = \mathcal{T}$
6: $C' := C$
7: **end if**
8: Calcul de $\mathcal{T}^0_{Bnf}$
9: $\mathcal{T}'$:= rename-existential($\mathcal{T}^0_{Bnf}$)
10: Calcul de la classification $Cl_{\mathcal{T}'}$ de $\mathcal{T}'$.
11: $Prefup$:= build-fill-up($\mathcal{T}', Cl_{\mathcal{T}'}, C'$)
12: Retourner clean-up($\mathcal{T}, C, Prefup$)

Algorithm 6 build-fill-up($\mathcal{T}', Cl_{\mathcal{T}'}, C'$).

Require: • une TBox $\mathcal{EL}$ normalisée et acyclique $\mathcal{T}'$ obtenue dans fill-up($\mathcal{T}, C$).
• la classification $Cl_{\mathcal{T}'}$ de $\mathcal{T}'$ (voir définition en fin de partie 1.1).
• un nom de concept $\mathcal{EL}$ C'.

Ensure: Un $Fup_{\mathcal{T}'}(C')$ (voir la définition 24 (p.55))

1: $E_1 := \bigsqcap_{Z \in \mathbf{S}(C') \mid Z \neq X_{\exists r.D}} Z$
2: $E_2 := \bigsqcap_{Z \in \mathbf{S}(C') \mid Z = X_{\exists r.D}} \exists r$.build-fill-up($\mathcal{T}', Cl_{\mathcal{T}'}, D$)
3: Retourner $E_1 \sqcap E_2$

— rename-input($\mathcal{T}, C$) retourne le couple $(\mathcal{T}^0, C^0)$. $(\mathcal{T}^0, C^0)$ est une transformation de $\mathcal{T}$ et de C où toutes les conjonctions et restrictions existentielles E dans C ont été remplacées partout dans C et $\mathcal{T}$ par un nouveau nom de concept X_E, et les deux axiomes $X_E \sqsubseteq E$ et $E \sqsubseteq X_E$ ont été ajoutés à $\mathcal{T}$. Si E_1 est une sous-description de E_2 alors X_{E_2} doit mentionner X_{E_1} dans son nom.

Par exemple si $\mathcal{T} = \{\exists r.(A \sqcap B) \sqsubseteq F\}$ et $C = \exists r.(A \sqcap B)$, alors $X_{A \sqcap B}$ et $X_{\exists r.X_{A \sqcap B}}$ sont créés. Alors $\mathcal{T}^0 = \{X_{\exists r.X_{A \sqcap B}} \sqsubseteq F, X_{A \sqcap B} \sqsubseteq A \sqcap B, A \sqcap B \sqsubseteq$

$X_{A\sqcap B}, X_{\exists r.X_{A\sqcap B}} \sqsubseteq \exists r.(A \sqcap B), \exists r.(A \sqcap B) \sqsubseteq X_{\exists r.X_{A\sqcap B}}\}$ et $C^0 = X_{\exists r.X_{A\sqcap B}}$.

— rename-existential($\mathcal{T}$) retourne une transformation de $\mathcal{T}$ où toutes les restrictions existentielles $\exists r.D$ ont été remplacées par un concept associé $X_{\exists r.D}$ partout excepté dans les axiomes $X_{\exists r.D} \sqsubseteq \exists r.D$ et $\exists r.D \sqsubseteq X_{\exists r.D}$. Si ce n'est pas le cas, la fonction crée $X_{\exists r.D}$ si ce n'est pas déjà fait, ajoute les deux axiomes si ce n'est pas déjà fait et termine les remplacements si nécessaire (en dehors de ces deux axiomes)

— clean-up($\mathcal{T}, C, D$) retourne le concept $\mathcal{EL}$ D' construit à partir de D où tous les noms de concept de D qui ne sont ni dans $\mathcal{T}$ ni dans C ont été retirés, et où les différentes occurrences de $\top$ dans D ont été retirées, en s'assurant que toutes les conjonctions n'importe où dans D' ne possèdent aucun conjonct $\top$ en conjonction avec des noms de concept, ou exactement un conjonct $\top$ dans le cas contraire.

3.2.2 Propriétés du fill-up

Nous donnons maintenant les propriétés liées aux algorithmes 5 (p.56) et 6 (p.56). Ceux-ci donnent toujours un résultat unique, conforme à la définition de $Fup_{\mathcal{T}}(C)$, en un temps qui est dans EXPTIME.

Proposition 4 (Propriétés du fill-up et des algorithmes 5 (p.56) et 6 (p.56))**.** *Soit $\mathcal{T}$ une TBox normalisée et acyclique $\mathcal{EL}$ TBox, et C un concept $\mathcal{EL}$. On a :*

a. $Fup_{\mathcal{T}}(C)$ existe toujours.

b. Les algorithmes 5 (p.56) et 6 (p.56) se terminent et donnent un résultat unique.

c. fill-up($\mathcal{T}, C$) respecte les propriétés 1. et 2. de la définition de $Fup_{\mathcal{T}}(C)$.

d. Le calcul de $Fup_{\mathcal{T}}(C)$ est dans EXPTIME par rapport aux tailles de $\mathcal{T}$ et C.

Elements de preuve. L'existence de $Fup_{\mathcal{T}}(C)$ provient du fait qu'on peut toujours construire au moins le fill-up suivant : $C \sqcap (\bigsqcap_{S\in \mathsf{br}_C | C\sqsubseteq_{\mathcal{T}} S)}$ La terminaison est facile à obtenir par récurrence puisque la TBox est acyclique, impliquant que les appels récursifs sont toujours appliqués à des descriptions qui sont plus profondément imbriquées (et donc plus petites que l'entrée initiale). L'unicité du résultat vient du fait qu'il n'y a pas d'indéterminisme dans l'algorithme, modulo la conjonction. La justesse vient principalement de la complétude de la classification (utilisée dans une preuve par récurrence). La complexité vient du fait que la taille de $Fup_{\mathcal{T}}(C)$ peut être exponentielle par rapport aux tailles de $\mathcal{T}$ et C. □

Les preuves de la proposition 4 se trouvent dans l'annexe C.3 (p.189).

3.2.3 Approximation du CSO avec le fill-up

On utilise maintenant le fill-up pour calculer une approximation du résultat (et non de la définition) du CSO comme suggéré en introduction du chapitre 3.2. On propose de modifier l'algorithme 4 (p.51) du CSO pour utiliser le fill-up à la place de l'expansion complète : on obtient l'algorithme 7 (p.59). Cet algorithme est une heuristique car nous

n'avons pas de définition du CSO pour une TBox acyclique générale. Il faudrait adapter la notion de branche primitive à un tel environnement pour pouvoir définir le CSO pour des TBox acycliques générales. Voyons maintenant deux exemples montrant que cette heuristique génère un résultat proche mais différent du résultat attendu.

Exemple 22. *Comme dans l'exemple 20 (p.52) on a la TBox acyclique générale, la réponse et la demande suivantes :*
$\mathcal{T} = \{Acier \equiv Fer \sqcap \exists aInclusion.Carbone,$
$Fer \sqsubseteq Métal,$
$Métal \sqsubseteq Matériau,$
$Bois \sqsubseteq Matériau,$
$Carbone \sqsubseteq Matériau,$
$Micromètre \sqsubseteq PaC,$
$PaCNumérique \sqsubseteq PaC,$
$PaC \sqsubseteq InstrumentDeMesure\}$
$Dem = \exists aMatériau.Acier \sqcap$
$\exists aType.Micromètre$
$Rep = \exists aMatériau.(Fer \sqcap Bois) \sqcap$
$\exists aType.PaC$
On attend comme résultats ceux obtenus dans l'exemple 20 :
$Dem \ominus_{\mathcal{T}} Rep = \exists aMatériau.\exists aInclusion.Carbone \sqcap \exists aType.Micromètre$
$Rep \ominus_{\mathcal{T}} Dem = \exists aMatériau.Bois$
Or, avec le fill-up :
$Fup_{\mathcal{T}}(Dem) = Dem \sqcap$
$\top \sqcap$
$\exists aMatériau.(Acier \sqcap Fer \sqcap Métal \sqcap Matériau \sqcap \exists aInclusion(Carbone \sqcap Matériau \sqcap \top) \sqcap \top) \sqcap$
$\exists aType.(Micromètre \sqcap PaC \sqcap InstrumentDeMesure \sqcap \top)$
$Fup_{\mathcal{T}}(Rep) = Rep \sqcap$
$\top \sqcap$
$\exists aMatériau.(Fer \sqcap Bois \sqcap Métal \sqcap Matériau \sqcap \top) \sqcap$
$\exists aType.(PaC \sqcap InstrumentDeMesure \sqcap \top)$
$Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep) = Dem \sqcap$
$\exists aMatériau.(Acier \sqcap \exists aInclusion(Carbone \sqcap Matériau \sqcap \top)) \sqcap$
$\exists aType.Micromètre$
$Fup_{\mathcal{T}}(Rep) \vartriangle Fup_{\mathcal{T}}(Dem) = Rep \sqcap$
$\exists aMatériau.Bois$
On constate que les branches Dem, Rep, $\exists aMatériau.Acier$, $\exists aMatériau.aInclusion.Matériau$ *et* $aMatériau.aInclusion.\top$ *sont toujours présentes dans le résultat et il faudrait les retirer pour obtenir le résultat escompté.*

Exemple 23. *Soient la TBox acyclique définitionnelle, la réponse et la demande suivantes :*
$\mathcal{T} = \{A \equiv B \sqcap C, C \equiv D \sqcap \exists r.B, D \equiv E, F \equiv \exists r.D\}$

$Rep = A \sqcap F \sqcap \exists r.\top \sqcap \exists s.\top$ *et* $Dem = B \sqcap \exists r.D \sqcap \exists s.E$
On attend comme résultats :
$Rep \ominus_{\mathcal{T}} Dem = E \sqcap \exists r.B$
$Dem \ominus_{\mathcal{T}} Rep = \exists s.E$

Avec le fill-up on a :
$Fup_{\mathcal{T}}(Rep) = Rep \sqcap \top \sqcap A \sqcap B \sqcap C \sqcap D \sqcap E \sqcap \exists r.(B \sqcap \top) \sqcap F \sqcap \exists r.(D \sqcap E \sqcap \top)$
$Fup_{\mathcal{T}}(Dem) = Dem \sqcap \top \sqcap B \sqcap F \sqcap \exists r.(E \sqcap D \sqcap \top) \sqcap \exists s.(E \sqcap D \sqcap \top)$
$Fup_{\mathcal{T}}(Rep) \vartriangle Fup_{\mathcal{T}}(Dem) = Rep \sqcap A \sqcap C \sqcap D \sqcap E \sqcap \exists r.B$
$Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep) = Dem \sqcap \exists s.(E \sqcap D \sqcap \top)$
On a dans le résultat avec le fill-up des concepts supplémentaires : Rep, A, C, D *dans* $Rep \vartriangle Dem$ *et* Dem, $\exists s.D$ *et* $\exists s.\top$ *dans* $Dem \ominus_{\mathcal{T}} Rep$. *Il faudrait les retirer pour obtenir le résultat escompté.*

On constate que le résultat obtenu avec le fill-up est bien une approximation du CSO car il peut contenir des noms de concept ou $\top$ en plus du résultat attendu. En pratique dans $Fup_{\mathcal{T}}(C) \vartriangle Fup_{\mathcal{T}}(D)$ on peut effectuer le nettoyage basique suivant : on supprime les $\top$ qui ne sont pas seuls conjoncts dans une conjonction et on supprime C du résultat. Les autres branches superflues par rapport au résultat souhaité peuvent quant à elles, être conservées et exploitées dans le calcul du CCO.

Par exemple, si $Fup_{\mathcal{T}}(Dem) = Dem \sqcap \top \sqcap \exists aMatériau.(Chêne \sqcap Bois \sqcap Matériau \sqcap \top)$, $Fup_{\mathcal{T}}(Rep_1) = Rep_1 \sqcap \top \sqcap \exists aMatériau.(Cerisier \sqcap Bois \sqcap Matériau \sqcap \top)$ et $Fup_{\mathcal{T}}(Rep_2) = Rep_2 \sqcap \top \sqcap \exists aMatériau.(Fer \sqcap Métal \sqcap Matériau \sqcap \top)$,
on constate que $Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep_1) = Dem \sqcap \exists aMatériau.Cerisier$
et $Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep_2) = Dem \sqcap \exists aMatériau.(Cerisier \sqcap Bois)$.
$Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep_1) \equiv_{\mathcal{T}} Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep_2)$.
Sémantiquement les deux résultats sont identiques, mais $\mathsf{size}(Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep_1)) < \mathsf{size}(Fup_{\mathcal{T}}(Dem) \vartriangle Fup_{\mathcal{T}}(Rep_2))$. Rep_1 est donc meilleure que Rep_2 par rapport à Dem car la taille du CSO est plus grande pour Rep_2. Intuitivement $\exists aMatériau.Bois$ est un point commun de plus entre Dem et Rep_1 par rapport Dem et Rep_2. On peut donc tirer parti du résultat approximé du calcul du CSO avec le fill-up pour des TBox acyclique générale. C'est comme cela que le CSO est implémenté dans le chapitre 5 (p.73).

Algorithm 7 approx-cso($\mathcal{T}, C, D$).

Require: • $\mathcal{T}$ une TBox $\mathcal{EL}$ générale, acyclique et normalisée.
• C et D deux concepts $\mathcal{EL}$.
Ensure: une approximation de $C \ominus_{\mathcal{T}} D$.
1: Retourner tso(fill-up($\mathcal{T}, C$), fill-up($\mathcal{T}, D$)) dont on a retiré C et les $\top$ superflus.

Proposition 5 (Propriétés de l'approximation du CSO avec le fill-up et de l'algorithme 7 (p.59))**.** *Soit* $\mathcal{T}$ *une TBox* $\mathcal{EL}$ *générale, normalisée et acyclique et* C *et* D *deux concepts* $\mathcal{EL}$. *On a :*

a. approx-cso$(\mathcal{T}, C, D)$ existe toujours, se termine toujours et a un résultat unique.
b. L'algorithme 7 (p.59) est dans PTIME en fonction de fill-up$(\mathcal{T}, C)$, fill-up$(\mathcal{T}, D)$ et $\mathcal{T}$ et est dans EXPTIME en fonction des tailles de C, D et $\mathcal{T}$.

Elements de preuve. a. existence et unicité et terminaison proviennent directement des algorithmes tso() et fill-up().

b. La complexité provient de la complexité du calcul du fill-up qui est EXPTIME en fonction des tailles de C, D et $\mathcal{T}$. L'algorithme tso() étant polynomial par rapport aux tailles de C et D, la complexité de l'algorithme 7 (p.59) est dans PTIME si les fill-up sont déjà calculés. □

La validation expérimentale de l'heuristique d'approximation du CSO pour des TBox générales et acycliques a été réalisée via les tests présentés au chapitre 6 (p.89). Les tests qualitatifs se sont montrés conformes aux résultats attendus (en tenant compte de l'approximation). Les résultats quantitatifs nous renseignent sur les étapes coûteuses de l'approche.

Chapitre 4

Perfect Memory au sein du projet STAM

Dans le cadre du projet STAM, l'heuristique présentée au chapitre chapter :CSO a été ajoutée à la plateforme Perfect Memory, afin d'être en mesure de répondre à une demande d'un utilisateur qui ne possède pas de réponses exactes. Nous donnons les détails de cette implementation dans le chapitre suivant et présentons dans ce chapitre la plateforme de Perfect Memory. Perfect Memory est une entreprise clermontoise qui propose à ses clients une plateforme de gestion de données et de médias. L'entreprise est spécialisée dans la valorisation sémantique des données, c'est-à-dire l'utilisation de la sémantique pour extraire de la valeur des données manipulées. Son rôle au sein du projet STAM est de fournir une plateforme de recherche de médias (texte, image, vidéo, audio...) en rapport avec la métrologie. Nous présentons dans à la section 4.1 leur plateforme de recherche et son fonctionnement. Nous proposons ensuite dans la section 4.2 les moyens d'intégrer le CCO dans la plateforme et les besoins associés.

4.1 La plateforme Perfect Memory

Perfect Memory propose à ses clients une plateforme de recherche qui se base sur trois ontologies. Elles sont initialement communes à toutes les plateformes et sont nécessaires aux différentes fonctionnalités de recherche de la plateforme (voir la section 4.1.1 (p.62)). Ces ontologies sont ensuite spécialisées avec le client pour y intégrer son domaine d'expertise, ce qui permet la spécificité de chaque plateforme par rapport aux autres .

Les plateformes des clients de Perfect Memory sont en mesure d'ingérer différents types de médias et d'en extraire sémantiquement des informations. Ainsi, pour chaque type de média, la plateforme est en mesure de contextualiser et d'annoter les données. Parmi les procédés d'extraction de données, on trouve entre autres :

— la reconnaissance d'entité,
— l'analyse textuelle, pour des documents ou pour des médias contenant de l'audio (textualisé via speech2text) ou des sous-titres et

— l'analyse d'images.

Les données extraites sont stockées dans les bases de données du système de gestion de données graphes GraphDB [9] et indexées dans le moteur de recherche ElasticSearch [10]. Par ailleurs, le processus de recherche est enrichi par l'utilisation de certaines données sous forme de tags et de facettes (voir section 4.1.3 (p.65)). La recherche s'effectue via une API : Wisdom. C'est cette API qui va interroger les bases de données de la plateforme : GraphDB est une base triple store qui contient toutes les données de la plateforme. ElasticSearch est un moteur de recherche performant qui, en se synchronisant à la base de GraphDB, offre à l'utilisateur de nombreuses possibilités lors de ses recherches. Leur fonctionnement individuel et conjoint est présenté dans la section 4.1.2 (p.62)

4.1.1 Les ontologies de la plateforme STAM

La plateforme STAM utilise les trois ontologies principales de Perfect Memory et ajoute à celles-ci l'ontologie du domaine de la métrologie. On compte ainsi :

— L'ontologie PMCore, qui regroupe les classes de gestion des documents et de ses annotations et de gestion des concepts qui peuvent être contenus dans les documents.
— L'ontologie PMMultimedia, qui regroupe les classes de gestions des médias et des annotations qui y sont liées.
— L'ontologie PMBroadcast, qui complète l'ontologie précédente pour gérer le contenu diffusable et diffusé.

Ces ontologies sont en langages OWL et sont volontairement gardées aussi petites que possible. On compte parmi les constructeurs principaux utilisés les conjonctions $\sqcap$, les restrictions existentielles $\exists$ et les restrictions numériques $\{>, <, =, \geq, \leq\}n$.

Une ontologie STAM (voir figure 4.1 (p.63)) est ensuite créée pour s'intégrer aux trois existantes. Elle va spécialiser les trois ontologies Perfect Memory, en définissant par exemple les types de document que devra gérer PMCore. Cette ontologie a été réalisée par Perfect Memory avec DeltaMU, l'entreprise experte de la métrologie.

4.1.2 ElasticSearch et GraphDB

Pour fonctionner, les plateformes de Perfect Memory utilisent un moteur de recherche, ElasticSearch et une base de données sémantique en graphes, GraphDB. Ces deux outils sont gérés via l'intermédiaire de l'API de Perfect Memory, Wisdom. C'est cette API qui va gérer toutes les demandes de modification de la base de données, et également les recherches des utilisateurs.

GraphDB est une base de données sémantique, proposée par Ontotext [11], utilisant des graphes de triplets RDF, basée sur RDF4J [12], un framework Java de gestion de données

9. `https://graphdb.ontotext.com/`
10. `https://www.elastic.co/fr/elasticsearch`
11. `https://graphdb.ontotext.com/`
12. `https://rdf4j.org/`

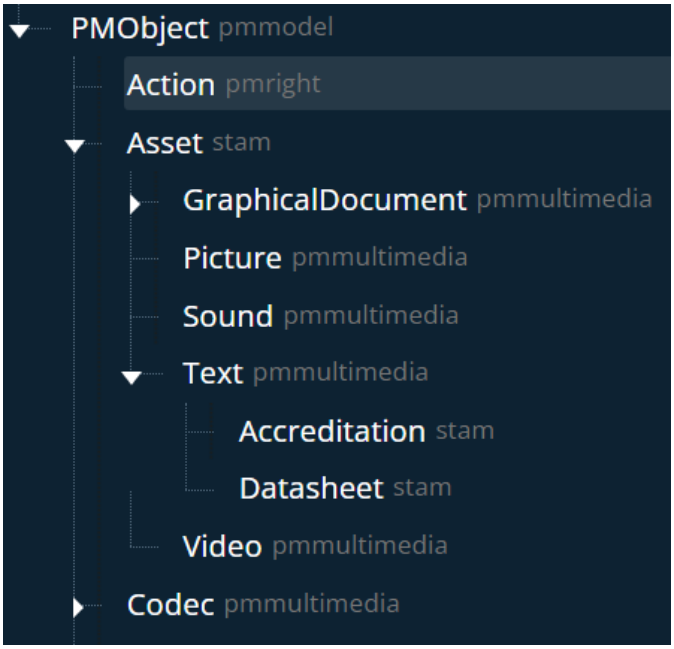


FIGURE 4.1 – Partie des ontologies de la plateforme STAM : on peut voir que les concepts de "Datasheet" et d'"Accreditation", spécifiques au contexte de métrologie, spécialisent le concept de "Text" présent dans une des ontologies principales de Perfect Memory, PMMultimedia.

RDF. GraphDB s'interroge et se modifie via des requêtes SPARQL [13]. Toute insertion de ressources, documents ou médias sur la plateforme fait l'objet d'une insertion d'annotations décrivant la ressource, le document ou le média, dans le graphe. Les données des plateformes de Perfect Memory sont stockées dans ces graphes qui permettent les raisonnements suivants :

— les tests de subsomption entre les classes (uniquement entre noms de concept),
— les tests de subsomption entre les rôles (uniquement entre noms de rôle),
— la recherche de réponse à une requête qui contient avec composition de rôle,
— la recherche de réponse à une requête avec des jointures non exprimables en OWL.

ElasticSearch est un moteur de recherche. Pour fonctionner ElasticSearch nécessite d'être synchronisé avec la base de triplets GraphDB. Pour cela, on va indexer le contenu du graphe, pour créer des documents ElasticSearch dans lesquels le moteur pourra effectuer ses recherches. Un document ElasticSearch (voir l'exemple 24 (p.64)) présente une syntaxe JSON et associe des prédicats à leur valeur. L'indexation va également associer à chaque prédicat ses valeurs inférées par graphDB. Ainsi, dans l'exemple 24 (p.64), on constate que le prédicat Matériau est associé à la valeur Acier, mais également à la valeur Métal (car l'Acier est un Métal). Ce processus automatique effectue un travail de saturation dans le même esprit que celui proposé par le fill-up (voir section 5.5 (p.82)),

13. https://www.w3.org/TR/rdf-sparql-query/

pour les noms de concept.

Exemple 24. *{*
"rdfs :label" : ["Pied à coulisse"],
"stam :material" : ["stam-kb :Material Steel", "stam-kb :Material Metal"],
"stam :manufacturer" : ["uuid :5d9f61c6-5c35-47d9-b113-5b42eb9282ab"]
...
}
Dans ce document ElasticSearch, on trouve les informations relatives à un Pied à Coulisse en acier. Le document contient des informations inférées : puisque le pied à coulisse est en acier, il est en métal. Il contient également des liens vers d'autres ressources, ici dans le troisième prédicat ("stam :manufacturer"), un constructeur dont l'uuid est ici donné.

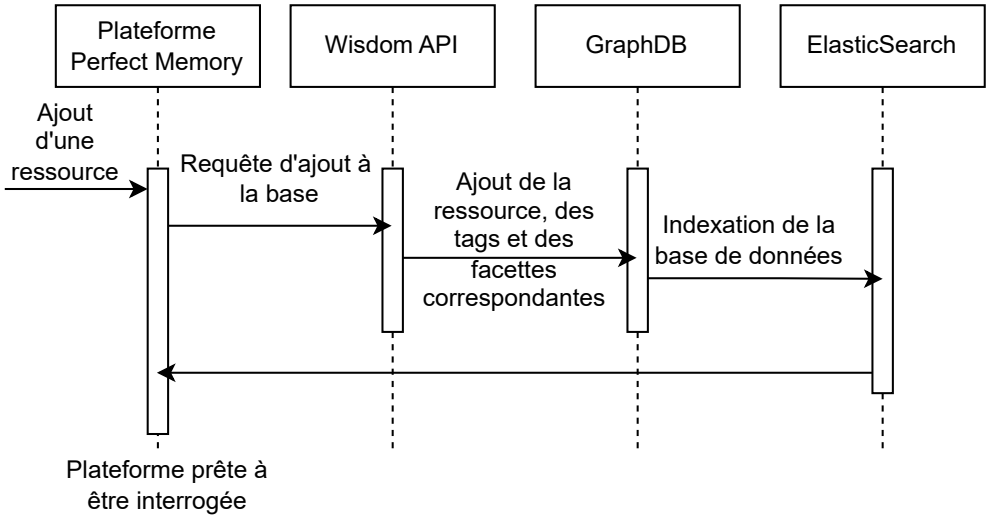


FIGURE 4.2 – Processus d'ajout d'une ressource à la plateforme.

L'indexation réalisée pour ElasticSearch est un processus réalisé hors-ligne, de manière automatique lors de la modification de la base GraphDB ou manuellement si besoin (voir figure 4.2 (p.64)). Une fois celle-ci terminée, ElasticSearch n'a plus besoin de GraphDB pour fonctionner et effectue donc ses recherches sur les documents indexés sans avoir besoin d'effectuer de requête d'interrogation de la base, pour plus de performances (voir figure 4.3 (p.65)).

L'association de GraphDB et d'ElasticSearch permet aussi de proposer un système de recherche original qui n'est pas uniquement textuel. La plateforme propose ainsi l'utilisation de filtres basés sur la notion de tags et de facettes pour affiner sa recherche. C'est l'objet de la section suivante.

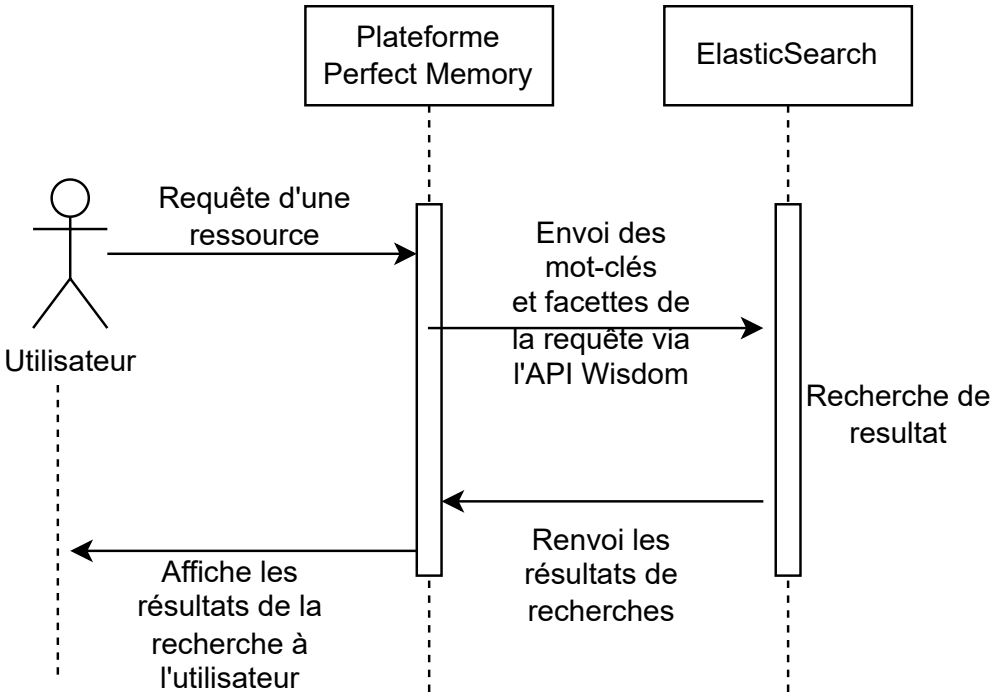


FIGURE 4.3 – Processus de recherche par un utilisateur.

4.1.3 Tags et facettes

Les tags sont des mots-clés décrivant le contenu des ressources stockées dans une plateforme. Lors de l'intégration d'une ressource, son contenu est parcouru (et textualisé si besoin) et des mots-clés sont détectés. Pour l'utilisateur, ils sont surlignés et comptés dans la version textuelle du média. Par exemple, sur la figure 4.4 (p.66), à gauche, on constate que les tags "Cale étalon", "acier', 'ISO" et "μm" ont été détectés et surlignés dans le texte. À droite, on a la liste des tags. Parmi les 144 tags du texte, 41 sont "μm", 15 "accréditation" et "Comité français d'accréditation" et 14 "acier".

Cliquer sur un tag permet d'en avoir les caractéristiques (voir figure 4.5 (p.66)). Enfin, l'indexation permet à ElasticSearch de déterminer des partitions des ressources ayant un rapport avec un sujet commun. Ces partitions sont appelées des facettes, et permettent à l'utilisateur d'affiner petit à petit sa recherche. Par exemple, dans la figure 4.6 (p.67), au cours de sa recherche, un utilisateur a affiné celle-ci en sélectionnant "Dimensionnel" et "Instruments manuels à cotes variables" de la facette "Famille". Ainsi, les résultats affichés sont ceux appartenant à ces deux partitions. L'utilisateur peut par la suite affiner plus encore sa recherche en sélectionnant une des familles proposées. Par exemple, sélectionner "Étalons de circularité" réduira la recherche à 8 résultats.

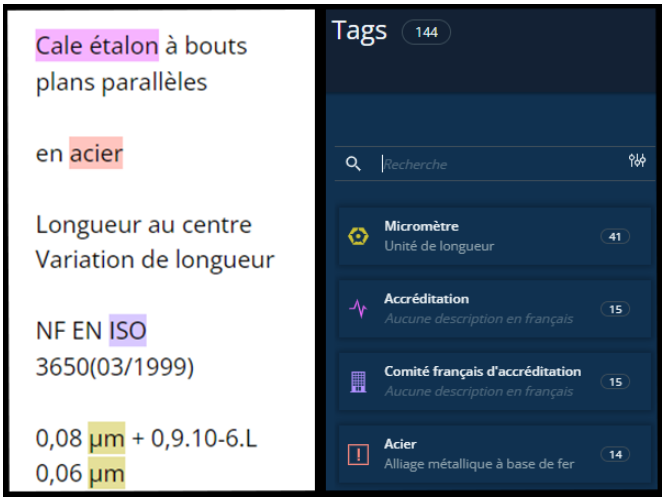


FIGURE 4.4 – Tags extraits d'un document d'accréditation de métrologie par la plateforme Perfect Memory et surlignés dans le texte.

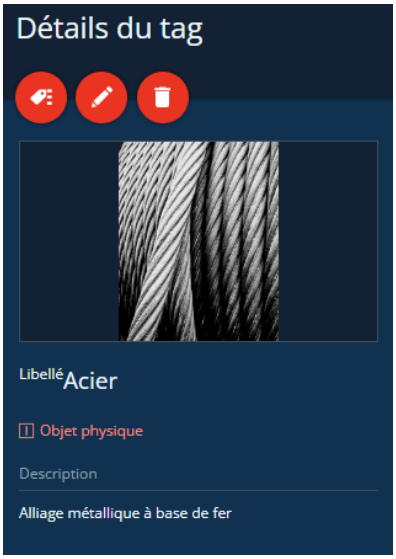


FIGURE 4.5 – Détails du tag Acier dans la plateforme Perfect Memory : à partir du tag on obtient l'information que l'acier est un objet physique, ainsi que sa description comme un alliage métallique à base de fer.

4.2 Workflow et intégration du CCO

Pour pouvoir s'intégrer à la plateforme, le CCO nécessite une ontologie du domaine pour fonctionner. En effet, le CCO doit pouvoir utiliser une ontologie décrivant le domaine des médias stockés. C'est donc une cinquième ontologie en plus des trois ontologies

FIGURE 4.6 – Exemple de la facette "Famille" dans la plateforme Perfect Memory. Ici, à partir d'une recherche textuelle, on obtient une liste de familles avec le nombre de documents associés. Après avoir sélectionné les familles "Dimensionnel" et "Instruments manuels à cotes variables", la plateforme propose d'autres familles pour affiner la sélection déjà effectuée.

de Perfect Memory pour la gestion de document et l'ontologie du client pour la gestion des documents spécifiques à son domaine.

On rappelle que la plateforme fonctionne en deux temps. Une première étape permet à la plateforme de se préparer à répondre aux requêtes des utilisateurs. Suite à une modification de la base de données graphDB, par exemple avec l'ajout d'une ressource, ElasticSearch indexe la base pour créer les documents ElasticSearch qui serviront lors de la recherche. Nous proposons d'intégrer le module CCO une fois l'indexation terminée (voir figure 4.7 (p.68)) lors de la phase de "préparation". À l'aide d'un appel d'API, ElasticSearch confirme le lancement du CCO qui pourra récupérer la TBox pour la normaliser, la classifier, et calculer les fill-up des ressources en offline. Une fois la préparation du CCO terminée, le module prévient la plateforme qu'il est prêt à fonctionner et se met en attente.

La seconde étape de la plateforme commence quand un utilisateur envoie une requête à la plateforme. C'est la phase de traitement. Celle-ci interroge alors ElasticSearch pour ensuite retourner les réponses à l'utilisateur. Nous proposons d'intégrer le module CCO dans le processus de réponse quand ElasticSearch ne retourne aucune réponse à la requête de l'utilisateur. La plateforme envoie alors la requête au module du CCO pour en obtenir

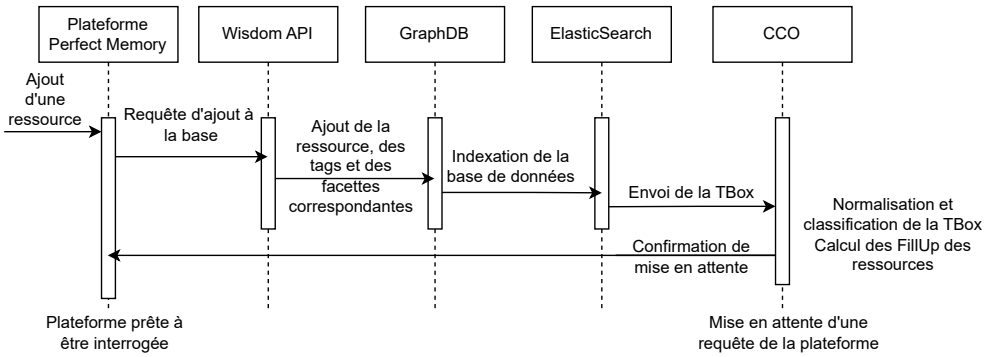


FIGURE 4.7 – Intégration du CCO à la plateforme de recherche.

les résultats approchés à afficher à l'utilisateur (voir figure 4.8 (p.69)).

L'implémentation actuelle n'est pas intégrée à la plateforme Perfect Memory. Elle est cependant en mesure d'interroger Wisdom pour récupérer l'ontologie et les réponses à donner aux requêtes. Une petite ontologie de test intégrée à la plateforme Perfect Memory a permis de vérifier le bon fonctionnement de la communication entre Wisdom et le module CCO via appel d'API. Plus précisément, l'étape "Confirmation de mise en attente" de la figure 4.7 et l'étape "requête d'un résultat approché" et "renvoi de la liste ordonnée [...]" de la figure 4.8 n'ont pas été réalisée. C'est-à-dire que le CCO est en mesure de récupérer les informations dont il a besoin pour son fonctionnement, est en mesure de donner le résultat attendu si on l'interroge, mais la plateforme Perfect Memory n'a pas intégré le module permettant la communication dans les deux sens et donc d'ajouter le CCO à son processus de recherche. Cela n'a pas été fait car les acteurs du projet n'ont pas encore eu le temps de réaliser l'ontologie du domaine nécessaire au fonctionnement du CCO sur la plateforme.

Pour rappel, l'objectif du CCO est de permettre à la plateforme de retourner des réponses proches d'une demande utilisateur en l'absence de résultats exacts :

Exemple 25. *Un utilisateur demande à la plateforme un pied à coulisse en plastique :*
$Dem = \exists aType.PaC \sqcap \exists aMatériau.Plastique \sqcap \exists aMode.Numerique.$
La plateforme ne possède pas de réponse exacte à cette demande et elasticSearch ne renvoie donc aucun résultat.
La plateforme interroge alors le CCO qui va comparer les réponses disponibles :
$Rep_1 = \exists aType.PaC \sqcap \exists aMatériau.Acier \sqcap \exists aMode.Numerique.$
$Rep_2 = \exists aType.Regle \sqcap \exists aMatériau.Plastique.$
Le CCO obtient le classement des réponses ($Rep_1 \succsim Rep_2$) *et transmet le résultat à la plateforme qui peut alors afficher les réponses de la meilleure à la moins bonne à l'utilisateur.*

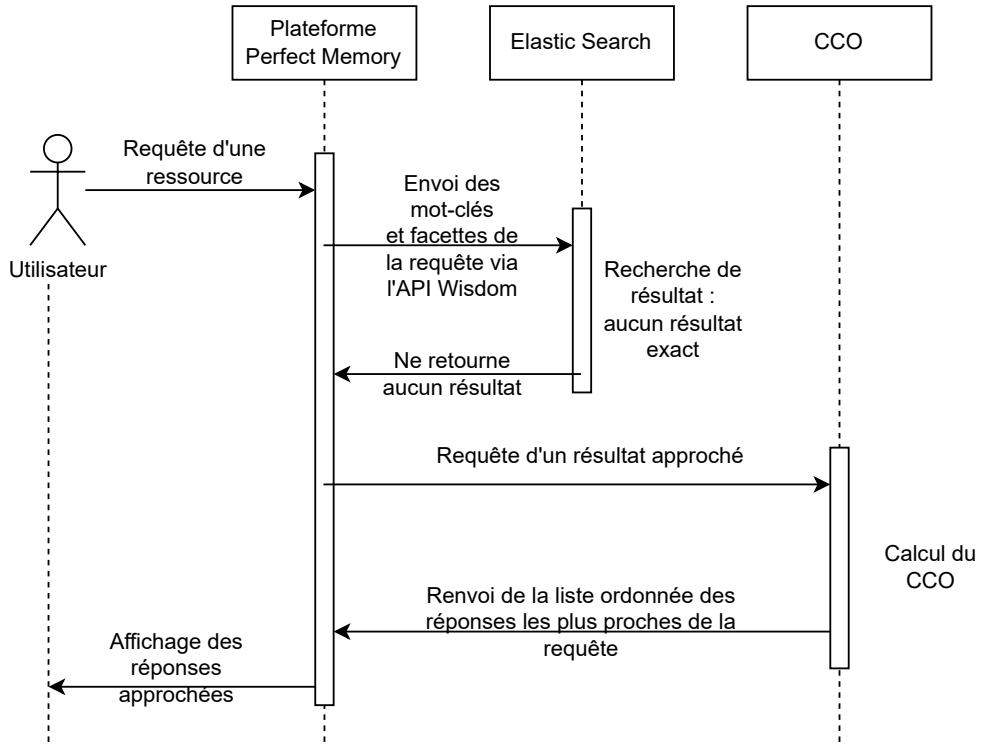


FIGURE 4.8 – Intégration du CCO au processus de recherche.

Partie deux : Implémentation

Chapitre 5

Implémentation du CCO

Dans ce chapitre, nous présentons l'implémentation d'une heuristique du CCO [14] présenté à la section 2 (p.35), réalisée en Ruby, avec pour objectif son utilisation par la plate forme de Perfect Memory. Nous présentons d'abord les notations utilisées pour représenter les logiques de descriptions dans le code dans la section 5.1 (p.73). Ensuite, chaque classe Ruby, représentant un algorithme ou une heuristique, est étudiée. L'implémentation de la normalisation selon Baader et al. (2017) est présentée à la section 5.3 (p.75), l'implémentation de la classification selon Baader et al. (2017) est présentée à la section 5.4 (p.79), l'implémentation du fill-up proposé dans la section 3.2.1 (p.55) est présentée dans la section 5.5 (p.82), l'implémentation du TSO proposé dans la section 2 (p.51) est présentée dans la section 5.6 (p.83), et l'implémentation du CCO proposé dans la section 2 (p.35) est présentée dans la section 5.7 (p.84). Enfin l'implémentation du programme principal s'occupant de réaliser toutes les opérations nécessaires au CCO est présentée dans la section 5.8 (p.86)

5.1 Notation

La représentation des symboles de la LD $\mathcal{EL}$ n'étant pas possible au sein du code, des notations équivalentes sont utilisées et répertoriées dans le tableau 5.1 (p.74). Des exemples d'axiomes construits avec ces notations sont proposés dans le tableau 5.1 (p.74).

Un axiome est représenté par un tableau dont la dimension dépend de la profondeur (définition 2 (p.14)) des concepts présents dans l'axiome. Ainsi, un axiome ayant un concept de profondeur 4, aura dans sa représentation sous forme de tableau au moins une imbrication de 4 tableaux. La taille d'un concept peut également être obtenue facilement à l'aide la fonction flatten(). Cette fonction Ruby retourne un tableau de profondeur 1, en désinbriquant les tableaux imbriqués, comme dans l'exemple 26 (p.73).

Exemple 26. *Soit la réponse* $r_1 = A \sqcap \exists r.\exists s.(B \sqcap C)$*, qui est de taille 7. Sa représentation sous forme de tableau est :* `["A", "&", "*r.", ["*s", ["B", "&", "C"]]]`.

14. le code source est disponible sur github : https://github.com/Myakko/CCO-Implementation-These

Constructeurs et axiomes	Syntaxe	Symbole dans le code
Top	$\top$	`"Top"`
Nom de concept $\in \mathbf{C}$	A	`"A"`
Rôle $\in \mathbf{r}$	r	`"r"`
Conjonction	$... \sqcap ...$	`..., "&", ...`
Restriction existentielle	$\exists r.\ ...$	`"*r.",[...]`
Description de concept	...	`[...]`
GCI	$... \sqsubseteq ...$	`[..., "subs", ...]`
Définition	$... \equiv ...$	`[..., "=", ...]`

TABLE 5.1 – Représentation des symboles de la LD $\mathcal{EL}$ dans l'implémentation du CCO. $A \in \mathbf{C}$, $r \in \mathbf{r}$.

Sa représentation aplatie est : `["A", "&", "*r.", "*s", "B", "&", "C"]`*, qui est de taille 7.*

Constructeurs	Syntaxe
$A \sqsubseteq \top$	`["A", "subs", "Top"]`
$A \sqsubseteq B$	`["A", "subs", "B"]`
$A \equiv B$	`["A", "=", "B"]`
$A \sqsubseteq B \sqcap C$	`["A", "subs", "B", "&", "C"]`
$A \sqsubseteq \exists r.B$	`["A", "subs", "*r.", ["B"]]`
$A \sqsubseteq \exists r.(B \sqcap C)$	`["A", "subs", "*r.", ["B", "&", "C"]]`
$A \sqsubseteq \exists r.\exists s.B$	`["A", "subs", "*r.", ["*s.",["B"]]]`

TABLE 5.2 – Exemple de représentation d'axiomes dans l'implémentation du CCO. A, B et $C \in \mathbf{C}$, r et $s \in \mathbf{r}$.

Une TBox est donc représentée sous la forme d'un tableau de tableau. Chaque case du tableau de la TBox est le tableau d'un axiome. Un exemple de représentation est proposé dans l'exemple 39 en annexe B.3.1 (p.136).

Pour les besoins du CCO, nous devons normaliser et classifier selon Baader et al. (2005) la TBox. La classification d'une TBox $\mathcal{T}$ normalisée selon Baader et al. (2005) est l'ensemble des mappings $\mathbf{S}(A)$ et $\mathrm{R}(v)$, où $A \in \mathbf{C}$ et $v \in \mathbf{r}$. Le mapping $\mathbf{S}(A)$ est l'ensemble des noms de concept qui subsument A. Le mapping $\mathbf{R}(v)$ contient une partie des couples (A, B) tel que $A \sqsubseteq_{\mathcal{T}} \exists v.B$, avec $B \in \mathbf{C}$, nécessaires à la réalisation des mappings $\mathbf{S}(A)$. La classification d'une TBox est alors représentée sous la forme d'une table de hachage, comme présenté dans l'exemple 40 de l'annexe B.3.1 (p.136). La table de hachage contient une clé pour chaque nom de concept de $\mathcal{T}$, contenant le mapping $\mathbf{S}$ de ce concept. De plus, elle possède une clé "role", qui contient une seconde table de hachage, où chaque clé représente un rôle de $\mathbf{r}$, contenant le mapping $\mathbf{R}$ de celui-ci, comme présenté dans l'exemple 27 (p.75).

Exemple 27. *Soit la TBox $\mathcal{T}$ suivante : $\mathcal{T} = \{A \sqsubseteq \exists r.B, A \sqsubseteq \exists r.C, A \sqsubseteq \exists s.C\}$ Sa classification est la suivante :*
$\boldsymbol{S}(A) = \{A, \top\}$
$\boldsymbol{S}(B) = \{B, \top\}$
$\boldsymbol{S}(C) = \{C, \top\}$
$\boldsymbol{R}(r) = \{(A, B), (A, C)\}$
$\boldsymbol{R}(s) = \{(A, C)\}$

La représentation de cette classification dans l'implémentation sera alors :

```
{"A" => ["A"],
"B" => ["B"],
"C" => ["C"],
"rôle" => [ "r" => [["A", "B"], ["A", "C"]], "s" => [["A", "C"]]]}
```

5.2 Principe général de l'implémentation du CCO

On présente le diagramme de la figure 5.1 (p.76), composé de deux sous-diagrammes. Le premier, à gauche, décrit le processus de préparation du CCO, après l'indexation des documents par ElasticSearch. Le second, à droite, décrit le processus de traitement d'une requête utilisateur.

La figure 5.2 (p.77) présente les différentes classes nécessaires à l'exécution du CCO et qui sont présentées dans les sections suivantes. On peut noter l'absence d'une classe pour le CSO. En effet, les opérations réalisées par l'opérateur sont effectuées durant l'exécution des autres classes. Le calcul des fill-up étant réalisé en amont de la différence, quand une différence est nécessaire, les différentes classes peuvent directement effectuer le TSO des deux fill-up. L'ensemble des opérations nécessaires à l'exécution du CCO ont été implémentées, y compris la normalisation et la classification d'une TBox qui aurait pu être réalisées par des raisonneurs existants. En effet, nous souhaitions offrir à Perfect Memory un contrôle total sur les opérations ainsi que la réduction des interfaces entre la plateforme, notre implémentation et un potentiel raisonneur.

5.3 Implémentation de la normalisation d'une TBox $\mathcal{EL}$

La classe Normalisation effectue la normalisation d'une TBox $\mathcal{EL}$ $\mathcal{T}$ d'après Baader et al. (2005) dont on rappelle les règles dans le tableau 5.3 (p.78). La normalisation nécessite une TBox générale, avec possiblement des axiomes définitionnels, qu'il faudra transformer en deux GCI équivalents. Résulte de la normalisation une TBox normalisée $\mathcal{T}_{Bnf}$ ne contenant que des axiomes de la forme $A \sqsubseteq B$, $A \sqcap B \sqsubseteq B$, $A \sqsubseteq \exists r.B$ et $\exists r.A \sqsubseteq B$, avec A et B des noms de concept et r un nom de rôle.

La classe possède trois méthodes principales, qui transforment une TBox en une TBox normalisée selon Baader et al. (2005) avec ses restrictions existentielles renommées (voir figure 5.3 (p.78)) :

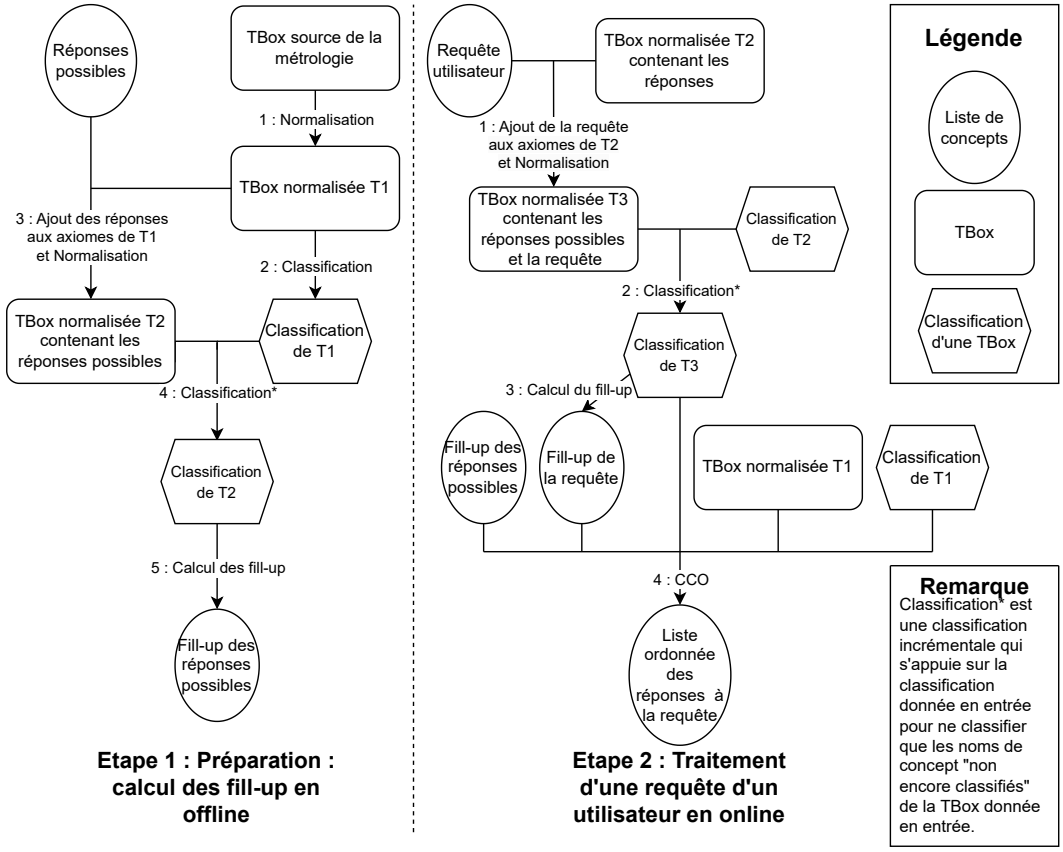


FIGURE 5.1 – Déroulement complet du processus du CCO.

1. preNorm() (algorithme 8 (p.127)) effectue un pré-traitement de la TBox, pour transformer les axiomes définitionnels $A \equiv C$ en deux axiomes équivalents $A \sqsubseteq C$ et $C \sqsubseteq A$. La fonction parcourt simplement la TBox et modifie les axiomes définitionnels.
2. norm() (algorithme 10 (p.128)) effectue la normalisation d'après Baader et al. (2005) à l'exception d'un cas. Le procédé implique l'ajout de nouveaux noms de concept, qui sont de la forme X_i, où i est un nombre qui n'a pas déjà été utilisé.Le cas non traité par la fonction est le cas où un axiome est de la forme $A \sqcap B \sqsubseteq \exists r.E$, où A, B et E sont des noms de concept, car la fonction qui suit, postNorm(), s'occupe de ce cas. Comme la fonction n'effectue pas complètement la normalisation selon Baader et al. (2005), on appellera la TBox résultante de norm() une TBox quasi-normalisée.
3. postNorm() (algorithme 9 (p.128)) effectue un traitement supplémentaire qui est nécessaire au calcul du fill-up. Elle modifie la TBox de manière à remplacer les restrictions existentielles par des noms de concept en ajoutant deux axiomes permettant de définir l'équivalence entre ces noms de concepts et la restriction exis-

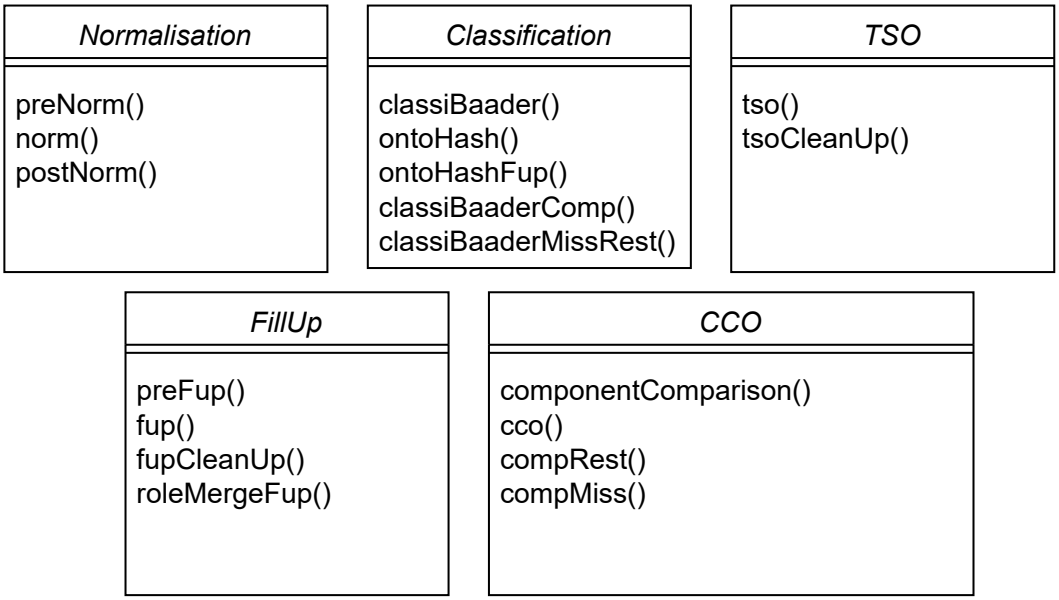


FIGURE 5.2 – Les cinq classes utilisées dans l'implémentation du CCO.

tentielle remplacée (voir les fonctions rename-input et rename-existential de la section 3.2.1). Ainsi, un axiome de type $A \sqsubseteq \exists v.B$ devient $A \sqsubseteq R\_v\_B$ et on ajoute les axiomes $\exists v.B \sqsubseteq R\_v\_B$ et $R\_v\_B \sqsubseteq \exists v.B$ à la TBox. Un axiome de type $\exists v.B \sqsubseteq A$ devient $R\_v\_B \sqsubseteq A$ et on ajoute les axiomes $\exists v.B \sqsubseteq R\_v\_B$ et $R\_v\_B \sqsubseteq \exists v.B$ à la TBox. Ce traitement permet également de traiter l'exception de la fonction norm() et finalise ainsi la normalisation selon Baader et al. (2005).

Le cas non traité dans norm() et traité dans postNorm() permet à la TBox finale de contenir moins de concepts et axiomes ajoutés. On évite ce cas en appliquant la règle NR3 dans le tableau 5.3 (p.78) dans tous les cas sauf celui où l'axiome est de la forme $A \sqcap B \sqsubseteq \exists r.E$, avec A et B des noms de concepts et E un concept (voir ligne 12 de l'algorithme norm() 10 (p.128)). La règle originale implique la création d'un nouveau nom de concept pour remplacer la partie droite de l'axiome (la restriction existentielle). Comme on propose de faire cette opération dans postNorm() lors du renommage de toutes les restrictions existentielles, on peut éviter de la faire dans norm(). On optimise ainsi le nombre d'axiomes et de noms de concept ajoutés dans le résultat final, comme l'exemple 28 (p.77) le montre.

Exemple 28. *Soit $\mathcal{T}$ une TBox $\mathcal{EL}$: $\{A \sqcap B \sqsubseteq \exists r.E\}$. Sa normalisation en appliquant strictement les règles de la table 5.3 (p.78) est :*
$\{A \sqcap B \sqsubseteq X_1, X_1 \sqsubseteq \exists r.E\}$
Le renommage des restrictions existentielles donne :
$\{A \sqcap B \sqsubseteq X_1, X_1 \sqsubseteq R\_r.E, R\_r.E \sqsubseteq \exists r.E, \exists r.E \sqsubseteq R\_r.E\}$
D'un autre côté sa normalisation en appliquant l'algorithme 10 (p.128) est :

NR1a	$C \sqcap \widehat{D} \sqsubseteq E$	$\rightarrow$	$\{\widehat{D} \sqsubseteq A, C \sqcap A \sqsubseteq E\}$
NR1b	$\widehat{C} \sqcap D \sqsubseteq E$	$\rightarrow$	$\{\widehat{C} \sqsubseteq A, A \sqcap D \sqsubseteq E\}$
NR2	$\exists r.\widehat{C} \sqsubseteq D$	$\rightarrow$	$\{\widehat{C} \sqsubseteq A, \exists r.A \sqsubseteq D\}$
NR3	$\widehat{C} \sqsubseteq \widehat{D}$	$\rightarrow$	$\{\widehat{C} \sqsubseteq A, A \sqsubseteq \widehat{D}\}$
NR4	$B \sqsubseteq \exists r.\widehat{C}$	$\rightarrow$	$\{B \sqsubseteq \exists r.A, A \sqsubseteq \widehat{C}\}$
NR5	$B \sqsubseteq C \sqcap D$	$\rightarrow$	$\{B \sqsubseteq C, B \sqsubseteq D\}$

TABLE 5.3 – Règles de normalisation d'après Baader et al. (2005) adaptées à $\mathcal{EL}$. $\widehat{C}$ et $\widehat{D}$ sont des concepts qui ne sont pas des noms de concept, r un rôle, C, D et E des concepts, B un nom de concept et A un nouveau nom de concept. On applique NR1a, NR1b et NR2 de manière exhaustive, puis NR3 à NR5 de manière exhaustive.

$\{A \sqcap B \sqsubseteq \exists r.E\}$
Le renommage des restrictions existentielles donne :
$\{A \sqcap B \sqsubseteq R\_r.E, R\_r.E \sqsubseteq \exists r.E, \exists r.E \sqsubseteq R\_r.E\}$
On constate que l'algorithme produit un axiome de moins ($X_1 \sqsubseteq R\_r.E$) et utilise un concept de moins (X_1).

L'algorithme 9 (en annexe 10 (p.128)) applique les règles de normalisation axiome par axiome plutôt que de considérer la TBox dans son ensemble (comme suggéré dans Baader et al. (2005)). En effet, chaque axiome est indépendant des autres et la modification de l'un d'eux n'affecte pas les autres. On évite ainsi de parcourir la TBox plus que nécessaire. Les règles du tableau 5.3 (p.78) nécessitent normalement d'appliquer d'abord les règles CR1 et CR2 de manière exhaustives, puis les autres règles, pour éviter une explosion quadratique de taille liée à la duplication du concept B dans la règle NR5 (voir Baader et al. (2005)). Dans notre implémentation, la structure conditionnelle de l'algorithme conserve cet ordre d'application et lorsque la règle NR5 s'applique (la ligne 15 de l'algorithme 9 (annexe 10 (p.128))), B est forcement un nom de concept ne nécessitant plus de transformation de ce côté de la subsomption. Les résultats expérimentaux obtenus confirment que l'on n'a pas cette explosion quadratique (voir test 14 en annexe B.4.1 (p.144)).

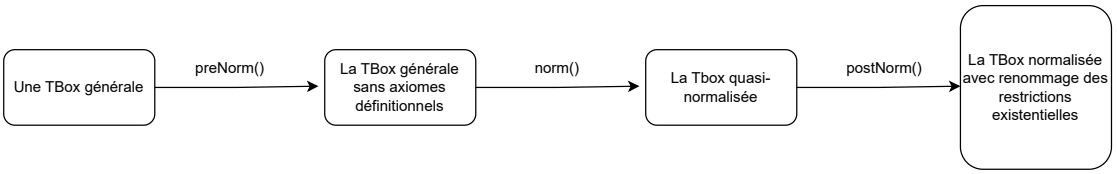


FIGURE 5.3 – Workflow de la classe Normalisation.

Un exemple complet d'exécution plus complexe que l'exemple 28 (p.77) est disponible dans l'annexe B.3.2 (p.139).

5.4 Implémentation de la classification d'une TBox $\mathcal{EL}$

La classification est le procédé consistant à expliciter chaque relation de subsomption entre tous les couples possibles de noms de concept d'une TBox (voir Baader et al. (2005, 2017)). Ce procédé n'explicite cependant pas les relations de subsomption impliquant des restrictions existentielles. On rappelle que le fill-up d'un concept nécessitant l'explicitation de toutes les branches qui le subsument, nous avons besoin des relations de subsomption impliquant des restrictions existentielles. L'étape de normalisation de postNorm(), qui transforme chaque restriction existentielle en un nom de concept dans tous les axiomes de la TBox permet de les obtenir. Ainsi après postNorm() et avant la classification, les seuls axiomes contenant des restrictions existentielles sont ceux qui permettent de les nommer et sont de la forme $R\_r.B \sqsubseteq \exists r.B$ ou $\exists r.B \sqsubseteq R\_r.B$ où r est un rôle de la TBox et B et $R\_r.B$ sont des noms de concept. Pour rappel, dans l'implémentation, une classification est représentée sous la forme d'un table de hachage (voir l'exemple 27 (p.75)).

La classe Classification possède cinq fonctions qui réalisent deux types d'opérations : le traitement de la table de hachage de la classification et la classification elle-même. Les différentes fonctions réalisent l'une ou l'autre de ces opérations en fonction des entrées. Les deux types d'opérations s'enchainent comme présenté dans la figure 5.4 (p.80), en utilisant des fonctions différentes selon le cas :

1. ontoHash() permet de préparer la table de hachage qui va accueillir la classification de la TBox. La fonction parcourt la TBox et insère une clé dans la table de hachage pour chaque nom de concept. La fonction n'ajoute pas $\top$ dans chaque clé car il est implicitement compris dans celle-ci. On réduit alors la taille du contenu de chaque clé sans perte d'information. Cette fonction est utilisée lors de la classification d'une TBox dont on n'a jamais réalisé sa classification.
2. classiBaader() effectue la classification d'une TBox normalisée, appliquant les règles de Baader et al. (2005) présentées dans la table 5.4 (p.81). Cette fonction est utilisée lors de la classification de la TBox initiale et pour la classification servant à déterminer le fill-up d'un concept. Si la TBox ne contient aucun axiome décrivant des réponses, la fonction produit ce qu'on appelle la classification initiale.
3. ontoHashExistant() effectue le même type de traitement que ontoHash(). La fonction est utilisée dans le cas où la TBox en entrée a déjà été en partie classifiée, en préparation de la classification servant à déterminer le fill-up d'un concept. La fonction reprend la table de hachage existante correspondante et complète celle-ci.
4. classiBaaderComp() effectue le même type de traitement que classiBaader(). Cette fonction est utilisée quand on a une classification existante et deux axiomes définissant C et D dont on veut connaître la relation de subsomption. Cette fonction est utilisée uniquement lors des tests qualitatifs (section 6.1 (p.89)).
5. classiBaaderMissRest() effectue le même type de traitement que classiBaaderComp(), à partir de la classification initiale. Elle est utilisée pour déterminer une

relation de subsomption entre deux rest ou deux miss lors d'une exécution du CCO.

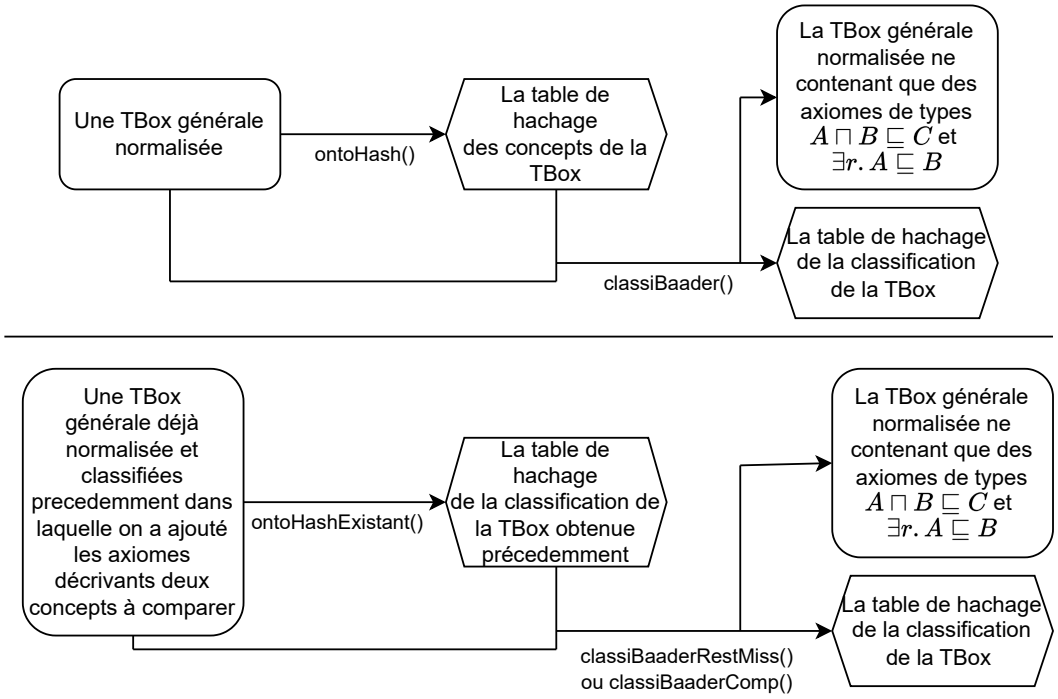


FIGURE 5.4 – Workflow de la classe Classification. Pour déterminer la relation de subsomption entre deux concepts à partir d'une classification déjà existante, on utilisera classiBaaderMissRest() ou ClassiBaaderComp() et ontoHashExistant().

Dans notre implémentation nous proposons une optimisation dans l'application des règles de classification. Parcourir la TBox (un tableau) étant plus coûteux que de parcourir la classification (une table de hachage, à l'aide des clés de celle-ci), nous proposons de travailler sur la TBox normalisée et de supprimer les axiomes "inutiles" de celle-ci. Ainsi, si un axiome est utilisé une première fois pour appliquer la règle CR1 ou CR3, l'information qu'il portait se trouve après application de CR1 ou CR3 dans la table de hachage. On propose donc d'enlever l'axiome de la TBox pour alléger les futurs parcours de celle-ci parce que la même information se trouve dans la table de hachage. En d'autres termes, pour vérifier si $A \sqsubseteq B$, plutôt que de parcourir le tableau représentant TBox à la recherche de l'axiome correspondant, il suffit de regarder si B est dans la table de hachage à la clé A (La classification de A). Cela impose de transformer les règles de classification de la table 5.4 (p.81) en remplaçant CR1 par CR1a et CR1b, et CR3 par CR3a et CR3b (voir table 5.5 (p.81)). Notons que cette optimisation n'a pas été testée par benchmark et n'a été testée qu'au travers des tests de la section 6.2 (p.94).

À l'initialisation de la classification, les $\mathbf{S}(C)$ de chaque concept ne contiennent que le concept lui-même : $\mathbf{S}(C) = C$. Ainsi, on constate qu'à l'état initial, seules les règles

CR1	Si $C' \in \mathbf{S}(C)$, $C' \sqsubseteq D \in \mathcal{T}$, et $D \notin \mathbf{S}(C)$, alors $\mathbf{S}(C) := \mathbf{S}(C) \cup D$
CR2	Si $C_1, C_2 \in \mathbf{S}(C)$, $C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{T}$, et $D \notin \mathbf{S}(C)$, alors $S(C) := \mathbf{S}(C) \cup D$
CR3	Si $C' \in \mathbf{S}(C)$, $C' \sqsubseteq \exists r.D \in \mathcal{T}$, et $(C, D) \notin \mathbf{R}(r)$, alors $\mathbf{R}(r) := \mathbf{R}(r) \cup (C, D)$
CR4	Si $(C, D) \in \mathbf{R}(r)$, $D' \in \mathbf{S}(D), \exists r.D' \sqsubseteq E \in \mathcal{T}$, et $E \notin \mathbf{S}(C)$, alors $\mathbf{S}(C) := \mathbf{S}(C) \cup E$

TABLE 5.4 – Règles de classification d'après Baader et al. (2005) avec $\mathbf{S}(C)$ l'ensemble des noms de concepts subsumant C et $\mathbf{R}(r)$ un ensemble de couples (C, D) tels que $C \sqsubseteq \exists r.D$, avec r un rôle de la TBox et C, D des noms de concept de la TBox.

CR1a	Si $C' \in \mathbf{S}(C)$, $C' \sqsubseteq D \in \mathcal{T}$, et $D \notin \mathbf{S}(C)$, alors $\mathbf{S}(C) := \mathbf{S}(C) \cup D$ et $\mathcal{T} \setminus C' \sqsubseteq D$
CR1b	Si $C' \in \mathbf{S}(C)$, $D \in \mathbf{S}(C')$, et $D \notin \mathbf{S}(C)$, alors $\mathbf{S}(C) := \mathbf{S}(C) \cup D$
CR2	Si $C_1, C_2 \in \mathbf{S}(C)$, $C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{T}$, et $D \notin \mathbf{S}(C)$, alors $\mathbf{S}(C) := \mathbf{S}(C) \cup D$
CR3a	Si $C' \in \mathbf{S}(C)$, $C' \sqsubseteq \exists r.D \in \mathcal{T}$, et $(C, D) \notin \mathbf{R}(r)$, alors $\mathbf{R}(r) := RrC) \cup (C, D)$ et $\mathcal{T} \setminus C' \sqsubseteq \exists r.D$
CR3b	Si $C' \in \mathbf{S}(C)$, $(C', D) \in \mathbf{R}(r)$, et $(C, D) \notin \mathbf{R}(r)$, alors $\mathbf{R}(r) := \mathbf{R}(r) \cup (C, D)$
CR4	Si $(C, D) \in \mathbf{R}(r)$, $D' \in \mathbf{S}(D), \exists r.D' \sqsubseteq E \in \mathcal{T}$, et $E \notin \mathbf{S}(C)$, alors $\mathbf{S}(C) := \mathbf{S}(C) \cup E$

TABLE 5.5 – Règles de classifications permettant de profiter de l'optimisation utilisant les tables de hachage adaptées de Baader et al. (2005).

CR1a et CR3a sont en mesure d'être appliquées. Un premier parcours de la TBox est alors réalisé pour appliquer l'ensemble des règles CR1a et CR3a possibles, et l'axiome utilisé est supprimé de la TBox. Après l'ensemble des applications possibles, il ne reste alors dans la TBox que des axiomes de types $C_1 \sqcap C_2 \sqsubseteq D$ et $\exists r.C \sqsubseteq D$, types nécessaires à l'application des autres règles. Notons que la règle CR3b n'a pas non plus besoin d'être appliquée grâce à la normalisation supplémentaire de postNorm(). La démonstration associée est disponible dans l'annexe B.2.1 (p.135).

Nous proposons maintenant une deuxième optimisation sur la constatation suivante : il n'est pas nécessaire de parcourir l'ensemble des mappings $\mathbf{S}(C)$ et $\mathbf{R}(r)$ à chaque itération de l'algorithme des règles. Si un de ces ensembles n'a pas subi de modification à l'itération précédente, il ne pourra pas satisfaire les conditions des règles de classification à l'itération courante. Les démonstrations associées sont disponibles dans l'annexe B.2.2 (p.135).

Ainsi, en mémorisant quels ensembles $\mathbf{S}(C)$ et $\mathbf{R}(r)$ ont été modifiés lors d'une itération d'application des règles, on peut réduire le nombre de tests à effectuer lors de l'itération suivante. On utilise pour cela un tableau qui contient les concepts et rôles dont le mapping a été modifié à l'itération précédente. Pour faciliter l'approche, on a ajouté un mapping inverse, **S***. Le premier mapping $\mathbf{S}(A)$ contient les noms de concept qui subsument A, et le second mapping $\mathbf{S^*}(A)$ contient les noms de concepts subsumés

par A. Ainsi, si on ajoute un concept B dans $\mathbf{S}(A)$, il suffit de parcourir $\mathbf{S^*}(A)$ pour connaître les concepts dont le mapping $\mathbf{S}$ doit être complété avec le concept B.

Enfin, $\top$ étant présent dans l'ensemble des ensembles $\mathbf{S}(C)$, nous n'avons pas besoin de le stocker dans la classification, sa présence étant implicite.

Un exemple d'une exécution de la classification se trouve en annexe B.3.3 (p.140), l'algorithme 11 ontoHash() et l'algorithme 12 de classification classiBaader() sont disponibles dans l'annexe B.1.2 (p.129).

5.5 Implémentation du fill-up

La classe FillUp possède trois fonctions principales, qui s'enchaînent comme présenté dans la figure 5.5 (p.82) :

1. preFup() s'occupe de renommer les restrictions existentielles du concept C dont on veut calculer le fill-up, de la même manière que postNorm(). La fonction effectue ensuite la classification de la TBox utilisée à laquelle on aura rajouté les axiomes $F\_y \sqsubseteq C$ et $C \sqsubseteq F\_y$ (où y est un nombre et F_y est un nouveau nom de concept). y est un nombre car on va ajouter plusieurs fois de nouveaux noms de concept au cours du processus.
2. fup() (algorithme 13 dans l'annexe B.1.3 (p.131)) effectue le fill-up du concept C à partir de la TBox classifiée.
3. fupCleanUp() retire du fill-up de C tous les concepts qui ont été ajoutés pour des besoins de normalisation, de création de nouveaux noms de concept et de renommage, afin de ne plus contenir que des concepts présents dans la TBox initiale.

Un exemple d'application du workflow de la figure 5.5 est donné dans l'exemple 43 de l'annexe B.3.4 (p.143).

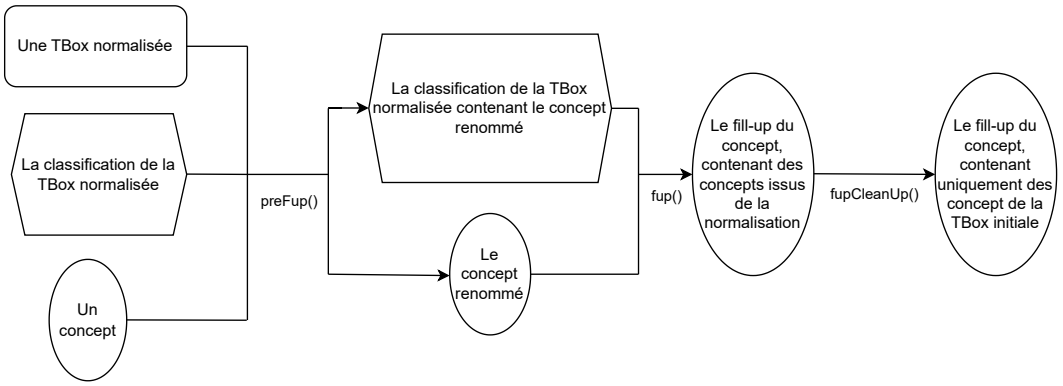


FIGURE 5.5 – Workflow de la classe FillUp. Ici, preFup() réalise le travail de classification* dans la figure 5.1 (p.76).

5.6 Implémentation du TSO

Le TSO (voir section 3.1.2 (p.49)) réalise la différence syntaxique de deux concepts C et D. On parcourt en parallèle les structures arborescentes de C et de D pour comparer deux à deux leurs branches. On retire les branches de br_C qui sont aussi des branches de br_D, mais également les branches de br_C qui se terminent par $\top$ quand il existe une branche dans br_D qui commence par les mêmes restrictions existentielles. Si toutes les branches de C sont retirées, alors le résultat est $\top$. La classe possède deux fonctions, dont l'enchaînement est présenté dans la figure 5.6 (p.83).

1. tso() (algorithme 14 en annexe B.1.3 (p.131)) réalise la différence syntaxique.
2. tsoCleanUp() est une fonction purement liée à l'implémentation qui s'assure que le concept final est bien formé (il n'y a pas de "&" en trop).

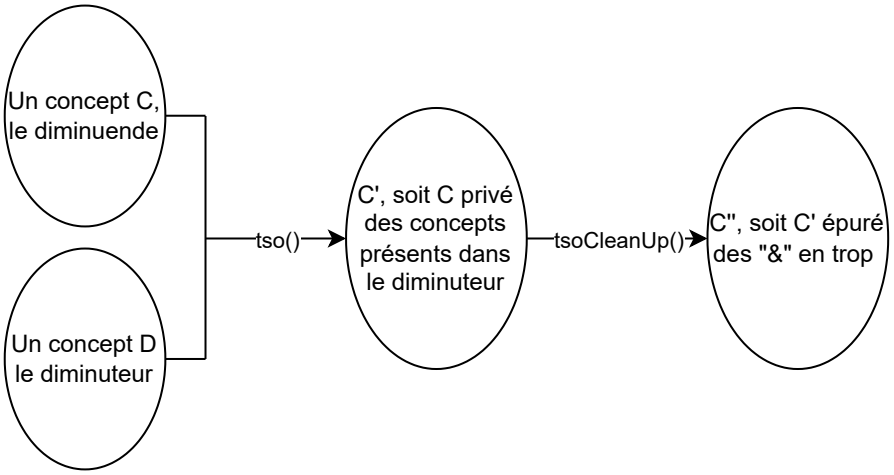


FIGURE 5.6 – Workflow du calcul tso(C, D) de la classe TSO.

On donne maintenant plusieurs exemple d'exécution de l'algorithme 14 (en annexe B.1.3 (p.131))) dans l'exemple 29 suivant :

Exemple 29. *Avec C=A et D=B, tso(C, D)=A*
Avec C=A et D=A, tso(C, D)=$\top$
Avec C=$A \sqcap B$ et D=A, tso(C, D)=B
Avec C=$A \sqcap B$ et D=$A \sqcap E$, tso(C, D)=B
Avec C=$A \sqcap B$ et D=$A \sqcap B$, tso(C, D)=$\top$
Avec C=$\exists r.A$ et D=A, tso(C, D)=$\exists r.A$
Avec C=$\exists r.A$ et D=$\exists s.A$, tso(C, D)=$\exists r.A$
Avec C=$\exists r.A$ et D=$\exists r.A$, tso(C, D)=$\top$
Avec C=$\exists r.(A \sqcap B)$ et D=$\exists r.A$, tso(C, D)=$\exists r.B$

5.7 Implémentation du CCO

Le CCO (voir section 2 (p.35)) effectue la comparaison des réponses potentielles à une demande d'un utilisateur. Grâce aux relations de subsomption entre la demande et les réponses, on peut effectuer un premier classement des réponses : si une réponse est subsumée par la demande, elle est meilleure qu'une qui subsume la demande ou qui n'a pas de relation de subsomption avec celle-ci. Un second classement est réalisé ensuite au sein d'un même rang de ce classement, en comparant les relations de subsomption des rest de deux réponses deux à deux, puis des miss, et si nécessaire leur taille. Les résultats du classement par composante sont ensuite agrégés en un score qui permet la détermination de l'ordre total multicritère.

La classe possède six fonctions principales. La figure 5.7 (p.85) présente l'enchaînement des deux plus importantes et 5.8 (p.86) le processus de cco().

- componentComparison() utilise les calculs de classification déjà réalisés pour déterminer les relations de subsomption entre les réponses et la demande, pour chaque composante.
- cco() est la fonction qui réalise la comparaison multicritère des réponses par rapport à la demande. En fonction du classement obtenu avec componentComparison(), la fonction effectue les comparaisons de toutes les réponses deux à deux pour chaque composante, en utilisant compRest(), compMiss() et compLongueur() lorsque c'est nécessaire.
- compRest() effectue la comparaison des rest de deux réponses par rapport à une demande, on calcule les deux rest puis leur relation de subsomption en utilisant comparaisonComposanteMissRest().
- compMiss() effectue la comparaison des miss de deux réponses par rapport à une demande. On effectue le même traitement que compRest mais avec les miss.
- compTaille() effectue la comparaison des tailles de deux miss ou de deux rest, si leur relation de subsomption n'était pas suffisante pour les départager. Il s'agit d'une simple comparaison de tailles de tableau.
- comparaisonComposanteMissRest() est une fonction qui sert à calculer les relations de subsomption entre deux rest ou deux miss à l'aide d'une classification effectuée à partir de la TBox et la classification initiale.

Pour chaque composante, on commence par calculer les relations de subsomption entre la demande et les réponses en utilisant la fonction componentComparison() (algorithme 15 (p.133)). Avec celles-ci on peut utiliser la fonction cco() qui est l'enchaînement de l'algorithme 2 (p.42) (pour le calcul des scores) et l'algorithme 1 (pour l'agrégation des scores) de la section du CCO. Dans la fonction cco(), en fonction des relations de subsomption, on commence par calculer et comparer soit des miss puis leur taille soit des rest puis des miss puis la taille des rest puis la taille des miss. On utilise compRest()(algorithme 16 (p.134)) (resp. compMiss()) pour calculer les rest (resp. les miss). compRest() et compMiss() utilisent toutes deux comparaisonComposanteMissRest() (algorithme 17 (p.134)) pour calculer les relations de subsomption si nécessaire. Enfin, compTaille(), qui est un simple calcul de longueur de tableau, permet de départager si

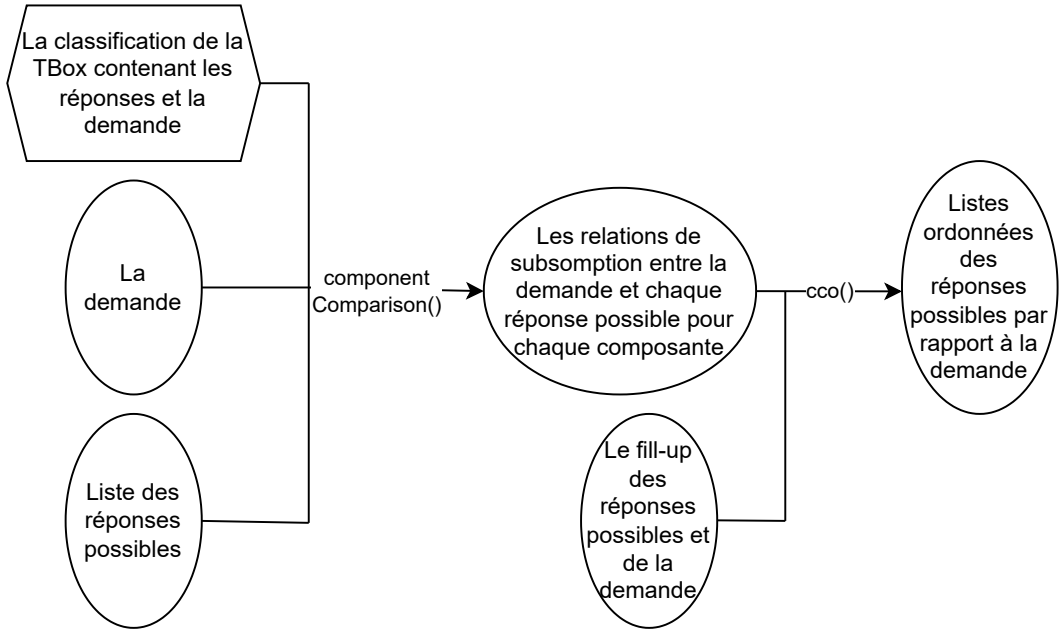


FIGURE 5.7 – Workflow de la classe CCO.

les calculs de subsomption précédents ne sont pas suffisants. Enfin, la fonction cco() se termine en agrégeant les scores de chaque composante pour trier le tableau des réponses.

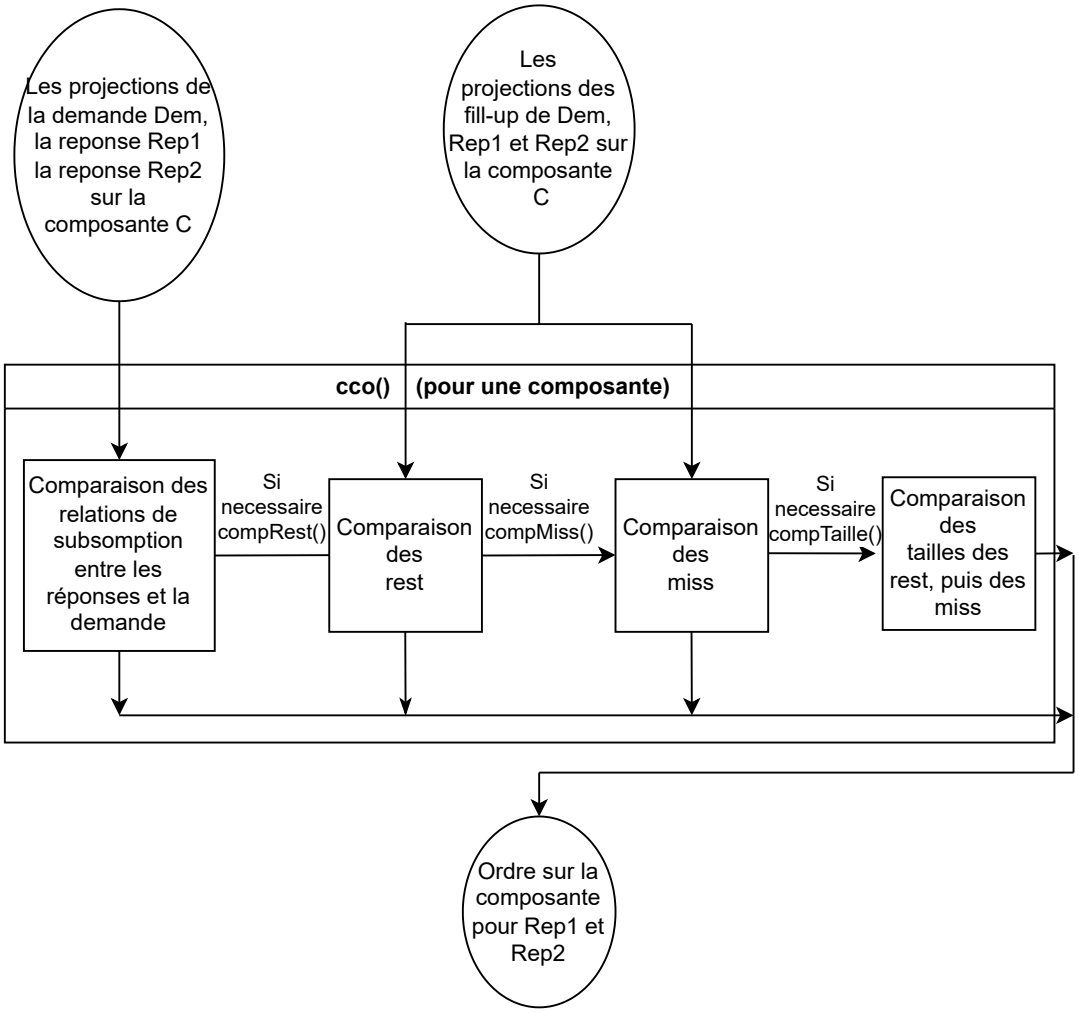


FIGURE 5.8 – Workflow de la fonction cco().

5.8 Execution

Le programme principal qui réalise l'ensemble du flux du CCO suit celui de la figure 5.1 (p.76). Pour la préparation (étape 1 de la figure), c'est-à-dire toutes les opérations qui peuvent être effectuées avant de recevoir une demande d'un utilisateur, on calcule dans cet ordre :

— la mise en hachage des réponses,
— 1 : la normalisation de la TBox,
— 2 : la classification de la TBox. Cette classification et la TBox associées sont stockées en temps que classification et normalisation initiales. On clone la TBox initiale pour qu'elle puisse être utilisée dans les opérations suivantes,
— Pour chaque réponse possible et chaque composante :

  - 3 : On ajoute la projection de la réponse sur la composante dans la TBox déjà normalisée,
  - 4 : On classifie la TBox déjà en partie classifiée,
  - 5 : Le fill-up de la réponse pour la composante est stocké dans une table de hachage

À la réception de la demande (étape 2 de la figure), on réalise calcul dans cet ordre :

- Pour chaque composante :
  - 1 : On ajoute la projection de la demande sur la composante dans la TBox déjà normalisée, qui a servi pour calculer les fill-up des réponses possibles,
  - 2 : On classifie la TBox déjà en partie classifiée et on obtient les relations de subsomption entre la demande et les réponses pour chaque composante
- 3 : On extrait le fill-up de la demande pour chaque composante.
- 4 : le CCO de la demande, à partir des relations de subsomption obtenues à l'opération 2 de l'étape 2 et les fill-up des réponses et de la demande. On utilise la TBox et la classification initiale pour la comparaison des rest et des miss quand c'est nécessaire.

Chapitre 6

Tests de l'implémentation

Pour observer le comportement de l'implémentation, nous avons réalisé des tests s'assurant de la qualité des traitements effectués. Des tests qualitatifs pour chaque étape du CCO sont présentés dans la section 6.1 (p.89) et des tests quantitatifs sont présentés dans la section 6.2 (p.94). Pour la réalisation des tests, nous avons fait le choix de ne pas utiliser la plateforme Perfect Memory car nous n'aurions pas pu tester tous les cas possibles. En effet l'ontologie de Perfect Memory ne couvre pas tous les cas de figure. Les tests de cette section ont tous été réalisés avec ordinateur portable ayant un IntelCore i7-8650U de 1.90 GHz et 16 Go de RAM.

6.1 Tests qualitatifs

Pour les tests qualitatifs, nous avons fait le choix de réaliser tout d'abord des tests unitaires, puis de combiner des cas unitaires pour obtenir des cas plus complexes. Les TBox restent cependant de petite taille pour pouvoir facilement vérifier les résultats et sont le plus souvent déjà sous forme normale pour éviter l'introduction de trop de concept supplémentaire réduisant la clarté des résultats. Pour les tests impliquant des réponses, leur nombre et leur taille sont également réduits pour une vérification plus facile. Chaque étape de l'algorithme CCO est testée, on a donc la normalisation dans la section 6.1.1 (p.89), la classification dans la section 6.1.2 (p.90), le fill-up dans la section 6.1.3 (p.90), le TSO dans la section 6.1.4 (p.91) les calculs de subsomption dans la section 6.1.5 (p.92), la determination du rest et du miss dans la section 6.1.6 (p.92), des calculs de CCO avec une seule et plusieurs composantes dans les sections 6.1.7 (p.93) et 6.1.8 (p.93). Enfin un dernier test s'assure que le cas cyclique n'est pas défini dans la section 6.1.9 (p.94).

Tous les résultats des tests qualitatifs se trouvent dans l'annexe B.4 (p.143).

6.1.1 Normalisation

Les différents tests évaluent les différentes règles de normalisation individuellement, puis plusieurs en même temps, jusqu'à un cas demandant l'application de toutes les

Numéro de test	Différence par rapport aux résultats attendus
1 à 4	Non
5 à 14	Les différences proviennent d'un ordre d'exécution des règles qui n'est pas le même. Cet ordre peut avoir un impact sur l'ordre d'apparition des concepts de type X_i. Par ailleurs la règle de renommage appliquée (restriction existentielle remplacée par un nom de concept) pour permettre le calcul du fill-up n'existe pas dans la normalisation originale et introduit des différences. Cette étape de renommage ne modifiant pas la sémantique de la TBox obtenue, on a bien une TBox équivalente au résultat attendu.

TABLE 6.1 – Synthèse des résultats des tests qualitatifs de la normalisation obtenus avec les algorithmes de normalisation présentés dans la section 5.3 (p.75) (algorithmes de l'annexe B.1.1 (p.127)) par rapport aux résultats attendus par application des règles de la table 5.3 (p.78).

règles. Les résultats sont synthétisés dans la table 6.1 (p.90). Les tests se trouvent à l'annexe B.4.1 (p.144). Les règles de normalisation sont celles de la table 5.3 (p.78). L'exemple 30 (p.90) montre un de ces tests.

Exemple 30. *Test 5 :*
TBox : $\{A \sqcap \exists r.B \sqsubseteq C\}$
Normalisation attendue : $\{A \sqcap X_1 \sqsubseteq C, \exists r.B \sqsubseteq X_1\}$
Normalisation obtenue : $\{A \sqcap X_1 \sqsubseteq C, R\_r.B \sqsubseteq X_1, \exists r.B \sqsubseteq R\_r.B, R\_r.B \sqsubseteq \exists r.B\}$

6.1.2 Classification

Les différents tests évaluent les différentes règles de classification individuellement, puis plusieurs en même temps dans des ordres différents, jusqu'à un cas demandant l'application de toutes les règles. Les résultats sont synthétisés dans la table 6.2 (p.91). Les tests se trouvent à l'annexe B.4.2 (p.145). Les règles de normalisation sont celles de la table 5.3 (p.78). L'exemple 31 (p.90) montre un de ces tests.

Exemple 31. *Test 19 :*
TBox : $\{\exists x.Y \sqsubseteq A\}$
Classification attendue : $S(A) = \{A\}, S(Y) = \{Y\}$
Classification obtenue : $S(A) = \{A\}, S(Y) = \{Y\}, S(R\_x.Y) = \{R\_x.Y, A\}, R(x) = \{R\_x.Y, Y\}$

6.1.3 Fill-up

Les différents tests évaluent les cas particuliers (comme $\top$) et unitaires, puis à tester des cas plus complexes impliquant plusieurs appels récursifs de la fonction. On rappelle

Numéro de test	Différence par rapport aux résultats attendus
15 à 17 et 20	Non
18 à 19 et 21 à 25	Les différences s'expliquent par une TBox normalisée potentiellement différente (voir table 6.1 (p.90)). Néanmoins, on peut facilement constater que les liens de subsomptions entre les noms de concept initiaux de la TBox sont les mêmes qu'on les obtienne avec l'algorithme ou les règles originales de la table 5.4.

TABLE 6.2 – Synthèse des résultats des tests qualitatifs de la classification obtenus avec les algorithme de classification présentés dans la section 5.4 (p.79) (algorithmes de l'annexe B.1.2 (p.129)) par rapport aux résultats attendus par application des règles de la table 5.4 (p.81).

qu'on omet $\top$ dans les résultats pour plus de clarté. Les résultats sont synthétisés dans la table 6.3 (p.91). Les tests se trouvent à l'annexe B.4.3 (p.146). L'exemple 32 (p.91) montre un de ces tests.

Exemple 32. *Test 30 :*
TBox : $\{A \sqsubseteq B, C \sqsubseteq D\}$
Concept : $\exists r.A \sqcap C$
Fill-up attendu : $\exists r.(A \sqcap B) \sqcap C \sqcap D$
Fill-up obtenu : $\exists r.(A \sqcap B) \sqcap C \sqcap D$

Numéro de test	Différence
26 à 33	Non

TABLE 6.3 – Synthèse des résultats des tests qualitatifs du fill-up obtenus avec les algorithmes présentés dans la section 5.5 (p.82) (algorithmes de l'annexe B.1.3 (p.131)) par rapport aux résultats attendus par application manuelle.

6.1.4 TSO

Les différents tests évaluent les cas particuliers (comme $\top$) et unitaires, puis à tester des cas plus complexes impliquant plusieurs appels récursifs de la fonction. Les résultats sont synthétisés dans la table 6.4 (p.92). Les tests se trouvent à l'annexe B.4.4 (p.147). L'exemple 33 (p.91) montre un de ces tests.

Exemple 33. *Test 45 :*
Diminuende : $\exists r.(A \sqcap B) \sqcap C \sqcap D$
Diminiteur : $\exists r.A \sqcap C \sqcap E$
TSO attendu : $\exists r.B \sqcap D$
TSO obtenu : $\exists r.B \sqcap D$

Numéro de test	Différence
34 à 50	Non

TABLE 6.4 – Synthèse des résultats des tests qualitatifs du tso obtenus avec les algorithmes présentés dans la section 5.6 (p.83) (algorithme de l'annexe B.1.4 (p.132)) par rapport aux résultats attendus par application manuelle.

6.1.5 Calcul de subsomption

Les différents tests évaluent les cas unitaires. Les tests sont cependant très proches de ceux de la classification car le calcul de subsomption se fait par l'intermédiaire d'une classification. Les résultats sont synthétisés dans la table 6.5 (p.92). Les tests se trouvent à l'annexe B.4.5 (p.149). L'exemple 34 (p.92) montre un de ces tests.

Exemple 34. *Test 56 :*
TBox : $\{A \sqsubseteq \exists r.B\}$
Demande : $\exists r.(B \sqcap C)$
Réponse : $\exists r.B \sqcap \exists r.C$
Subsomption attendue : demande subsumée
Subsomption obtenue : demande subsumée

Numéro de test	Différence
51 à 56	Non

TABLE 6.5 – Synthèse des résultats des tests qualitatifs de subsomption obtenus avec la fonction componentComparison() (algorithme 15 (p.133)) présentée dans la section 5.7 (p.84) par rapport aux résultats attendus par application manuelle de la classification.

6.1.6 Rest et miss

Les différents tests évaluent les cas particuliers (comme $\top$) et unitaires, de manière similaire aux tests du tso dans la section 6.1.4 (p.91), puis à tester des cas plus complexes avec des TBox plus fournies. Les résultats sont synthétisés dans la table 6.6 (p.93). Les tests se trouvent à l'annexe B.4.6 (p.150). L'exemple 35 (p.92) montre un de ces tests.

Exemple 35. *Test 61 :*
TBox : $\{A \sqsubseteq B, B \sqsubseteq \exists r.C\}$
Demande : $\exists r.C$
Réponse : A
Rest attendu : $\top$
Rest obtenu : $\top$
Miss attendu : $A \sqcap B$
Miss obtenu : $A \sqcap B$

Numéro de test	Différence
57 à 66	Non

TABLE 6.6 – Synthèse des résultats des tests qualitatifs de détermination du rest et du miss obtenus en calculant le fill-up de la demande et la réponse et en effectuant le tso avec les fill-up obtenus par rapport aux résultats attendus par application manuelle du fill-up et du TSO.

6.1.7 CCO avec une seule composante

Les différents tests évaluent les cas unitaires, puis à tester des cas plus complexes avec plusieurs réponses et des TBox plus complexes, avec une seule composante. Les résultats sont synthétisés dans la table 6.7 (p.93). Les tests se trouvent à l'annexe B.4.7 (p.152). L'exemple 36 (p.93) montre un de ces tests.

Exemple 36. *Test 70 :*
TBox : $\{A \sqsubseteq B, C \sqsubseteq A, B \sqsubseteq D\}$
Demande : B
Réponses : r0 : A*, r1 :* D
Classement attendu : (r0,1), (r1, -1)
Classement obtenu : (r0,1), (r1, -1)

Numéro de test	Différence
67 à 75	Non

TABLE 6.7 – Synthèse des résultats des tests qualitatifs du cco obtenus avec les algorithmes présentés dans la section 5.7 (p.84) (algorithme B.1.5 (p.133)) par rapport aux résultats attendus par application manuelle.

6.1.8 CCO avec plusieurs composantes

Les différents tests évaluent les cas unitaires, puis à tester des cas plus complexes avec plusieurs réponses et des TBox plus complexes, avec plusieurs composantes. Les résultats sont synthétisés dans la table 6.8 (p.94). Les tests se trouvent à l'annexe B.4.8 (p.153). L'exemple 37 (p.93) montre un de ces tests.

Exemple 37. *Test 77 :*
TBox : $\{A2 \sqsubseteq A1, B2 \sqsubseteq B1\}$
Demande : $\exists a.A1 \sqcap \exists b.B1 \sqcap \exists c.\top$
Réponses : r0 : $\exists a.A1 \sqcap \exists b.B1 \sqcap \exists c.C2$*, r1 :* $\exists a.A1 \sqcap \exists b.B1 \sqcap \exists c.\top$*, r2 :* $\exists a.A2 \sqcap \exists b.B2 \sqcap \exists c.\top$*, r3 :* $\exists a.A1 \sqcap \exists b.B2 \sqcap \exists c.\top$*, r4 :* $\exists a.A2 \sqcap \exists b.B1 \sqcap \exists c.\top$*, r5 :* $\exists a.(A2 \sqcap A1) \sqcap \exists b.B1 \sqcap \exists c.\top$*, r6 :* $\exists a.A3 \sqcap \exists b.B1 \sqcap \exists c.\top$

Classement attendu pour a : (r0,4), (r1, 4), (r3, 4), (r5, -2), (r4, -2), (r2, -2), (r6, -6)
Classement obtenu pour a : (r0,4), (r1, 4), (r3, 4), (r5, -2), (r4, -2), (r2, -2), (r6, -6)
Classement attendu pour b : (r0,2), (r6, 2), (r5, 2), (r4, 2), (r1, 2), (r3, -5), (r2, -5)
Classement obtenu pour b : (r0,2), (r6, 2), (r5, 2), (r4, 2), (r1, 2), (r3, -5), (r2, -5)
Classement attendu pour c : (r1,1), (r3, 1), (r2, 1), (r4, 1), (r5, 1), (r6, 1), (r0, -6)
Classement obtenu pour c : (r1,1), (r3, 1), (r2, 1), (r4, 1), (r5, 1), (r6, 1), (r0, -6)
Classement attendu : (r1,7), (r4, 1), (r5, 1), (r0, 0), (r3, 0), (r6, -3), (r2, -6)
Classement obtenu : (r1,7), (r4, 1), (r5, 1), (r0, 0), (r3, 0), (r6, -3), (r2, -6)

Numéro de test	Différence
76 et 77	Non. Remarque : Le test 77 met en exergue une caractéristique de la prise en compte des informations superflues (le miss) dans le classement. L'algorithme actuel prend en compte les composantes inexistantes de la demande lors du classement. Ainsi, si un utilisateur ne renseigne pas, par exemple, les matériaux de l'instrument qu'il recherche, l'algorithme discriminera les réponses potentielles dont les matériaux sont précisément renseignés.

TABLE 6.8 – Synthèse des résultats des tests qualitatifs du cco obtenus avec les algorithmes présentés dans la section 5.7 (p.84) (algorithmes de l'annexe B.1.5 (p.133)) par rapport aux résultats attendus par application manuelle.

6.1.9 TBox cyclique

Un test de calcul du fill-up est effectué avec pour TBox $\mathcal{T} = \{D \sqsubseteq \exists r.D\ \}$ et une demande Dem $= \exists r.D$ Cette TBox comportant un cycle, le calcul du fill-up boucle à l'infini jusqu'à atteindre la limite de pile de l'exécutable. Ce test sert uniquement à vérifier le comportement de l'algorithme en présence de cycle.

6.2 Tests quantitatifs

Pour tester les performances de l'implémentation proposées dans la section 5 (p.73), nous avons généré aléatoirement plusieurs jeux de TBox et réponses correspondantes en fonction des facteurs identifiés comme influents sur le temps de traitement de la demande. Ces facteurs sont :

- La taille de la TBox initiale, dans la section 6.2.1 (p.96).
- La composition des axiomes de la TBox, dans la section 6.2.2 (p.98), c'est-à-dire si un axiome possède, à gauche, des restrictions existentielles ou des conjonctions, en pourcentage. On les nomme axiomes composés, d'après la définition 25 (p.95).
- Le nombre de réponses à comparer (sur la plateforme Perfect Memory, le nombre de documents peut varier de quelques centaines à plusieurs millions), dans la section 6.2.3 (p.100).

— Le nombre de composantes, dans la section 6.2.4 (p.102).
— Le nombre de conjoncts maximum dans un axiome par composante, dans la section 6.2.5 (p.104).
— La profondeur maximale d'un axiome ou d'une réponse dans la section 6.2.6 (p.106).

Definition 25. *On nomme axiome composé les axiomes de la forme :*
$\exists r.A \sqsubseteq B$ *ou* $A \sqcap B \sqsubseteq C$*, qui sont les axiomes restants dans la TBox après la classification de celle-ci par les algorithmes présentés dans la section 5.4 (p.79) et listés dans l'annexe B.1.2 (p.129).*

Les temps d'exécution sont mesurés aux moments suivants, avec les moments numérotés 1 appartenant au temps de préparation et les moments numérotés 2 appartenant au temps de traitement de la demande de l'utilisateur (d'après la figure 5.1 (p.76)) :

1.a. Fin de la normalisation de la TBox initiale
1.b. Fin de la classification de la TBox initiale normalisée
1.c. Fin du calcul des fill-up de l'ensemble des réponses
2.a. Fin du calcul du fill-up de la demande de l'utilisateur
2.b. Fin du calcul des relations de subsomption entre les réponses et la demande
2.c. Fin des comparaisons entre les réponses
2.d. Fin de l'ordonnancement et terminaison du programme

De plus, nous comptons les valeurs suivantes :
— Le nombre de comparaisons (qui est proportionnel au nombre de réponses et de composantes)
— Le nombre de comparaisons ayant nécessité un calcul de différence
— Le temps total passé à réaliser ces différences.

La génération aléatoire produit deux fichiers, l'un contenant la TBox, sous forme d'un tableau d'axiomes et l'autre contenant les réponses construites à partir des concepts contenus dans la TBox. La demande de l'utilisateur est la première réponse de la liste du fichier correspondant.

Les TBox générées pour faire les tests sont toutes des TBox générales et acycliques. Pour en générer une, nous la construisons de manière itérative (voir l'annexe B.5 (p.155)). À chaque itération, on ajoute un nouveau concept, qui sera subsumé par un nombre aléatoire, entre 1 et un maximum que nous avons défini à 4, de concepts déjà existants dans la TBox. Par conséquent, lors des itérations suivantes, ce concept ne sera utilisé que comme concept qui en subsume un autre. Cela garantit l'acyclicité de la TBox.

Pour générer aléatoirement les réponses, on forme des concepts composés de nom de concepts contenus dans la TBox associée et de restrictions existentielles générées aléatoirement, en fonction des valeurs de taille et de profondeur du test à réaliser. On propose en annexe un exemple de génération d'une TBox (exemple 44 (p.158)) et d'un jeu de réponse (exemple 45 (p.161)).

Ensuite, on réalise les tests en ne faisant varier qu'un seul des six facteurs présentés précédemment à la fois. Chacun de ces tests est réalisé dix fois avec des TBox et des jeux de réponses différents. Les résultats obtenus sont agrégés sous forme de moyennes.

Nous discutons ensuite des raisons de l'obtention de ces résultats et des possibilités d'amélioration de l'implémentation.

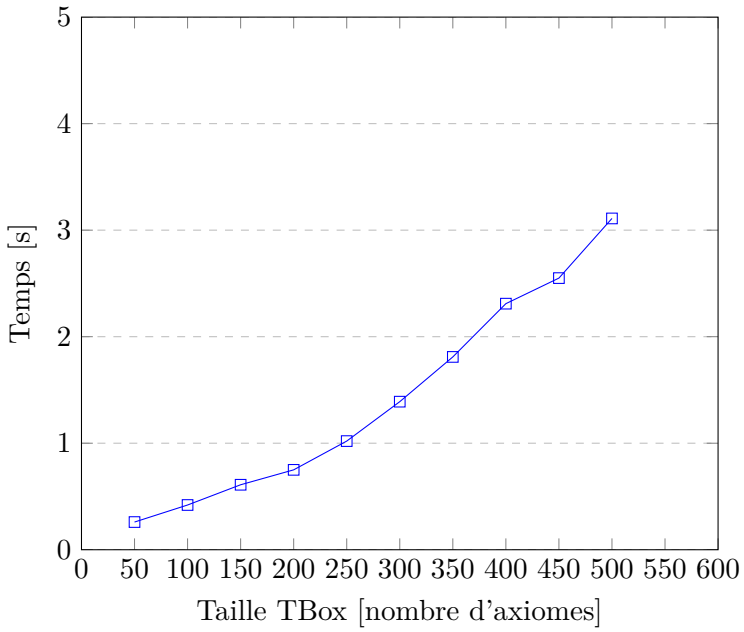
6.2.1 Tests avec variation de la taille de la TBox

Dans cette section, on étudie l'influence de la taille de la TBox sur le temps d'exécution du CCO. Les caractéristiques des tests sont présentées dans la table 6.9 (p.96). Le tableau des résultats complets se trouve à l'annexe B.6.1 (p.163). La taille considérée est le nombre d'axiome plutôt que la taille en octet de la TBox. En effet, lors de la génération de la TBox, comme on fixe le nombre maximum de conjoncts d'un axiome ou d'une réponse, il y a peu de variance sur la taille en octet de la TBox pour le même nombre d'axiome considéré.

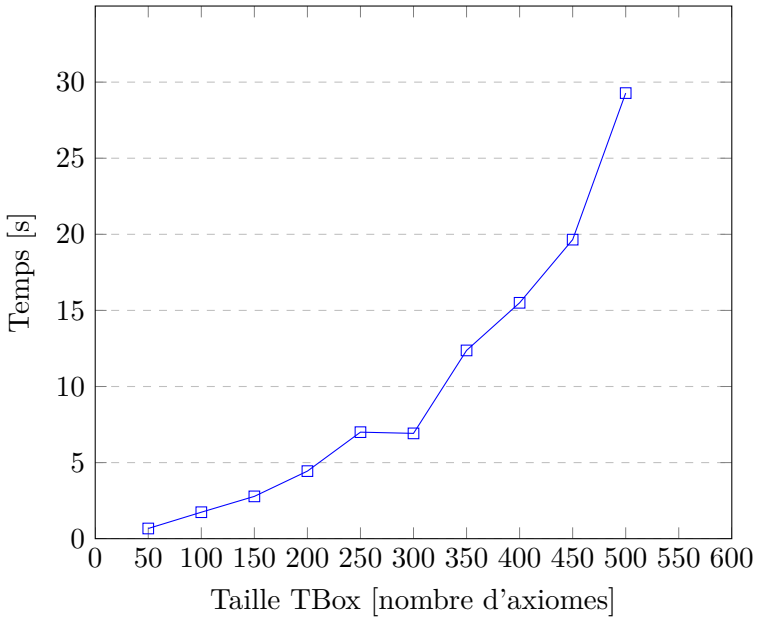
Variable	Valeur
Taille de la TBox (en nb d'axiomes)	de 50 à 500
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	20
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	99
Nombre de composantes	3
Nombre de conjoncts dans un axiome ou une réponse	3
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	1

TABLE 6.9 – Caractéristiques des tests pour l'étude de l'influence de la taille de la TBox sur le temps d'exécution.

Influence de la taille de la TBox sur le temps de préparation



Influence de la taille de la TBox sur le temps de traitement



Résultats Les résultats sont deux courbes d'allure polynomiale pour la préparation et le temps de traitement de la demande. En effet, la taille de la TBox intervient durant ces deux périodes. Pendant la préparation, la TBox est normalisée et classifiée, ce qui

est obtenu en temps polynomial d'après Baader et al. (2005, 2017). De la même manière, pendant le temps de traitement de la demande, la détermination des relations de subsomption entre la requête et les réponses, puis entre les rest et les miss des réponses s'effectue par l'intermédiaire d'une classification, expliquant le temps polynomial.

Notons également que plus la taille de la TBox augmente, plus il est nécessaire de calculer des différences, car les chances d'obtenir des relations de subsomption entre la demande et les réponses par génération aléatoire se réduisent. C'est-à-dire que le temps de traitement augmente également parce que le nombre de comparaisons à effectuer augmente et pas seulement parce que la détermination des relations de subsomption prend plus de temps.

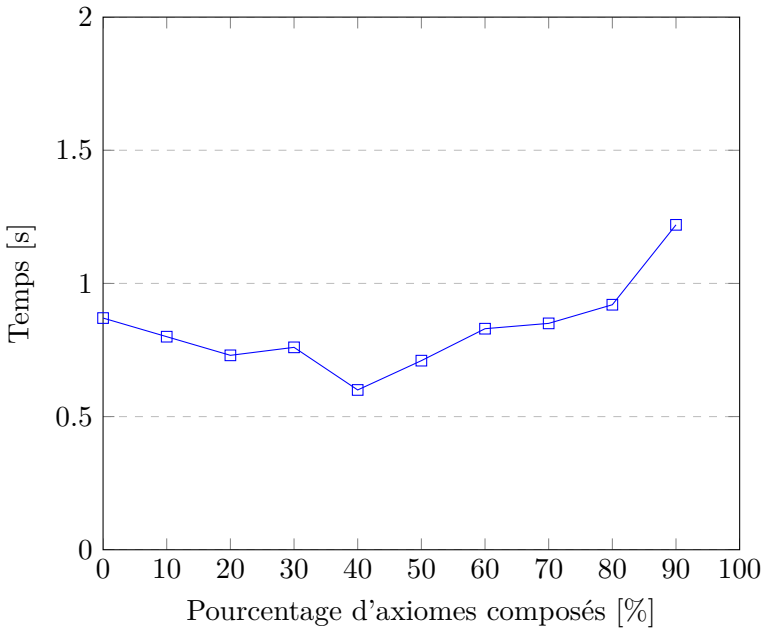
6.2.2 Tests avec variation du nombre d'axiomes composés dans la TBox

Dans cette section, on étudie l'influence de la composition des axiomes de la TBox sur le temps d'exécution du CCO. Les caractéristiques des tests sont présentées dans la table 6.10 (p.98). Le tableau des résultats complets se trouve à l'annexe B.6.2 (p.164).

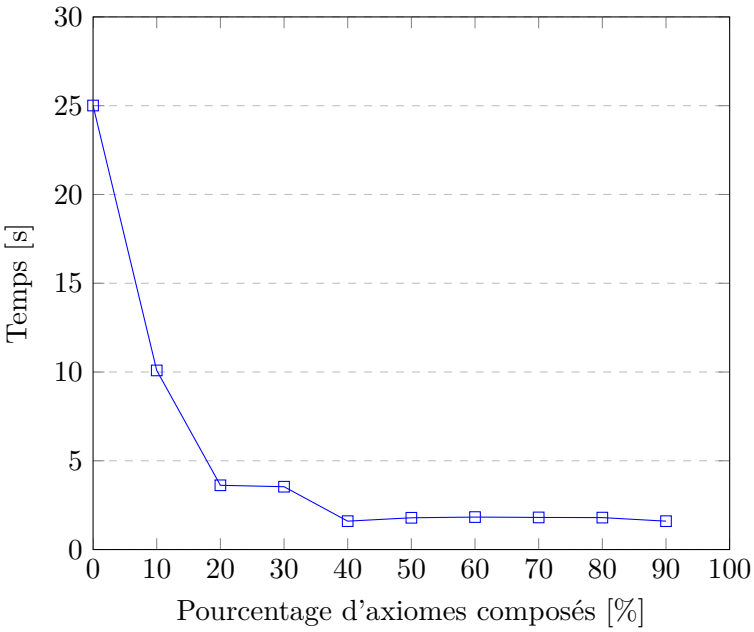
Variable	Valeur
Taille de la TBox (en nb d'axiomes)	200
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	de 0 à 90
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	99
Nombre de composantes	3
Nombre de conjoncts dans un axiome ou une réponse	3
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	1

TABLE 6.10 – Caractéristiques des tests pour l'étude de l'influence de la composition des axiomes sur le temps d'exécution.

Influence du nombre d'axiomes composés dans la TBox sur le temps de préparation



Influence du nombre d'axiomes composés dans la TBox sur le temps de traitement



Résultats Le nombre d'axiomes composés n'a pas vraiment d'influence sur le temps de préparation. Pour le temps de traitement de la demande, moins il y a d'axiomes composés, plus le temps de traitement est élevé. Ce temps élevé est dû au fait que sans axiome composé, il y a beaucoup plus de cas de subsomption de type $A \sqsubseteq B$ et $A \sqsubseteq \exists r.B$,

ce qui augmente le nombre de relations de subsomption dans la classification de la TBox. Avec un nombre élevé d'axiome composé, la TBox après classification est plus grande (il reste beaucoup d'axiomes composés) mais la classification est plus petite (il y a moins de chance de satisfaire les règles qui utilisent des axiomes composés). Quand il y a peu d'axiomes composés, les fill-up augmentent en taille et le temps de traitement augmente. Après l'introduction de suffisamment d'axiomes composés, le temps de traitement n'est plus impacté et se stabilise.

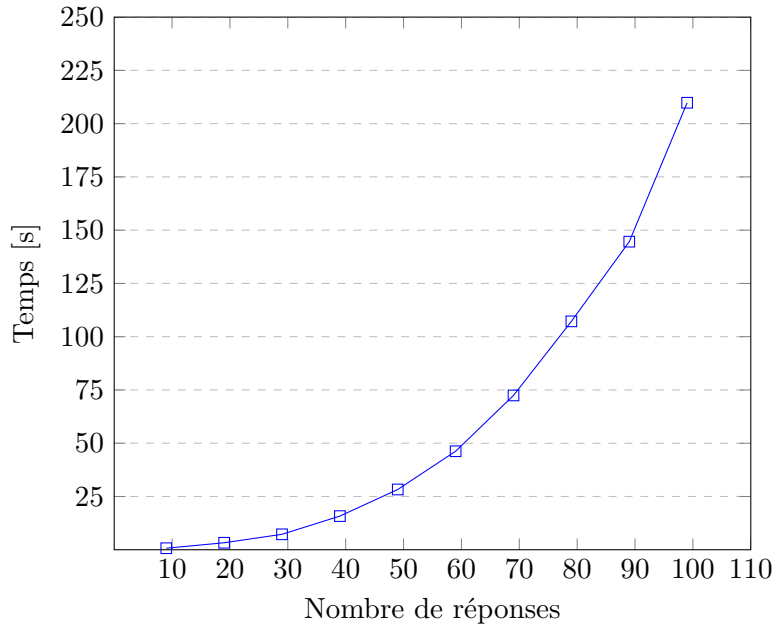
6.2.3 Tests avec variation du nombre de réponses

Dans cette section, on étudie l'influence du nombre de réponses sur le temps d'exécution du CCO. Les caractéristiques des tests sont présentées dans la table 6.11 (p.100). Le tableau des résultats complets se trouve à l'annexe B.6.3 (p.165).

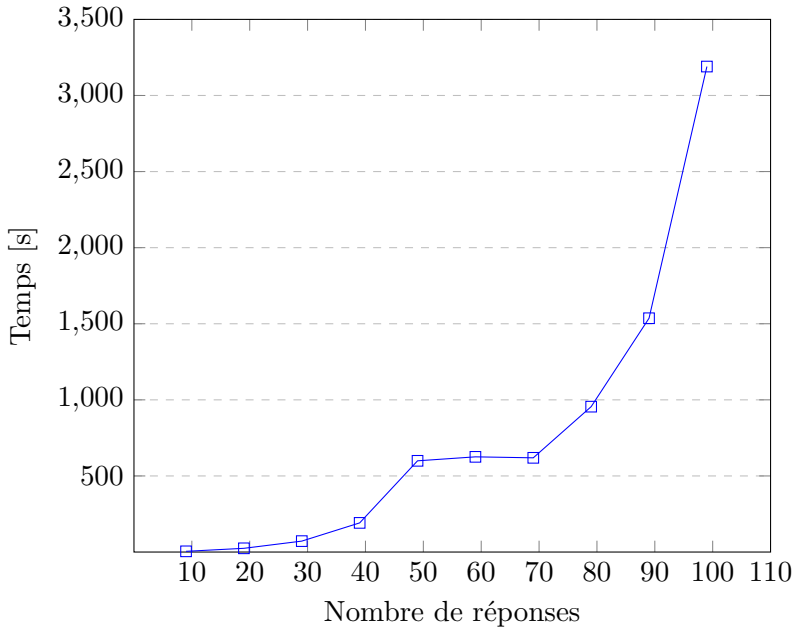
Variable	Valeur
Taille de la TBox (en nb d'axiomes)	200
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	20
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	de 9 à 99
Nombre de composante	3
Nombre de conjonct maximum dans un axiome ou une réponse	3
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	1

TABLE 6.11 – Caractéristiques des tests pour l'étude de l'influence du nombre de réponses sur le temps d'exécution.

Influence du nombre de réponses sur le temps de préparation



Influence du nombre de réponses sur le temps de traitement



Résultats Le nombre de réponses augmente significativement de manière polynomiale le temps de préparation et de traitement. Pour la préparation, la classification et les calculs des fill-up se font dans une TBox à laquelle on rajoute les réponses à étudier, ce qui augmente sa taille et augmente le nombre d'opérations proportionnellement au

nombre de réponses. Pour le temps de traitement de la demande, la partie calcul de subsomption et la partie CCO augmentent toutes les deux, mais en regardant plus précisément le tableau de l'annexe B.6.3 (p.165), c'est le calcul du CCO qui nécessite le plus de temps, car il doit effectuer un nombre quadratique de calculs de subsomption. On constate également lorsqu'on étudie les résultats individuellement qu'il arrive parfois qu'un cas particulier de création de TBox et de réponses se produise, et où la complexité exponentielle du fill-up augmente drastiquement le temps de traitement de la demande. Les valeurs élevées aux abscisses 50 et 60 dans la courbe du temps de traitement de la demande sont une conséquence de ces très mauvais cas.

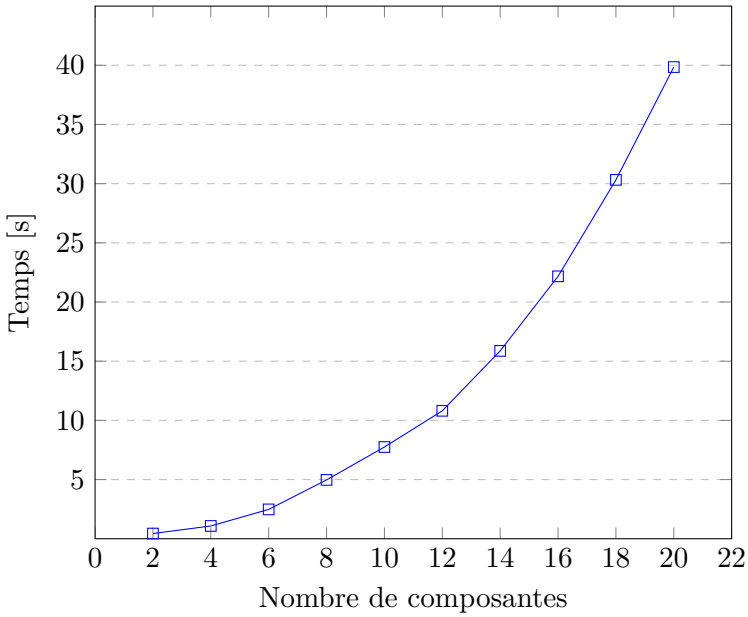
6.2.4 Tests avec variation du nombre de composantes

Dans cette section, on étudie l'influence du nombre de composantes sur le temps d'exécution du CCO. Les caractéristiques des tests sont présentées dans la table 6.12 (p.102). Le tableau des résultats complets se trouve à l'annexe B.6.4 (p.166).

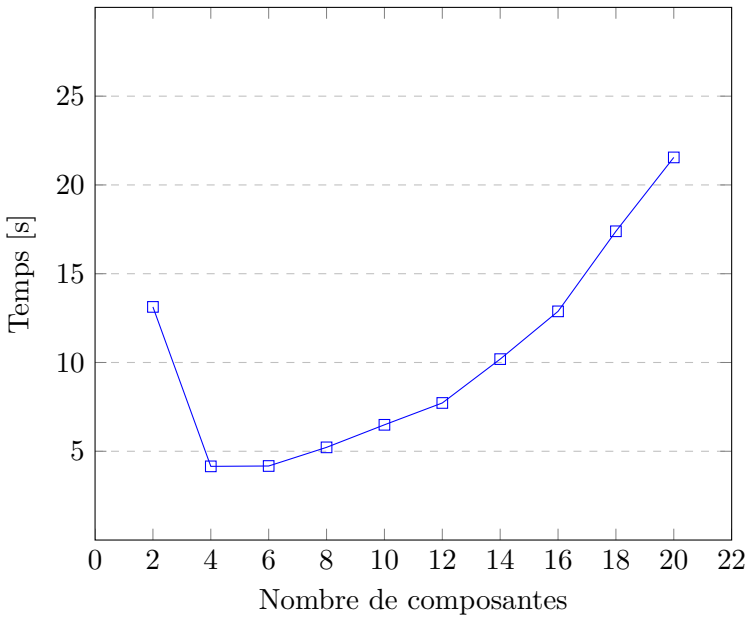
Variable	Valeur
Taille de la TBox (en nb d'axiomes)	200
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	20
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	99
Nombre de composantes	de 2 à 20
Nombre de conjoncts dans un axiome ou une réponse	3
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	1

TABLE 6.12 – Caractéristiques des tests pour l'étude de l'influence du nombre de composante sur le temps d'exécution.

Influence du nombre de composantes sur le temps de préparation



Influence du nombre de composantes sur le temps de traitement



Résultats Le temps de préparation augmente polynomialement en fonction du nombre de composantes. Le temps de traitement de la demande augmente polynomialement en fonction du nombre de composantes, mais on observe un cas particulier plus complexe quand il y a peu de composantes. En effet, de la manière dont est construite la TBox

dans les tests, moins il y a de composantes, plus il y a d'axiomes de la TBox concernant cette composante (voir définition 15 (p.38)). Par conséquent, il y a plus de concepts concernant cette composante et on observe le même résultat que pour la composition des axiomes : les fill-up sont de taille plus élevée et le temps de traitement augmente. Au dessus d'un certain nombre de composantes, le temps de traitement augmente car plus il y a de composantes, plus il y a de comparaisons de réponses à effectuer.

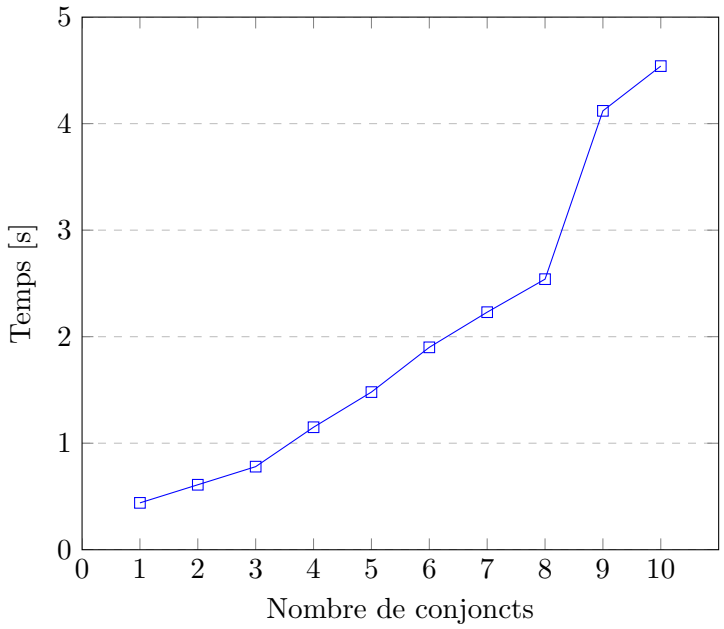
6.2.5 Tests avec variation du nombre de conjoncts dans les réponses

Dans cette section, on étudie l'influence du nombre de conjoncts sur le temps d'exécution du CCO. Les caractéristiques des tests sont présentées dans la table 6.13 (p.104). Le tableau des résultats complets se trouve à l'annexe B.6.5 (p.167). Une règle de génération particulière a été définie pour ce cas et l'ensemble des réponses étudiées dans cette section sont générées à chaque fois avec le maximum de conjoncts plutôt qu'un nombre aléatoire entre 1 et cette valeur (voir annexe B.5 (p.155)).

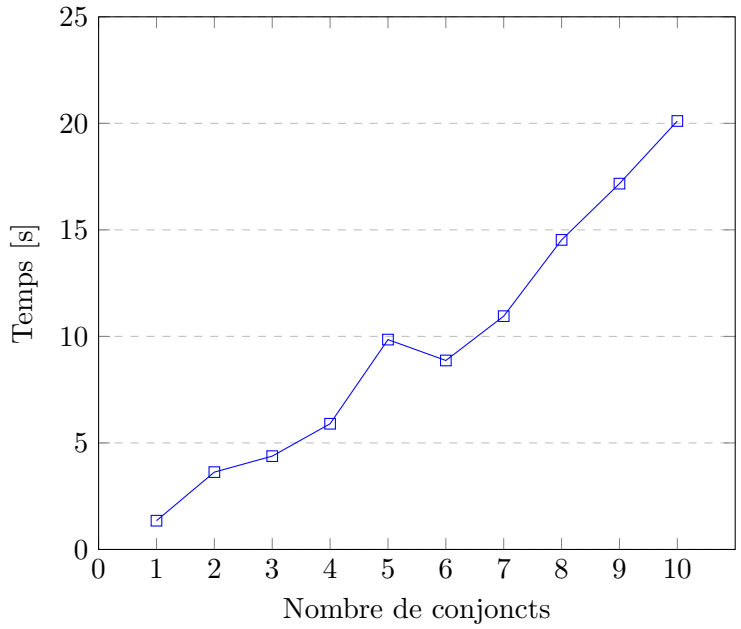
Variable	Valeur
Taille de la TBox (en nb d'axiomes)	200
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	20
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	99
Nombre de composantes	3
Nombre de conjoncts dans un axiome ou une réponse	de 1 à 10
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	1

TABLE 6.13 – Caractéristiques des tests pour l'étude de l'influence du nombre de conjonct sur le temps d'exécution.

Influence du nombre de conjoncts sur le temps de préparation



Influence du nombre de conjoncts sur le temps de traitement



Résultats Le temps de traitement semble augmenter proportionnellement au nombre de conjoncts. En effet, plus il y a de conjoncts, plus les calculs des fill-up sont longs et plus le résultat est un concept de grande taille, augmentant le temps de comparaison. De

plus, plus il y a de conjoncts, moins la demande et les réponses ont des chances d'avoir un lien de subsomption, ce qui augmente le nombre de différences à calculer.

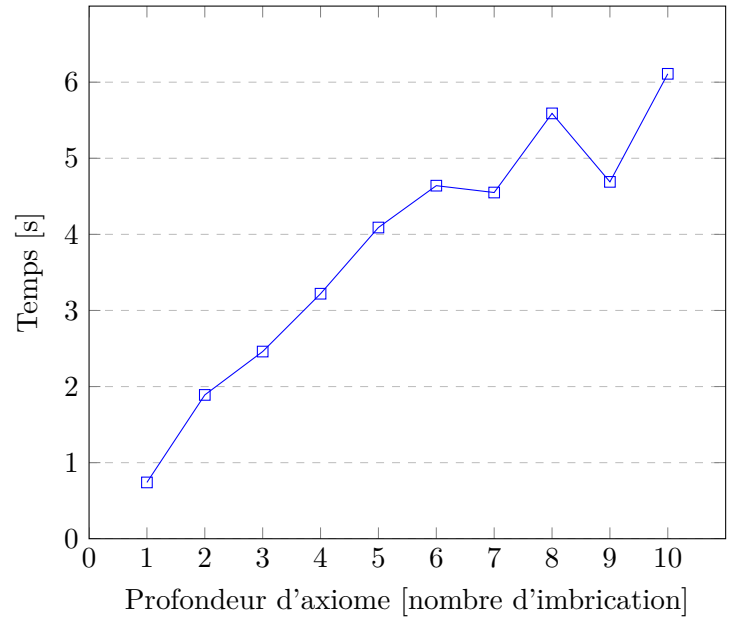
6.2.6 Tests avec variation de la profondeur des axiomes

Dans cette section, on étudie l'influence du nombre de réponses sur le temps d'exécution du CCO. Les caractéristiques des tests sont présentées dans la table 6.14 (p.106). Le tableau des résultats complets se trouve à l'annexe B.6.6 (p.168).

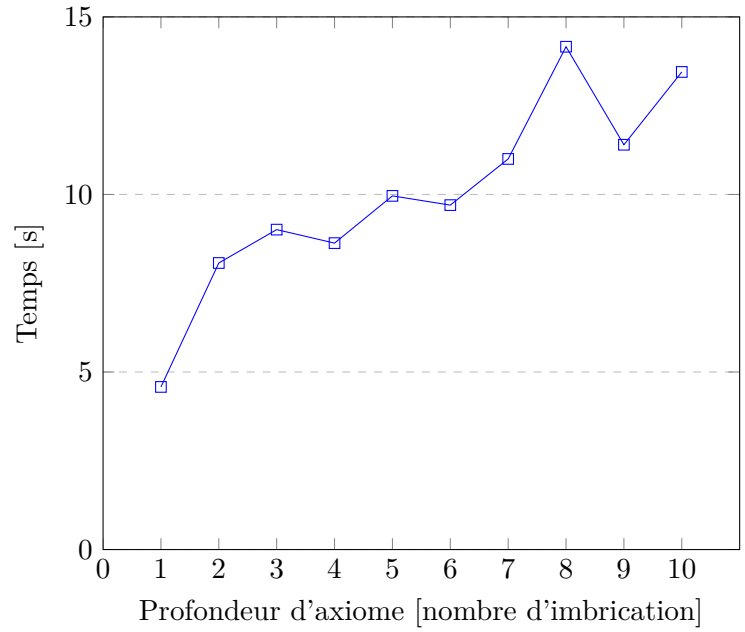
Variable	Valeur
Taille de la TBox (en nb d'axiomes)	200
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	20
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	99
Nombre de composantes	3
Nombre de conjoncts dans un axiome ou une réponse	3
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	1 à 10

TABLE 6.14 – Caractéristiques des tests pour l'étude de l'influence de la profondeur des axiomes sur le temps d'exécution.

Influence de la profondeur des axiomes sur le temps de préparation



Influence de la profondeur des axiomes sur le temps de traitement



Résultats Le temps d'exécution semble être proportionnel à la profondeur des restrictions existentielles. Notons qu'il n'y a, pour les tests, que trois rôles différents, et que s'il y avait davantage de rôles différents, le temps d'exécution du TSO serait réduit car les chances d'avoir un rôle identique lors de la différence serait grandement réduit.

6.2.7 Pistes d'améliorations

Variable	Allure de la courbe	Impact
Taille de la TBox (nb d'axiomes)	Polynomiale	Faible
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	Constante	Faible
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	Polynomiale	Fort
Nombre de composantes	Polynomiale	Moyen
Nombre de conjoncts dans un axiome ou une réponse	Linéaire	Faible
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	Linéaire	Faible

TABLE 6.15 – Synthèse des résultats des tests quantitatifs pour le temps de préparation. L'impact correspond à l'importance de la variable sur le temps de préparation. Les courbes sont croissantes, sauf mention contraire.

Variable	Allure de la courbe	Impact
Taille de la TBox (en nb d'axiomes)	Polynomial	Moyen
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions (en %)	Exponentielle décroissante	Faible
Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	Polynomiale	Très fort
Nombre de composantes	Polynomiale	Moyen
Nombre de conjoncts dans un axiome ou une réponse	Linéaire	Moyen
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	Linéaire	Moyen

TABLE 6.16 – Synthèse des résultats des tests quantitatifs pour le temps de traitement. L'impact correspond à l'importance de la variable sur le temps de préparation. Les courbes sont croissantes, sauf mention contraire.

Les tables 6.15 et 6.16 font la synthèse des résultats obtenus dans les sections précédentes. Les trois facteurs ayant le plus d'impact sur le temps de traitements, par ordre d'importance sont le nombre de réponse, le nombre de composante et la taille de la TBox. Nous proposons ensuite les pistes d'améliorations suivantes.

Le nombre de réponses est le principal facteur influençant sur le temps de traitement de la demande puisqu'il est nécessaire de comparer toutes les réponses les unes avec les autres. Par conséquent, tout filtre en mesure d'écarter des réponses avant le CCO augmentera les performances de ce calcul. On peut par exemple :

- Ne pas sélectionner les réponses n'ayant aucune composante en commun avec la demande,
- Ne pas prendre en compte les réponses n'étant pas subsumées par la demande si un certain seuil de réponses subsumées par la demande a été atteint,
- Ne pas prendre en compte les réponses dont le rest est l'intégralité de la demande,
- Ne sélectionner que des réponses qui seraient les résultats exacts de la même demande utilisateur dont on aurait retiré un certain nombre de composantes.

Pour optimiser le traitement de la TBox, il est possible de découper la TBox en autant de fragments de TBox que de composantes permet d'éviter le parcours des autres composantes lors d'un traitement. Ce n'est cependant possible que si la construction de la TBox par l'expert est effectuée de manière à ce que chaque composante soit indépendante des autres.

Pour un traitement le plus performant possible, il faudra enfin trouver le bon équilibre dans le découpage de la TBox en composantes, pour éviter d'en avoir un nombre plus important que nécessaire.

Pour les autres critères (profondeur, nombre de conjoncts), il est difficile de leur imposer des contraintes en vue d'optimiser le temps de traitement, car ces critères sont relatifs à la construction de la TBox et à l'expression des ressources et de la demande utilisateur.

Enfin, il est intéressant de noter que ce qui nécessite le plus de temps dans le temps de traitement de la demande est la détermination du lien de subsomption entre chaque rest et chaque miss calculés (voir ligne 10 des tableaux de l'annexe B.6 (p.162)). Cette détermination passe par la classification de la TBox initiale et des deux rest (resp. miss) à comparer. On rappelle que le calcul de subsomption est PTIME par rapport aux entrées, et qu'il faut considérer que l'on peut, au pire, effectuer $\mathcal{O}(n^2)$ calculs de subsomption pour les n réponses potentielles par composantes.

Si on souhaitait accélérer le processus, on pourrait modifier l'algorithme du CCO afin qu'il ne compare que la taille du rest et du miss. On obtient alors une approximation du CCO, ce qui réduit fortement le temps d'exécution. En annexe B.7 (p.168) est proposé un exemple de temps d'exécution pour le CCO tel qu'implémenté et un CCO ne faisant aucun calcul de subsomption pour ses comparaisons de rest et de miss. On constate que l'approximation du CCO ne comparant que les tailles est beaucoup plus performante, mais n'obtient pas les mêmes résultats. Le classement général est cependant assez proche pour être utilisé dans les situations où le temps de traitement est plus important que la précision du résultat.

Conclusion et perspectives

Dans un contexte de recherche de ressources par un utilisateur sur une plateforme documentaire utilisant les logiques de descriptions, nous avions pour objectif de fournir à cette plateforme un moyen de donner en réponse à un utilisateur les réponses les plus proches de sa demande en cas d'absence de résultats exacts. Pour cela, nous avons réalisé un procédé de matchmaking multicritère sémantique, le Component Comparison Operator, utilisant le principe de la concordance relative lié à un découpage original des concepts en composantes. Cet opérateur détermine la proximité de chaque ressource avec la demande de l'utilisateur pour chaque composante. Cette proximité est obtenue grâce à des calculs de différence mettant en exergue les informations manquantes et superflues. Un nouvel opérateur de différence pour $\mathcal{EL}$, le Component Subtraction Operator, qui a été défini pour les TBox acycliques et définitionnelles, et son exécution est dans EXPTime par rapport à la taille de la TBox et de ses deux opérandes, son exécution est PTime par rapport à la taille de la TBox étendue et de ses deux opérandes. Le CSO utilise lui-même un opérateur de différence syntaxique pour $\mathcal{EL}$, le TSO, qui retire du diminuende tous les concepts présents dans le diminuteur, en conservant la structure arborescente de ces concepts. Le TSO a une complexité PTime par rapport à la taille de ses opérandes. Nous proposons également une version heuristique du CSO pour les TBox acycliques et générales, utilisant une nouvelle notion appelée fill-up qui effectue une saturation sémantique d'un concept. En contribution annexe nous avons proposé un état de l'art des opérateurs de différences pour les LD et un cadre de comparaison des approches de matchmaking.

L'implémentation de l'heuristique du CSO a été réalisée en Ruby, en prévision d'une intégration à la plateforme de gestion documentaire Perfect Memory. Cette implémentation est en mesure de récupérer des informations sur cette plateforme pour créer la TBox et la liste de réponses correspondantes.

Des tests qualitatifs et quantitatifs ont été réalisés. Les tests qualitatifs sur des jeux de tests unitaires donnent des résultats conformes à ceux attendus. Les tests quantitatifs, sur des jeux de tests générés aléatoirement, montrent qu'il est pour le moment coûteux en temps d'exploiter le CCO, car le processus demande pour chaque calcul de subsomption de réaliser une classification complète ou partielle. On pourrait s'affranchir de ce coût en temps avec un moyen plus rapide de calculer un lien de subsomption ou en ne comparant que les tailles des résultats (et plus leur lien de subsomption).

Dans le futur, nous souhaiterions proposer une version du CSO fonctionnant avec les TBox générales et cycliques. Dans ce cas il est nécessaire de redéfinir le CSO, car la définition actuelle utilise la notion de concepts primitifs qui n'existent pas en dehors des TBox définitionnelles. De plus, il serait souhaitable de proposer une caractérisation différente de la subsomption dans ce contexte. La classification de Baader et al. (2005) pourrait être utilisée mais elle impose également de réaliser des calculs superflus lors de la détermination des relations de subsomption entre deux concepts. Il serait plus efficace d'avoir une caractérisation de la subsomption entre deux concepts. Nous pensons utiliser pour cela des hypergraphes orientés.

Par ailleurs, nous envisageons de définir nos trois opérateurs (CCO, CSO, TSO) dans des logiques de descriptions plus complexes, comme $\mathcal{EL}++$ dont la classification reste

polynomiale par rapport à la taille de la TBox.

Enfin, nous aimerions optimiser l'implémentation du CCO, en mettant en place les différents procédés de filtre de réponses avant la comparaison et comparer avec l'implémentation actuelle. Nous souhaiterions également finaliser l'intégration à la plateforme Perfect Memory et tester le module en situation réelle.

—

Bibliographie

F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In Thomas Dean, editor, *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 96–103. Morgan Kaufmann, 1999. URL `http://ijcai.org/Proceedings/99-1/Papers/015.pdf`.

F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope. In *International Joint Conferences on Artificial Intelligence (IJCAI-05)*, 2005. URL `http://www.ijcai.org/papers/0372.pdf`.

F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017. doi : 10.1017/9781139025355.

B. Benatallah, M.-S. Hacid, C. Rey, and F. Toumani. Request rewriting-based web service discovery. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *The Semantic Web - ISWC 2003*, pages 242–257, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-39718-2.

S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. In *in Proceedings of the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR'2002), Toulouse, France*. D. Fensel, F. Giunchiglia et al., Eds. San Francisco, California, USA : Morgan Kaufmann Publishers, 2002.

S. Colucci, T. Di Noia, E. Di Sciascio, F. Donini, and M. Mongiello. Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. *Electronic Commerce Research and Applications*, 4 :345–361, 03 2004. doi : 10.1016/j.elerap.2005.06.004.

HJ. Heinz. How i lost my owl : Retracting knowledge from el concepts. Master's thesis, Koblenz-Landau University, 2018.

P. Marquis, O. Papini, and H. Prade, editors. *Panorama de l'intelligence artificielle, ses bases méthodologiques, ses développements*, chapter 13.2. Cépaduès Éditions, 2014.

A. Mascaro and C. Rey. Recommandation ontologique multicritère pour la métrologie. In *Rencontres des Jeunes Chercheur · ses en Intelligence Artificielle, in PFIA 2020, Angers*, 2020.

M. A. Musen. The protégé project : A look back and a look forward. *AI Matters*, 1(4) : 4–12, jun 2015. doi : 10.1145/2757001.2757003. URL `https://doi.org/10.1145/2757001.2757003`.

B. Nebel. Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43(2) :235–249, 1990. ISSN 0004-3702. doi : https://doi.org/10.1016/0004-3702(90)90087-G. URL `https://www.sciencedirect.com/science/article/pii/000437029090087G`.

T. Di Noia, E. Di Sciascio, F. M. Donini, and M. Mongiello. Abductive matchmaking using description logics. In *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence IJCAI-03*, pages 337–342, Acapulco, Mexico, Aug. 2003. MK. URL `http://sisinflab.poliba.it/Publications/2003/DDDM03`.

T. Di Noia, E. Di Sciascio, and F. M. Donini. Semantic matchmaking as non-monotonic reasoning : A description logic approach. *CoRR*, abs/1110.2742, 2011. URL `http://arxiv.org/abs/1110.2742`.

R Peñaloza and A-Y Turhan. A practical approach for computing generalization inferences in el. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *The Semantic Web : Research and Applications*, pages 410–423, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

T. Rienstra, C. Schon, and S. Staab. Concept contraction in the description logic el. In *International Conference on Principles of Knowledge Representation and Reasoning (KR-20)*, page 723–732, 2020.

F. Suchanek, C. Menar, M. Bienvenu, and C. Chapellier. Technical report : a language for combinatorial creativity. Technical report, Telecom ParisTech, sep 2016.

F. M. Suchanek and C. Menard. Can you imagine... a language for combinatorial creativity ? In *First Part of the Proceededings of the 15th International Semantic Web Conference (ISWC'2016), Kobe, Japan*, volume v.9981 of *Lecture Notes in Computer Science, P. T. Groth, E. Simperlet al., Eds.*, pages 532 – 548. Cham, Switzerland :Springer International Publishing, 2016.

G. Teege. Making the difference : A subtraction operation for description logics. In *International Conference on Principles of Knowledge Representation and Reasoning*, 1994. URL `https://api.semanticscholar.org/CorpusID:1817927`.

Annexes

Annexe A

Précisions formelles à propos des opérateurs de différence

A.1 Teege

La description informelle de l'opérateur se trouve dans la section 1.3.1 (p.24).

L'opérateur de Teege (1994) est défini ainsi :

Definition 26. $C \ominus_{Te} D = max_{\sqsubseteq}\{E \in \mathcal{L} \mid E \sqcap D \equiv C\}$, *avec* $C \sqsubseteq D$

A.2 Brandt

La description informelle de l'opérateur se trouve dans la section 1.3.2 (p.24).

Dans $\mathcal{EL}$, un concept C ne possède pas de redondance s'il n'existe pas de concept $C' \equiv C$ tel qu'il est possible d'obtenir C' à partir de C :

- En retirant des concepts primitifs de C
- En retirant des restrictions existentielles de C
- En remplaçant le contenu des restrictions existentielles de C par leur contenu privé de ses redondances

Voir Brandt et al. (2002) pour la normalisation dans $\mathcal{ALC}$. La forme normale pour $\mathcal{EL}$ est celle de $\mathcal{ALC}$ privée des unions et de la négation :

- $norm(C) = \top$, si $C \equiv \top$

L'opérateur de Brandt et al. (2002) est défini ainsi :

Definition 27. *Soient* $\prec_d$ *un ordre syntaxique défini dans* $\mathcal{ALC}$ *Brandt et al. (2002), et* C *et* D *deux concepts normalisés selon* $\prec_d$*, respectivement dans* $\mathcal{ALC}$ *et* $\mathcal{ALE}$*.*
$C \ominus_{Br} D = min_{\preceq_d}\{E \in \mathcal{L} \mid E \sqcap D \equiv C \sqcap D\}$.

A.3 Suchanek

La description informelle de l'opérateur se trouve dans la section 1.3.3 (p.25).
Suchanek et al. (2016) définissent la notion de concept entièrement réduit :

Definition 28 (Concept entièrement réduit)**.** *Un concept C est entièrement réduit, noté $red(C)$, s'il n'existe pas un autre concept C' tel que*

1. *$C' \sqsubseteq C$*
2. *Tous les conjoncts de C' apparaissent dans C*
3. *C' possède moins de conjoncts que C*

Ils définissent la forme normale d'un concept C, notée $norm(C)$, telle que :
— $norm(C) = C$, si C est un concept primitif ou $\top$
— $norm(\exists r.C) = \exists r.norm(C)$
— $norm(C_1 \sqcap ... \sqcap C_n) = red(norm(C_1) \sqcap ... \sqcap norm(C_n))$
Enfin, Suchanek et al. (2016) définissent un ordre complet $\prec$ pour des concepts entièrement réduits basé sur deux ordres complets $\prec_{N_\mathbf{C}}$ et $\prec_{N_\mathbf{r}}$, respectivement sur les concepts primitifs et $\top$, $\mathbf{C} \cup \top$ et sur les rôles , tels que :
— $A \prec B$ ssi $A \prec_{N_\mathbf{C}} B$, A, $B \in \mathbf{C}$
— $A \prec C$, pour $A \in \mathbf{C}$ et $C \notin \mathbf{C}$
— $\exists r.C \prec D$ pour une conjonction D
— $\exists r.C \prec \exists s.D$ ssi $r \prec_{N_\mathbf{r}} s$
— $\exists r.C \prec \exists r.D$ pour $C \prec D$
— $C \prec D$, pour des conjonctions C, D par l'extension lexicographique de $\prec$
L'opérateur de Suchanek et al. (2016) est défini ainsi :

Definition 29. *Soient C un concept tel que $norm(C) = C_1 \sqcap \ldots \sqcap C_n$ et D un concept. Si B est une conjonction d'un seul concept :*

Si $\exists k \mid k = argmin_i\{C_i \mid C_i \sqsubseteq D\}$:

$C \ominus_{Su} D = norm(C_1 \sqcap ... \sqcap C_{k-1} \sqcap C_{k+1} \sqcap ... \sqcap C_n)$

Sinon : $C \ominus_{Su} D = C$

Si D est une conjonction de concepts, D possède $m \geq 2$ conjoncts et $norm(D) = D_1 \sqcap \ldots \sqcap D_m$,

$C \ominus_{Su} D = (...((C \ominus_{Su} D_1) \ominus_{Su} ...) \ominus_{Su} D_m)$

A.4 Heinz

La description informelle de l'opérateur se trouve dans la section 1.3.4 (p.25).
Heinz (2018) utilise les concepts entièrement réduits et la normalisation proposée par Suchanek et al. (2016), présentée dans A.3 (p.120). Ils proposent cependant un ordre $\prec_{He}$ différent, avec $\prec_{N_\mathbf{C}}$ un ordre complet sur les concepts primitifs, sans $\top$ et $\prec_{N_\mathbf{r}}$ un ordre complet sur les rôles sans u (u est le rôle tel que $\forall$ rôle r, $r \sqsubseteq u$) :
— $C \prec \top$ pour $C \neq \top$
— $A \prec B$ ssi $A \prec_{N_\mathbf{C}} B$, pour A, $B \in \mathbf{C}$

— $C \prec A$, pour $A \in \mathbf{C}$ et $C \notin \mathbf{C}$
— $\exists r.C \prec \exists s.D$ ssi $r \prec_{N_\mathbf{r}} s$
— $\exists r.C \prec \exists r.D$ pour $C \prec D$
— $\exists r.C \prec \exists u.D$
— $\exists u.C \prec \exists u.D$ pour $C \prec D$
— $C_1 \sqcap ... \sqcap C_n \prec D$ avec $n > 1$ $C_1 \prec$ D et la conjonction est déjà ordonnée selon $\prec$
— $D \prec C_1 \sqcap ... \sqcap C_n$ avec $n > 1$ $D \prec C_1$ ou $D = C_1$
— $C_1 \sqcap ... \sqcap C_n \prec P_1 \sqcap ... \sqcap P_m$ avec $n > 1$, $m > 1$ et $C_1 \prec P_1$ ou si $C_1 = P_1$, $C_2 \sqcap ... \sqcap C_n \prec P_2 \sqcap ... \sqcap P_m$

L'opérateur de différence de Heinz (2018) est défini ainsi :

Definition 30. *Soient $\mathcal{T}$ une TBox $\mathcal{EL}$ normalisée, définitionnelle et acyclique, deux concepts de $\mathcal{T}$ C et D dont les redondances ont été retirées (voir A.4 (p.120)).*

Si $C \not\sqsubseteq D$, alors $C \ominus_{He} D = C$

Sinon (i.e. $A \sqsubseteq B$) :

Cas 1 : C est un nom de concept : on a $C \ominus_{He} D = \top$

Cas 2 : $C = C_1 \sqcap ... \sqcap C_n$

on a $C \ominus_{He} D = \sqcap_{i=1}^{n}(C_i \ominus_{He} D_\prec)$

Cas 3 : $C = \exists r.C_1$ et

3.a : $D_\prec = \exists r.D_1$ et $C_1 \ominus_{He} D_1 \not\equiv \top$

on a$C \ominus_{He} D = \exists r.(C_1 \ominus_{He} D_1)$

3.b : $D_\prec = \exists r.D_1$ et $C_1 \ominus_{He} D_1 \equiv \top$

on a$C \ominus_{He} D = \top$

3.c : $D \equiv \top$: on a $C \ominus_{He} D = \top$

3.d : autre : on a $C \ominus_{He} D = C$

Avec C_i qui n'est pas une conjonction et $\forall i$ $D_\prec$ est le conjonct le plus petit de D par rapport $\prec$, c'est-à-dire le premier conjonct de D.

A.5 Rienstra

La description informelle de l'opérateur se trouve dans la section 1.3.5 (p.26).

Definition 31. *Soit $\mathcal{T}$ une TBox une définitionnelle et acyclique et C et D deux concepts. On a C' un résidu de C par rapport à D si :*

a. $C \sqsubseteq C'$

b. $C' \not\sqsubseteq D$

c. $\nexists\, C''$ *tel que* $C \sqsubseteq C''$, $C'' \not\sqsubseteq D$ *et* $C'' \sqsubseteq C'$.

L'ensemble des résidus de deux concepts C et D, modulo la commutativité de la conjonction est noté $C \perp D$ [15].

Une fonction de sélection σ est utilisée pour identifier les meilleurs résidus.

15. Plus de détails sont donnés dans Rienstra et al. (2020) sur les classes d'équivalence des résidus.

Definition 32. *Soient deux concepts A et B. Une fonction de sélection σ choisit, pour chaque paire (C, D) un ensemble $\sigma(C \perp D)$ de résidus de C par rapport à D tel que :*

1. *Si $C \perp D \neq \emptyset$ alors $\sigma(C \perp D) \neq \emptyset$*
2. *Si $C \perp D \neq \emptyset$ alors $\sigma(C \perp D) \subseteq C \perp D$*
3. *Si $C \perp D = \emptyset$ alors $\sigma(C \perp D) = \{C\}$*

σ est appelée maxi-choice si la fonction ne choisit qu'un seul élément de $A \perp B$. Si elle choisit l'intégralité de $A \perp B$, σ est appelée full meet.

L'opérateur de différence de Rienstra et al. (2020) est défini ainsi :

Definition 33. $A \ominus_{Ri} B = LCS(\sigma(C \perp D))$
(voir A.6 pour plus de détail sur le LCS)

La figure A.1 (p.123) est le treillis de généralisation du concept $Dem\ PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$, obtenu par l'utilisation du MSG (Most General Concept), voir Rienstra et al. (2020). Considérons la contraction $PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique) \ominus_{Ri} PaC \sqcap Numérique \sqcap \exists aMatériau.Acier$. Les résidus de Dem sont les concepts entourés d'un trait plein, ce sont les résultats possibles de la contraction si σ est une fonction de *maxi-choice*. Si σ est une fonction *full meet* alors son résultat est le concept entouré de pointillés gras, qui est le LCS des résidus. En pointillés normaux sont représentés les résultats intermédiaires de la détermination du résultat par la fonction *full meet*.

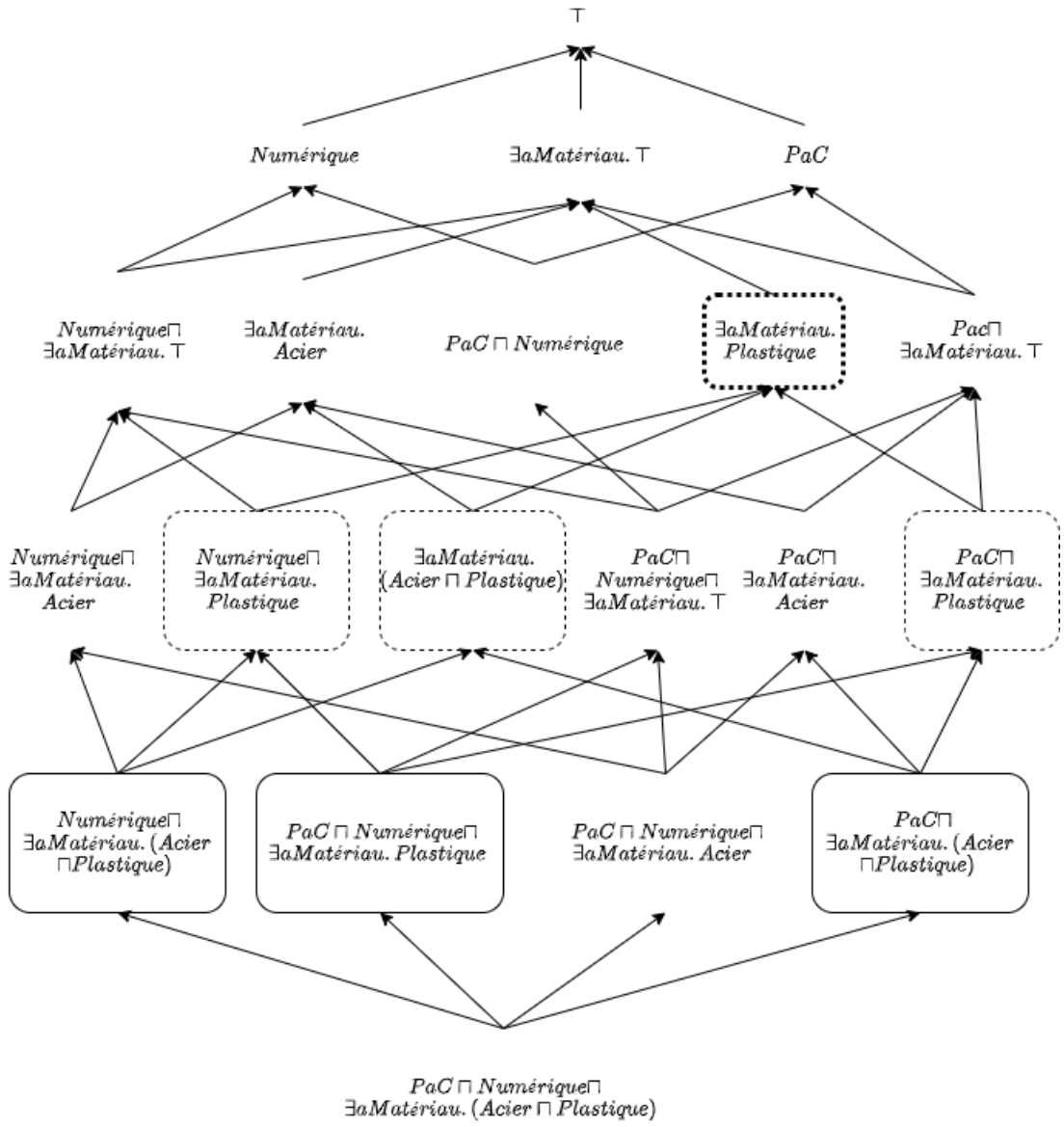


FIGURE A.1 – Treillis de généralisation de $PaC \sqcap Numérique \sqcap \exists aMatériau.(Acier \sqcap Plastique)$.

A.6 LCS

Le Least Common Subsumer (LCS) de deux concepts C et D est le concept le plus spécifique qui subsume à la fois C et D pour une TBox et une LD données :

Definition 34 (de Peñaloza and Turhan (2011))**.** *Avec $\mathcal{L}$ une logique de description, $\mathcal{T}$ une TBox de $\mathcal{L}$ et C_1, ..., C_n un ensemble de n concepts, le Least Common Subsumer de C_1, ..., C_n dans $\mathcal{L}$ par rapport) $\mathcal{T}$ est le concept le plus spécifique qui subsume C_1, ..., C_n, c'est-à-dire un concept D tel que :*

1. *$C_i \sqsubseteq_{\mathcal{T}} D$, $\forall i = 1, ..., n$*
2. *Si E est un concept de $\mathcal{L}$ qui satisfait $C_i \sqsubseteq_{\mathcal{T}} E$, $\forall i = 1, ..., n$, alors $D \sqsubseteq_{\mathcal{T}} E$*

Exemple 38. *Soit la TBox $\mathcal{EL}$ $\mathcal{T}$ suivante :* $\mathcal{T} = \{Fer \sqsubseteq Métal,$
$Argent \sqsubseteq Métal,$
$Métal \sqsubseteq Matériau,$
$Bois \sqsubseteq Matériau\}$

$LCS(Fer, Argent) = Métal.$
$LCS(Fer, Bois) = Matériau.$

A.7 Synthèse des opérateurs de différence dans les logiques de description

Une version informelle de cette synthèse se trouve dans la section 1.3.6 (p.27). Soient les quatre critères de comparaison des opérateurs de différences que nous proposons :

- Leur type ① : S pour une soustraction (on retire du diminuende) ou C pour une complétion (on ajoute au diminuteur)
- Nature de la définition ② : T pour syntaxique et M pour sémantique, B pour les deux
- Leur critère d'optimalité ③ : T pour syntaxique et M pour sémantique ou − si non applicable
- La capacité à retirer des concepts des restrictions existentielles ④ : O pour Oui et N pour Non

Table A.1: Comparaison des opérateurs de différence dans les LD.

Définition	①	②	③	④	Remarques
$A \ominus_{Te} B =$ $max_{\sqsubseteq}(\{C \in \mathcal{L} \mid B \sqcap C \equiv A\})$	C	M	M	N	• Voir Teege (1994) • Défini pour une LD quelconque $\mathcal{L}$ • non défini si $A \not\sqsubseteq B$

$A \ominus_{Br} B =$ $min_{\preceq}(\{C \in L : C \sqcap B \equiv A \sqcap B\})$	C	B	T	N	• Voir Brandt et al. (2002) • A est dans $\mathcal{ALC}$, B est dans $\mathcal{ALE}$
$A \ominus_{Ri} B = LCS(\sigma(A \perp B))$ avec $A \perp B = \{C \mid$ a. $A \sqsubseteq C$ b. $C \not\sqsubseteq B$ c. $\nexists\, C''$ tel que $C'' \not\sqsubseteq D$ et $C'' \sqsubseteq D\}$.	S	M	M	O	• Voir A.5 (p.121) et Rienstra et al. (2020) • voir A.6 (p.124) pour plus de détail sur le LCS • σ est une fonction de *maxi-choice* ou de *full meet* • $A \ominus_{Ri} B = \top$ si $B \equiv \top$ ou si $A \equiv B$ • $A \ominus_{Ri} B = A$ si $A \not\sqsubseteq B$
Cas 1 : $A \ominus_{Su} B =$ Si $\exists k \mid k = argmin_i\{C_i \mid C_i \sqsubseteq B\}$: $A \ominus_{Su} B =$ $norm(C_1 \sqcap ... \sqcap C_{k-1} \sqcap C_{k+1} \sqcap ... \sqcap C_n)$ Sinon : $A \ominus_{Su} B = A$ Cas 2 : $A \ominus_{Su} D =$ $(...((A \ominus_{Su} D_1) \ominus_{Su} ... \ominus_{Su} D_m)$	S	B	–	N	•Voir Suchanek and Menard (2016) et A.3 (p.120) • Défini dans $\mathcal{EL}$ • B est une conjonction d'un seul conjonct • D a $m \geq 2$ conjoncts et $norm(D) = D_1 \sqcap \ldots \sqcap D_m$. • $norm(A) = C_1 \sqcap \ldots \sqcap C_n$ • $\forall i \in \{1, \ldots, n-1\}, C_i \prec C_{i+1}$ • $\prec$ est un ordre total syntaxique pour des concepts $\mathcal{EL}$.
Si $A \not\sqsubseteq B$, alors $A \ominus_{He} B = A$ Sinon ($A \sqsubseteq B$) : Cas 1 : A est un concept primitif : $A \ominus_{He} B = \top$ Cas 2 : $A = A_1 \sqcap ... \sqcap A_n$ $A \ominus_{He} B = \bigsqcap_{i=1}^{n}(A_i \ominus_{He} B_{\prec})$ Cas 3 : $A = \exists r.A_1$ et 3.a : $B_{\prec} = \exists r.B_1$ et $A_1 \ominus_{He} B_1 \not\equiv \top$ $A \ominus_{He} B = \exists r.(A_1 \ominus_{He} B_1)$ 3.b : $B_{\prec} = \exists r.B_1$ et $A_1 \ominus_{He} B_1 \equiv \top$ $A \ominus_{He} B = \top$ 3.c : $B \equiv \top$: $A \ominus_{He} B = \top$ 3.d : sinon : $A \ominus_{He} B = A$	S	B	–	O	• Voir Heinz (2018) et A.4 (p.120) • A_i n'est pas une conjonction, $\forall i$ • $B_{\prec}$ est le plus petit conjonct de B par rapport à $\prec$ • A et B sont normalisés et entièrement étendus.
$A \ominus_{\mathcal{T}} B$ l'opérateur que l'on propose dans cette thèse, voir la section 3.1.1 (p.46)	S	B	–	O	• A et B sont dans $\mathcal{EL}$. • Soient deux concepts primitifs C et D, $C \ominus_{\mathcal{T}} D = \top$ si et seulement si $D \sqsubseteq C$. On ne retire des concepts du diminuende que si ces concepts subsument un ou des concepts du diminuteur.

Annexe B

Détails relatifs à l'implémentation

B.1 Algorithmes implémentés

B.1.1 Normalisation

Les algorithmes sont ceux de l'implémentation proposée dans la section 5.3 (p.75).

Algorithm 8 preNorm(), qui transforme les axiomes d'équivalence en deux axiomes de subsomptions.

Require: Une $\mathcal{EL}$-TBox $\mathcal{T}$ générale.
Ensure: Une $\mathcal{EL}$-TBox $\mathcal{T}$ générale sans axiomes définitionnels.

1: **for all** Axiomes Ax de $\mathcal{T}$ **do**
2: **if** Ax de la forme $C \equiv D$ **then**
3: Ax devient $C \sqsubseteq D$.
4: Ajout de l'axiome $D \sqsubseteq C$ à $\mathcal{T}$.
5: **end if**
6: **end for**
7: renvoyer la TBox

Algorithm 9 postNorm(), qui renomme les restrictions existentielles
B est un nom de concept de $\mathcal{T}$, C est un concept de $\mathcal{T}$, r est un rôle de $\mathcal{T}$, est un concept de $\mathcal{T}$.

Require: Une TBox $\mathcal{T}$ quasi-normalisée.
Ensure: La TBox $\mathcal{T}$ normalisée selon Baader et al. avec renommage des restrictions existentielles.

for all Axiomes Ax de $\mathcal{T}$ **do**
  if Ax de la forme $C \sqsubseteq \exists r.B$ **then**
    Ax devient $C \sqsubseteq R\_r.B$.
    Ajout de l'axiome $R\_r.B \sqsubseteq \exists r.B$ à $\mathcal{T}$ si l'axiome n'existe pas déjà.
    Ajout de l'axiome $\exists r.B \sqsubseteq R\_r.B$ à $\mathcal{T}$ si l'axiome n'existe pas déjà.
  end if
  if Ax de la forme $\exists r.B \sqsubseteq C$ **then**
    Ax devient $R\_r.B \sqsubseteq C$.
    Ajout de l'axiome $R\_r.B \sqsubseteq \exists r.B$ à $\mathcal{T}$ si l'axiome n'existe pas déjà.
    Ajout de l'axiome $\exists r.B \sqsubseteq R\_r.B$ à $\mathcal{T}$ si l'axiome n'existe pas déjà.
  end if
end for
Renvoyer la TBox

Algorithm 10 norm() : normalisation selon Baader et al. (2005) à l'exception des axiomes de type $A \sqcap B \sqsubseteq \exists r.E$
$\widehat{C}$ et $\widehat{D}$ sont des concepts qui ne peuvent pas être des noms de concept et E et F sont des concepts de $\mathcal{T}$, r est un rôle de $\mathcal{T}$, A et B sont des noms de concept de $\mathcal{T}$ ou $\top$, X_i un nouveau nom de concept non présent dans $\mathcal{T}$ initialement avec $\_i$ un nombre. On rappelle qu'un axiome est en forme normale s'il est de la forme $A \sqsubseteq B$, $A \sqcap B \sqsubseteq B$, $A \sqsubseteq \exists r.B$ ou $\exists r.A \sqsubseteq B$.

Require: Une $\mathcal{EL}$-TBox $\mathcal{T}$ générale sans axiomes définitionnels.
Ensure: $\mathcal{T}$ quasi normalisée.

for all Axiomes Ax de $\mathcal{T}$ **do**
  while Ax n'est pas en forme normale **do**
    if Ax de la forme $\widehat{C} \sqcap E \sqsubseteq F$ ou de la forme $E \sqcap \widehat{C} \sqsubseteq F$ **then**
      Ax devient $X\_i \sqcap E \sqsubseteq F$.
      Ajout de l'axiome $\widehat{C} \sqsubseteq X\_i$ à $\mathcal{T}$, avec X_i non présent dans $\mathcal{T}$.
    else if Ax de la forme $\exists r.\widehat{C} \sqsubseteq E$ **then**
      Ax devient $\exists r.X\_i \sqsubseteq E$.
      Ajout de l'axiome $\widehat{C} \sqsubseteq X\_i$ à $\mathcal{T}$, avec X_i non présent dans $\mathcal{T}$.
    else if Ax de la forme $\exists r.E \sqsubseteq \widehat{C}$ **then**
      Ax devient $\exists r.E \sqsubseteq X\_i$.
      Ajout de l'axiome $X\_i \sqsubseteq \widehat{C}$ à $\mathcal{T}$, avec X_i non présent dans $\mathcal{T}$.
    else if Ax de la forme $E \sqsubseteq \exists r.\widehat{C}$ **then**
      Ax devient $E \sqsubseteq \exists r.X\_i$.
      Ajout de l'axiome $X\_i \sqsubseteq \widehat{C}$ à $\mathcal{T}$, avec X_i non présent dans $\mathcal{T}$.
    else if Ax de la forme $B \sqsubseteq E \sqcap F$ **then**
      Ax devient $B \sqsubseteq E$.
      Ajout de l'axiome $B \sqsubseteq F$ à $\mathcal{T}$.
    end if
  end while
end for
Renvoyer la TBox

B.1.2 Classification

Les algorithmes sont ceux de l'implémentation proposée dans la section 5.4 (p.79).

Algorithm 11 ontoHash()
r est un rôle de $\mathcal{T}$, B est un nom de concept de $\mathcal{T}$.

Require: Une TBox $\mathcal{T}$ normalisée selon Baader et al. avec ses axiomes contenant des restrictions existentielles renommées pour le fill-up.
Ensure: La table de hachage de la classification initialisée.

1: **for all** Axiomes Ax de $\mathcal{T}$ **do**
2: **if** Si Ax a un nom de concept B a gauche **then**
3: **if** Si B n'est pas déjà dans la table de hachage **then**
4: On ajoute $\mathbf{S}(B) => B$ à la table de hachage.
5: **end if**
6: **end if**
7: **if** Si Ax a un nom de concept B en second membre d'une conjonction ou a droite d'une subsomption **then**
8: **if** Si B n'est pas déjà dans la table de hachage **then**
9: On ajoute $\mathbf{S}(B) => B$ à la table de hachage.
10: **end if**
11: **end if**
12: **if** Si Ax a un une restriction existentielle $\exists r.B$ a gauche **then**
13: **if** Si B n'est pas déjà dans la table de hachage **then**
14: On ajoute $\mathbf{S}(B) => B$ à la table de hachage.
15: **end if**
16: **end if**
17: **if** Si Ax a un une restriction existentielle $\exists r.B$ a droite **then**
18: **if** Si B n'est pas déjà dans la table de hachage **then**
19: On ajoute $\mathbf{S}(B) => B$ à la table de hachage.
20: **end if**
21: **end if**
22: **end for**
23: On ajoute une clé "rôle" vide à la fin de la table de hachage.
24: Renvoyer la table de hachage

Algorithm 12 classiBaader(), qui calcule la classification selon Baader et al. (2005)
C est un concept de $\mathcal{T}$, r est un rôle de $\mathcal{T}$ et B est un nom de concept de $\mathcal{T}$.

Require: Une TBox $\mathcal{T}$ normalisée selon Baader et al. avec ses axiomes contenant des restrictions existentielles renommées pour le fill-up et la table de hachage initialisée correspondante.
Ensure: La table de hachage contenant la classification de la TBox $\mathcal{T}$.

for all Axiomes Ax de $\mathcal{T}$ **do**
  if Ax de la forme $B \sqsubseteq \exists r.C$ **then**
    if $(B, C) \notin \mathbf{R}(r)$ **then**
      $\mathbf{R}(r) := \mathbf{R}(r) \cup (B, C)$
      Suppression de Ax de la TBox.
    end if
  end if
  if Ax de la forme $B \sqsubseteq C$ **then**
    if $C \notin \mathbf{S}(B)$ **then**
      $\mathbf{S}(B) := \mathbf{S}(B) \cup C$ et $\mathbf{S^*}(C) := \mathbf{S^*}(C) \cup B$
      Suppression de Ax de la TBox.
    end if
  end if
end for
Initialisation du tableau ModTab des ensembles $S(B)$ modifiés, contenant toutes les clés pour un premier parcours complet.
while ModTab n'est pas vide **do**
  ModIteration est initialisé vide.
  for all Concepts I de ModTab **do**
    for all $\mathbf{S}(B)$ où $B \in \mathbf{S^*}(I)$ **do**
      for all Concept $C \in S(I)$ **do**
        if $C \notin \mathbf{S}(B)$ **then**
          $\mathbf{S}(B) := \mathbf{S}(B) \cup C$ et $\mathbf{S^*}(C) := \mathbf{S^*}(C) \cup B$
          if $B \notin$ ModIteration **then**
            ModIteration :=ModIteration $\cup B$
          end if
        end if
      end for
    end for
    for all Axiomes Ax de la TBox **do**
      if Ax est de la forme $B \sqcap C \sqsubseteq D$ **then**
        if $B \in \mathbf{S}(I)$ et $C \in \mathbf{S}(I)$ et $D \notin \mathbf{S}(I)$ **then**
          $\mathbf{S}(I) := \mathbf{S}(I) \cup D$ et $\mathbf{S^*}(D) := \mathbf{S^*}(D) \cup I$
          if $I \notin$ ModIteration **then**
            ModIteration :=ModIteration $\cup I$
          end if
        end if
      else if Ax est de la forme $\exists r.B \sqsubseteq C$ **then**
        if $B \in \mathbf{S}(I)$ **then**
          for all Couples $(C1, C2)$ de la clé B de la table de hachage rôle **do**
            if $B = C2$ et $C \notin \mathbf{S}(C1)$ **then**
              $S(C1) := \mathbf{S}(C1) \cup C$ et $\mathbf{S^*}(C) := \mathbf{S^*}(C) \cup C1$
              if $C1 \notin$ ModIteration **then**
                ModIteration :=ModIteration $\cup C1$
              end if
            end if
          end for
        end if
      end if
    end for
  end for
  ModTab :=ModIteration
end while
Renvoyer la table de hachage

B.1.3 Fill-up

L'algorithme est celui de l'implémentation proposée dans la section 5.5 (p.82).

Algorithm 13 fup(), qui calcule le fill-up d'un concept à partir d'une classification d'une TBox normalisée avec ses restrictions existentielles renommées. r est un rôle de $\mathcal{T}$, B est un concept nom de concept de $\mathcal{T}$. L'algorithme est simplifié par rapport à l'implémentation réel car l'implémentation prend en compte des cas particuliers sur le format des concepts, qui appliquent cependant la même transformation que la ligne 4 ou la ligne 6.

Require: La classification d'une TBox normalisée $\mathcal{T}$ selon Baader et al. avec ses axiomes contenant des restrictions existentielles renommées pour le fill-up et contenant la définition de C. C un concept.
Ensure: Le fill-up de C.

```
1: Result := C
2: for all Concepts A de S(C) dans la classification do
3:    if A n'est pas de la forme R_desc then
4:       Résultat := Résultat ⊓A
5:    else
6:       Résultat := Résultat⊓∃role.fup(desc)
7:    end if
8: end for
9: Renvoyer Résultat
```

B.1.4 TSO

L'algorithme est celui de l'implémentation proposée dans la section 5.6 (p.83).

Algorithm 14 tso(), qui calcule la différence syntaxique de deux concepts. La capacité de recherche dans un tableau de l'implémentation permet de s'affranchir des appels récursifs nécessaires pour ramener le tso() à une comparaison de deux noms de concept ou deux restrictions existentielles.

Require: Deux concepts C et D.
Ensure: La différence syntaxique de C et D.
1: **for all** Conjoncts C_i de C **do**
2: **if** C_i est un nom de concept **then**
3: **if** D possède le même nom de concept **then**
4: **if** C est une conjonction d'un seul concept **then**
5: $C_i = \top$
6: **else if** C_i est une conjonction de plusieurs concepts **then**
7: On retire A de C_i.
8: **end if**
9: **end if**
10: **else if** C_i est une restriction existentielle, de la forme $\exists r.E$ **then**
11: **if** B possède la même restriction existentielle, de la forme $\exists r.F$ **then**
12: E = TSO(E,F)
13: **if** E=$\top$ **then**
14: A=$\top$
15: **end if**
16: **end if**
17: **end if**
18: **end for**
19: Renvoyer C

B.1.5 CCO

Les algorithmes sont ceux de l'implémentation proposée dans la section 5.7 (p.84).

Algorithm 15 componentComparison() qui détermine la relation de subsomption entre une demande et les réponses pour chaque composante. "fm" signifie full miss (la demande ne possède pas la composante en cours) et "fr" signifie full rest (la réponse ne répond à aucune partie de la demande). Pour la composante c signifie la projection sur la composante c d'un concept.

Require: Une demande Dem, l'ensemble Rep_i des réponses possibles, la classification de la TBox contenant les projections sur chaque composante de Dem et des Rep_i, le tableau des composantes CompTab.

Ensure: Le hachage contenant les relations de subsomptions des réponses avec la demande pour chaque composante, hashComp.

for all Composantes de CompTab c **do**
  if Dem est existante dans c **then**
    for all Réponses Rep_i **do**
      if Rep_i possède la composante **then**
        if Rep_i et Dem n'ont pas $\top$ pour c **then**
          if $Rep_i \equiv Dem$ pour c **then**
            hashComp[c]<={Rep_i, "equiv"}
          else if $Rep_i \sqsubseteq Dem$ pour c **then**
            hashComp[c]<={Rep_i, "subsuming"}
          else if $Dem \sqsubseteq Rep_i$ pour c **then**
            hashComp[c]<={Rep_i, "subsumed"}
          else
            hashComp[c]<={Rep_i, "no relation"}
          end if
        else if $Rep_i = Dem = \top$ pour c **then**
          hashComp[composante] <= {$demande_i$, "equiv"}
        else if $Rep_i \neq \top$ et $Dem = \top$ pour c **then**
          hashComp[c]<={Rep_i, "fm"}
        else
          hashComp[c]<={Rep_i, "fr"}
        end if
      else
        hashComp[c]<={Rep_i, "fr"}
      end if
    end for
  else{(Dem est inexistante dans c)}
    for all Réponses Rep_i **do**
      if Rep_i possède la composante **then**
        hashComp[c]<={Rep_i, "fm"}
      else
        hashComp[c]<={Rep_i, "equiv"}
      end if
    end for
  end if
end for
Renvoyer hashComp

Algorithm 16 compRest() qui détermine la relation de subsomption entre deux rest compMiss() est structurellement similaire mais le diminuteur et le diminuende sont inversé dans les calculs de tso.

Require: Le fill-up de la demande Dem, les fill-up des deux réponses Rep_1 et Rep_2.

Ensure: La relation de subsomption entre le rest (ou miss) de deux réponses par rapport à une demande.

1: Dif_1 := tso(Dem, Rep_1) (pour compMiss() Dif_1 := tso(Rep_1,Dem))
2: Dif_2 := tso(Dem, Rep_2) (pour compMiss() Dif_2 := tso(Rep_2,Dem)
3: **if** Dif_1=$\top$ et Dif_2=$\top$ **then**
4: Relation := "equiv"
5: **else if** Dif_1=$\top$ **then**
6: Relation := "subsuming" (c'est-à-dire que $Dif_2 \sqsubset Dif_1$)
7: **else if** Dif_2=$\top$ **then**
8: Relation := "subsumed" (c'est-à-dire que $Dif_1 \sqsubset Dif_2$)
9: **else**
10: Relation := comparaisonComposanteMissRest(Dif_1, Dif_2)
11: **end if**
12: Renvoyer Relation

Algorithm 17 comparaisonComposanteMissRest() qui calcule la relation de subsomption entre deux rest (ou miss) en calculant la classification de la TBox initiale à laquelle on a ajouté les axiomes définissant les deux rests (ou miss).

Require: Deux rest (ou miss) $Diff_1$ et $Diff_2$, la TBox initiale $\mathcal{T}_{ini}$, le hachage de la classification initiale.

Ensure: La relation de subsomption entre le rest (ou miss) de deux réponses par rapport à une demande.

1: On effectue la classification avec Dif_1 et Dif_2 dans la TBox initiale.
2: **if** La classification de Dif_1 contient Dif_2 et la classification de Dif_2 contient Dif_1 **then**
3: Relation := "equiv"
4: **else if** La classification de Dif_2 contient Dif_1 **then**
5: Relation := "subsuming" (c'est-à-dire que $Dif_2 \sqsubset Dif_1$)
6: **else if** La classification de Dif_1 contient $Rest_2$ **then**
7: Relation := "subsumed" (c'est-à-dire que $Dif_1 \sqsubset Dif_2$)
8: **else**
9: Relation := "no relation"
10: **end if**
11: Renvoyer Relation

B.2 Preuves liées à l'implémentations

Les preuves de cette section sont liées à l'implémentation de la normalisation (voir section 5.3 (p.75)) et de la classification (voir section 5.4 (p.79)).

B.2.1 Conséquence de postNorm() sur la règle CR3b

On veut montrer que l'application des renommages des restrictions existentielles par postNorm() permet de ne pas avoir à utiliser la règle CR3b (voir 5.5) lors de la classification.

On rappelle la règle CR3b : Si C' $\in \mathbf{S}(C)$, $(C', D) \in \mathbf{R}(r)$ et $(C, D) \notin \mathbf{R}(r)$ alors $\mathbf{R}(r) := \mathbf{R}(r) \cup \{(C, D)\}$. postNorm() (algorithme 9 (p.128)) renomme toutes les restrictions existentielles d'une TBox normalisée. Ainsi, chaque restriction existentielle est remplacée par un nom de concept dans les axiomes existants, et deux axiomes (par restriction existentielle) sont ajoutés à la TBox pour effectuer ce renommage. Par exemple, l'axiome $A \sqsubseteq \exists r.B$ est changé en $A \sqsubseteq R\_r.B$ et les axiomes $\exists r.B \sqsubseteq R\_r.B$ $R\_r.B \sqsubseteq \exists r.B$ sont ajoutés à la TBox. Par conséquent, lors de l'itération initiale de l'algorithme 12 (p.130), tous les axiomes de type $R\_r.B \sqsubseteq \exists r.B$ seront supprimés et ajouté à l'ensemble R correspondant. La règle CR3b, qui permet d'expliciter les rapports de subsomption entre un concept et une restriction existentielle, n'a alors plus la nécessité d'être appliquée car la présence d'un $R\_r.B$ dans un ensemble $\mathbf{S}(C)$ implique immédiatement $A \sqsubseteq \exists r.B$ et il n'est pas nécessaire d'ajouter (C, B) à l'ensemble $R(r)$. La règle CR3b n'a donc pas besoin d'être appliquée dans l'algorithme 12 (p.130).

B.2.2 Preuves liées à l'implémentation itérative des règles de classification

On veut montrer qu'il n'est pas nécessaire de tester l'intégralité des ensembles **S** lors d'une itération du calcul de la classification et que l'on peut ignorer les ensembles **S** qui n'ont pas été modifié à l'itération précédente :

- CR1b : Si $C' \in \mathbf{S}(C)$, $D \in \mathbf{S}(C')$ et $D \notin \mathbf{S}(C)$ alors $\mathbf{S}(C) := \mathbf{S}(C) \cup \{D\}$ Si aucun nouveau C' n'a été ajouté à $\mathbf{S}(C)$ à l'itération précédente, les conditions d'application de la règle n'ont pas changé et on ne pourra pas l'appliquer à cette itération.
- CR2 : Si $C1$, $C2 \in \mathbf{S}(C)$, $C1 \sqcap C2 \sqsubseteq D \in \mathcal{T}$ et $D \notin \mathbf{S}(C)$ alors $\mathbf{S}(C) := \mathbf{S}(C) \cup \{D\}$. Si aucun nouveau $C1$ ou $C2$ n'ont été ajouté à $\mathbf{S}(C)$ à l'itération précédente, les conditions d'application de la règle n'ont pas changé et on ne pourra pas l'appliquer à cette itération.
- CR4 : Si $(C, D) \in \mathbf{R}(r)$, $D' \in \mathbf{S}(D)$, $\exists\ r.D' \sqsubseteq E \in \mathcal{T}$, et $E \notin \mathbf{S}(C)$ alors $\mathbf{S}$(C) :=$\mathbf{S}$(C) $\cup$ {E}. Si aucun nouveau D' n'a été ajouté à $\mathbf{S}(D)$ à l'itération précédente et si aucun nouveau couple (C, D) n'a été ajouté à $\mathbf{R}(r)$ à l'itération précédente, les conditions d'application de la règle n'ont pas changé et on ne pourra pas l'appliquer à cette itération. Notons cependant qu'une modification dans l'un de ces deux mappings est une condition suffisante à l'exécution de la

règle CR4. En ce cas, il faut chercher dans chaque $\mathbf{R}(r)$ un couple (C, D) satisfaisant les conditions en cas d'ajout de D' à $\mathbf{S}(D)$ à l'itération précédente. ou chercher dans l'ensemble des $\mathbf{S}(C)$ un concept D' satisfaisant les conditions en cas d'ajout de (C, D) à $\mathbf{R}(r)$ à l'itération précédente.

B.3 Exemples illustrant l'implémentation

B.3.1 Exemple de représentation de données

Cette section contient les exemples de représentations d'axiomes (exemple 39) et d'une classification (exemple 40), dont les principes sont donnés dans la section 5.1 (p.73).

Exemple 39. *A gauche, les axiomes de $\mathcal{T}_{ex}$, la TBox $\mathcal{EL}$ de métrologie de l'exemple 1 (p.4) et à droite sa représentation dans l'implémentation.*

$Bois \sqsubseteq Matériau$,	`["Bois", "subs", "Matériau"],`
$Chêne \sqsubseteq Bois$,	`["Chêne", "subs", "Bois"],`
$Métal \sqsubseteq Matériau$,	`["Métal", "subs", "Matériau"],`
$Plastique \sqsubseteq Matériau$,	`["Plastique", "subs", "Matériau"],`
$Fer \sqsubseteq Métal$,	`["Fer", "subs", "Métal"],`
$Acier \sqsubseteq Métal$,	`["Acier", "subs", "Métal"],`
$PaC \sqsubseteq Type$,	`["PaC", "subs", "Type"]`
$Micromètre \sqsubseteq PaC$,	`["Micromètre", "subs", "PaC"],`
$Règle \sqsubseteq Type$,	`["Règle", "subs", "Type"],`
$Fab - Européen \sqsubseteq Fabricant$,	`["Fab-Européen", "subs", "Fabricant"],`
$Fab - Américain \sqsubseteq Fabricant$,	`["Fab-Américain", "subs", "Fabricant"],`
$Fab - Français \sqsubseteq Fab - Européen$,	`["Fab-Français", "subs", "Fab-Européen"],`
$Fab - Allemand \sqsubseteq Fab - Européen$,	`["Fab-Allemand", "subs", "Fab-Européen"],`
$Fab - Canadien \sqsubseteq Fab - Américain$,	`["Fab-Canadien", "subs", Fab-Américain"],`
$Mm \sqsubseteq Unité$,	`["Mm", "subs", "Unité"],`
$Cm \sqsubseteq Unité$,	`["Cm", "subs", "Unité"],`
$Analogique \sqsubseteq ModeLecture$,	`["Analogique", "subs", "ModeLecture"],`
$Numérique \sqsubseteq ModeLecture$.	`["Numérique", "subs", "ModeLecture"]`

Exemple 40. *A gauche, les mappings obtenus après classification de la TBox d'exemple et à droite leur représentation dans l'implémentation, à l'intérieur de la table de hachage.*

$S(Bois) = \{Bois, Matériau, \top\}$
`"Bois"=>["Bois", "Matériau"]`
$S(Chêne) = \{Chêne, Bois, Matériau, \top\}$
`"Chêne"=>["Chêne", "Bois", "Matériau"]`
$S(Métal) = \{Métal, Matériau, \top\}$
`"Métal"=>["Métal", "Matériau"]`

$S(Plastique) = \{Plastique, Matériau, \top\}$
`"Plastique"=>["Plastique", "Matériau"]`
$S(Fer) = \{Fer, Métal, Matériau, \top\}$
`"Fer"=>["Fer", "Métal", "Matériau"]`
$S(Acier) = \{Acier, Métal, Matériau, \top\}$
`"Acier"=>["Acier", "Métal", "Matériau"]`
$S(PiedaCoulisse) = \{PiedaCoulisse, Type, \top\}$
`"Pied a Coulisse"=>["Pied a Coulisse", "Type"]`
$S(Micromètre) = \{Micromètre, PiedaCoulisse, Type, \top\}$
`"Micromètre"=>["Micromètre", "PiedaCoulisse", "Type"]`
$S(Règle) = \{Règle, Type, \top\}$
`"Règle"=>["Règle", "Type"]`
$S(Fab - Européen) = \{Fab - Européen, Fabricant, \top\}$
`"Fab-Européen"=>["Fab-Européen", "Fabricant"]`
$S(Fab - Américain) = \{Fab - Américain, Fabricant, \top\}$
`Fab-Américain"=>["Fab-Américain", "Fabricant"]`
$S(Fab - Français) = \{Fab - Français, Fab - Européen, Fabricant, \top\}$
`"Fab-Français"=>["Fab-Français", "Fab-Européen", "Fabricant"]`
$S(Fab - Allemand) = \{Fab - Allemand, Fab - Européen, Fabricant, \top\}$
`"Fab-Allemand"=>["Fab-Allemand", "Fab-Européen", "Fabricant"]`
$S(Fab - Canadien) = \{Fab - Canadien, Fab - Américain, \top\}$
`"Fab-Canadien"=>["Fab-Canadien", "Fab-Américain", "Fabricant"]`
$S(Mm) = \{Mm, Unité, \top\}$
`"Mm"=>["Mm", "Unité"]`
$S(Cm) = \{Cm, Unité, \top\}$
`"Cm"=>["Cm", "Unité"]`
$S(Analogique) = \{Analogique, ModeLecture, \top\}$
`"Analogique"=>["Analogique", "ModeLecture"]`
$S(Numérique) = \{Numérique, ModeLecture, \top\}$
`"Numérique"=>["Numérique", "ModeLecture"]`
$S(Matériau) = \{Matériau, \top\}$
`"Matériau"=>["Matériau"]`
$S(Fabricant) = \{Fabricant, \top\}$
`"Fabricant"=>["Fabricant"]`
$S(Type) = \{Type, \top\}$
`"Type"=>["Type"]`
$S(Unité) = \{Unité, \top\}$
`"Unité"=>["Unité"]`
$S(ModeLecture) = \{ModeLecture, \top\}$
`"ModeLecture"=>["ModeLecture"]`

B.3.2 Exemple de normalisation

L'exemple de cette section est une exécution de l'implémentation de la normalisation présentée dans la section 5.3 (p.75).

Exemple 41. *Soit la TBox $\mathcal{T}$ contenant les axiomes suivants :*
$A \sqcap \exists r.B \sqcap C \sqsubseteq D \sqcap \exists r.E$, $F \equiv G$.

preNorm() transforme la TBox de manière suivante :

— $A \sqcap \exists r.B \sqcap C \sqsubseteq D \sqcap \exists r.E$, $F \sqsubseteq G$, $G \sqsubseteq F$.

norm() transforme la TBox de manière suivante :

— *Itération 1, Règle NR1 :* $A \sqcap C \sqcap X_1 \sqsubseteq D \sqcap \exists r.E$, $F \sqsubseteq G$, $G \sqsubseteq F$, $\exists r.B \sqsubseteq X_1$

— *Itération 2, Règle NR1 :*$A \sqcap X_2 \sqsubseteq D \sqcap \exists r.E$, $F \sqsubseteq G$, $G \sqsubseteq F$, $\exists r.B \sqsubseteq X_1$, $C \sqcap X_1 \sqsubseteq X_2$

— *Itération 3, Règle NR5 :* $A \sqcap X_2 \sqsubseteq D$, $F \sqsubseteq G$, $G \sqsubseteq F$, $\exists r.B \sqsubseteq X_1$, $C \sqcap X_1 \sqsubseteq X_2$, $A \sqcap X_2 \sqsubseteq \exists r.E$

postNorm() transforme la TBox de manière suivante :

— $A \sqcap X_2 \sqsubseteq D$, $F \sqsubseteq G$, $G \sqsubseteq F$, $R\_r.B \sqsubseteq X_1$, $C \sqcap X_1 \sqsubseteq X_2$, $A \sqcap X_2 \sqsubseteq R\_r.E$, $R\_r.B \sqsubseteq \exists r.B$, $\exists r.B \sqsubseteq R\_r.B$, $R\_r.E \sqsubseteq \exists r.E$, $\exists r.E \sqsubseteq R\_r.E$

Dans le cas de l'application stricte des règles du tableau 5.3 (p.78) on aurait pour normalisation :

— *Itération 1, Règle NR1 :* $A \sqcap C \sqcap X_1 \sqsubseteq D \sqcap \exists r.E$, $F \sqsubseteq G$, $G \sqsubseteq F$, $\exists r.B \sqsubseteq X_1$

— *Itération 2, Règle NR1 :*$A \sqcap X_2 \sqsubseteq D \sqcap \exists r.E$, $F \sqsubseteq G$, $G \sqsubseteq F$, $\exists r.B \sqsubseteq X_1$, $C \sqcap X_1 \sqsubseteq X_2$

— *Itération 3, Règle NR3 :* $A \sqcap X_2 \sqsubseteq X_3$, $F \sqsubseteq G$, $G \sqsubseteq F$, $\exists r.B \sqsubseteq X_1$, $C \sqcap X_1 \sqsubseteq X_2$, $X_3 \sqsubseteq D \sqcap \exists r.E$

— *Itération 4, Règle NR5 :* $A \sqcap X_2 \sqsubseteq X_3$, $F \sqsubseteq G$, $G \sqsubseteq F$, $\exists r.B \sqsubseteq X_1$, $C \sqcap X_1 \sqsubseteq X_2$, $X_3 \sqsubseteq D$,$X_3 \sqsubseteq \exists r.E$

postNorm() transforme la TBox de manière suivante :

— $A \sqcap X_2 \sqsubseteq X_3$, $F \sqsubseteq G$, $G \sqsubseteq F$, $R\_r.B \sqsubseteq X_1$, $C \sqcap X_1 \sqsubseteq X_2$, $X_3 \sqsubseteq D, X_3 \sqsubseteq R\_r.E$, $\exists r.B \sqsubseteq R\_r.B$, $R\_r.B \sqsubseteq \exists r.B$, $\exists r.E \sqsubseteq R\_r.E$, $R\_r.E \sqsubseteq \exists r.E$

B.3.3 Exemple de classification

L'exemple de cette section est une exécution de l'implémentation de la normalisation présentée dans la section 5.4 (p.79).

Exemple 42. *Soit la $\mathcal{T}$ la TBox suivante :*
$\{A \sqsubseteq B, B \sqcap A \sqsubseteq C, C \sqsubseteq \exists r.D, \exists r.D \sqsubseteq E, E \sqsubseteq F, E \sqsubseteq G, B \sqcap G \sqsubseteq H\}$
On rappelle que dans l'implémentation, la classification est représentée sous la forme d'une table de hachage dont chaque clé est un ensemble ***S****. Pour les rôles, les ensembles* ***R*** *sont également stockés dans une seconde table de hache stockée dans la clé "Role" de la classification. ontoHash() réalise les tables de hachages suivantes (chaque clé est représentée individuellement pour plus de clarté. De la même manière, $\top$ est omis car implicitement présent dans chaque ensemble* ***S****) :*

S*(A) =>* $\{A\}$
S*(B) =>* $\{B\}$
S*(C) =>* $\{C\}$
S*(D) =>* $\{D\}$
S*(E) =>* $\{E\}$
S*(F) =>* $\{F\}$
S*(G) =>* $\{G\}$
S*(H) =>* $\{H\}$
S*(Role) =>* $\{\}$

On réalise ensuite la classification et on crée ***S*****(X) la table de hachage contenant tous les concepts subsumés par X :*

Itération initiale (ligne 1 à 16 de l'algorithme 12 (p.130)), qui effectue toutes les CR1a et CR3a possibles :

S*(A) =>* $\{A, B\}$,
S*(B) =>* $\{B\}$,
S*(C) =>* $\{C\}$,
S*(D) =>* $\{D\}$,
S*(E) =>* $\{E, F, G\}$,
S*(F) =>* $\{F\}$,
S*(G) =>* $\{G\}$,
S*(H) =>* $\{H\}$,
S*(Role) =>* $\{\boldsymbol{R}(r) => \{(C, D)\}\}$

S**(A) =>* $\{\}$
S**(B) =>* $\{A\}$
S**(C) =>* $\{\}$
S**(D) =>* $\{\}$
S**(E) =>* $\{\}$
S**(F) =>* $\{E\}$
S**(G) =>* $\{E\}$
S**(H) =>* $\{\}$

et on a retiré de la TBox les axiomes utilisés.
$\mathcal{T} = \{B \sqcap A \sqsubseteq C, \exists r.D \sqsubseteq E, B \sqcap G \sqsubseteq H\}$
Le tableau ModTab est créé et contient tous les concepts (de A à H).

Première itération (ligne 18 à 54) :
On initialise ModIteration à vide et on parcourt ModTab On ajoute C à $\boldsymbol{S}(A)$ *(CR2) et on ajoute A à ModIteration, on ajoute E à* $\boldsymbol{S}(C)$ *(CR4) et on ajoute C à ModIteration.*

$\boldsymbol{S}(A) => \{A, B, C\}$, $\boldsymbol{S*}(A) => \{\}$
$\boldsymbol{S}(B) => \{B\}$, $\boldsymbol{S*}(B) => \{A\}$
$\boldsymbol{S}(C) => \{C, E\}$, $\boldsymbol{S*}(C) => \{A\}$
$\boldsymbol{S}(D) => \{D\}$, $\boldsymbol{S*}(D) => \{\}$
$\boldsymbol{S}(E) => \{E, F, G\}$, $\boldsymbol{S*}(E) => \{C\}$
$\boldsymbol{S}(F) => \{F\}$, $\boldsymbol{S*}(F) => \{E\}$
$\boldsymbol{S}(G) => \{G\}$, $\boldsymbol{S*}(G) => \{E\}$
$\boldsymbol{S}(H) => \{H\}$, $\boldsymbol{S*}(H) => \{\}$
$\boldsymbol{S}(Role) => \{\boldsymbol{R}(r) => \{(C, D)\}\}$

ModTab devient ModIteration

Seconde itération :
On initialise ModIteration à vide et on parcourt ModTab.
$\boldsymbol{S}(A)$ *et* $\boldsymbol{S}(C)$ *ont été modifiés, on doit donc modifier tous les ensemble* $\boldsymbol{S}(X)$ *contenant A ou C pour les compléter et vérifier si de nouvelles règles peuvent être appliquées dans* $\boldsymbol{S}(A)$ *et* $\boldsymbol{S}(C)$ *:*
On ajoute E à $\boldsymbol{S}(A)$ *(CR1b), on ajoute F et G à C (CR1b). On ajoute A et C à ModIteration.*

$\boldsymbol{S}(A) => \{A, B, C, E\}$, $\boldsymbol{S*}(A) => \{\}$
$\boldsymbol{S}(B) => \{B\}$, $\boldsymbol{S*}(B) => \{A\}$
$\boldsymbol{S}(C) => \{C, E, F, G\}$, $\boldsymbol{S*}(C) => \{A\}$
$\boldsymbol{S}(D) => \{D\}$, $\boldsymbol{S*}(D) => \{\}$
$\boldsymbol{S}(E) => \{E, F, G\}$, $\boldsymbol{S*}(E) => \{C, A\}$
$\boldsymbol{S}(F) => \{F\}$, $\boldsymbol{S*}(F) => \{E, C\}$
$\boldsymbol{S}(G) => \{G\}$, $\boldsymbol{S*}(G) => \{E, C\}$
$\boldsymbol{S}(H) => \{H\}$, $\boldsymbol{S*}(H) => \{\}$
$\boldsymbol{S}(Role) => \{\boldsymbol{R}(r) => \{(C, D)\}\}$

ModTab devient ModIteration

Troisième itération :
On initialise ModIteration à vide et on parcourt ModTab.
$\boldsymbol{S}(A)$ *et* $\boldsymbol{S}(C)$ *ont été modifiés, on doit donc modifier tous les ensemble* $\boldsymbol{S}(X)$ *contenant A ou C pour les compléter et vérifier si de nouvelles règles peuvent être appliquées dans*

***S**(A) et **S**(C) :*
*On ajoute F et G à **S**(A) (CR1b) et on ajoute A à ModInteration*

$\boldsymbol{S}(A) => \{A, B, C, E, F, G\}$, $\boldsymbol{S}*(A) => \{\}$
$\boldsymbol{S}(B) => \{B\}$, $\boldsymbol{S}*(B) => \{A\}$
$\boldsymbol{S}(C) => \{C, E, F, G\}$, $\boldsymbol{S}*(C) => \{A\}$
$\boldsymbol{S}(D) => \{D\}$, $\boldsymbol{S}*(D) => \{\}$
$\boldsymbol{S}(E) => \{E, F, G\}$, $\boldsymbol{S}*(E) => \{C, A\}$
$\boldsymbol{S}(F) => \{F\}$, $\boldsymbol{S}*(F) => \{E, C, A\}$
$\boldsymbol{S}(G) => \{G\}$, $\boldsymbol{S}*(G) => \{E, C, A\}$
$\boldsymbol{S}(H) => \{H\}$, $\boldsymbol{S}*(H) => \{\}$
$\boldsymbol{S}(Role) => \{\boldsymbol{R}(r) => \{(C, D)\}\}$

ModTab devient ModIteration

Quatrième itération :
On initialise ModIteration à vide et on parcourt ModTab.
***S**(A) a été modifié, on doit donc modifier tous les ensemble **S**(X) contenant A pour les compléter et vérifier si de nouvelles règles peuvent être appliquées dans **S**(A) et **S**(C) :*
*On ajoute H à **S**(A) (CR2) et on ajoute A à ModInteration*

$\boldsymbol{S}(A) => \{A, B, C, E, F, G, H\}$, $\boldsymbol{S}*(A) => \{\}$
$\boldsymbol{S}(B) => \{B\}$, $\boldsymbol{S}*(B) => \{A\}$
$\boldsymbol{S}(C) => \{C, E, F, G\}$, $\boldsymbol{S}*(C) => \{A\}$
$\boldsymbol{S}(D) => \{D\}$, $\boldsymbol{S}*(D) => \{\}$
$\boldsymbol{S}(E) => \{E, F, G\}$, $\boldsymbol{S}*(E) => \{C, A\}$
$\boldsymbol{S}(F) => \{F\}$, $\boldsymbol{S}*(F) => \{E, C, A\}$
$\boldsymbol{S}(G) => \{G\}$, $\boldsymbol{S}*(G) => \{E, C, A\}$
$\boldsymbol{S}(H) => \{H\}$, $\boldsymbol{S}*(H) => \{A\}$
$\boldsymbol{S}(Role) => \{\boldsymbol{R}(r) => \{(C, D)\}\}$

ModTab devient ModIteration

Dernière itération :
On initialise ModIteration à vide et on parcourt ModTab.
***S**(A) a été modifié, on doit donc modifier tous les ensemble **S**(X) contenant A pour les compléter et vérifier si de nouvelles règles peuvent être appliquées dans **S**(A) et **S**(C) :*
Plus aucune règle ne peut s'appliquer, l'algorithme s'arrête et la classification finale est :

$\boldsymbol{S}(A) => \{A, B, C, E, F, G, H\}$
$\boldsymbol{S}(B) => \{B\}$
$\boldsymbol{S}(C) => \{C, E, F, G\}$
$\boldsymbol{S}(D) => \{D\}$
$\boldsymbol{S}(E) => \{E, F, G\}$
$\boldsymbol{S}(F) => \{F\}$

$\boldsymbol{S}(G) => \{G\}$
$\boldsymbol{S}(H) => \{H\}$
$\boldsymbol{S}(Role) => \{\boldsymbol{R}(r) => \{(C, D)\}\}$

B.3.4 Exemple de fill-up

L'exemple de cette section est une exécution de l'implémentation du fill-up présentée dans la section 5.5 (p.82).

Exemple 43. *Soit la TBox* $\mathcal{T} = \{A \sqsubseteq B, \exists r.A \sqsubseteq C\}$. *Elle est tout d'abord normalisée en*
$\{A \sqsubseteq B, R\_r.A \sqsubseteq C, \exists r.A \sqsubseteq R\_r.A, R\_r.A \sqsubseteq \exists r.A\}$. *Après classification, la TBox est maintenant* $\mathcal{T}* = \{\exists r.A \sqsubseteq R\_r.A\}$ *et sa classification est :*
$\boldsymbol{S}(A) => \{A, B\}$
$\boldsymbol{S}(B) => \{B\}$
$\boldsymbol{S}(C) => \{C\}$
$\boldsymbol{S}(R\_r.A) => \{R\_r.A, C\}$
$\boldsymbol{R}(r) => \{(R\_r.A, A)\}$

Soit la demande d'un utilisateur $\exists r.A$. *Elle est renommée en* Z_1 *et on ajoute les axiomes* $Z\_1 \sqcap \exists r.A$ *et* $\exists r.A \sqcap Z\_1$ *à* $\mathcal{T}*$. *Après normalisation et classification on a :*
$\boldsymbol{S}(A) => \{A, B\}$
$\boldsymbol{S}(B) => \{B\}$
$\boldsymbol{S}(C) => \{C\}$
$\boldsymbol{S}(R\_r.B) => \{R\_r.B, C\}$
$\boldsymbol{S}(R\_r.A) => \{R\_r.A, Z\_1, C\}$
$\boldsymbol{R}(r) => \{(R\_r.A, A)\}$
$\boldsymbol{S}(Z\_1) => \{Z\_1, R\_r.A, C\}$

On calcule alors le fill-up de Z_1 *:*
$Fup_{\mathcal{T}}(Z\_1) = Z\_1 \sqcap R\_r.A \sqcap C \sqcap \exists r.Fup_{\mathcal{T}}(A)$
$Fup_{\mathcal{T}}(Z\_1) = Z\_1 \sqcap R\_r.A \sqcap \exists r.(A \sqcap B)$
Z_1 *et* $R\_r.A$ *sont encore présents et ne sont pas dans la TBox* $\mathcal{T}$. *On effectue le nettoyage avec fupCleanUp() :*
$Fup_{\mathcal{T}}(Z\_1) = C \sqcap \exists r.(A \sqcap B)$

B.4 Résultats des tests qualitatifs

Dans cette section, on présente les résultats des tests qualitatifs du chapitre 6 (p.89), présentés dans la section 6.1 (p.89).

B.4.1 Normalisation

Voir 6.1.1 (p.89) pour la description des tests. Les règles de normalisation sont celles de la table 5.3 (p.78).

Test 1 (Normalisation des équivalences). *TBox :* $\{A \equiv B\}$
Normalisation attendue : $\{A \sqsubseteq B,\ B \sqsubseteq A\}$
Normalisation obtenue : $\{A \sqsubseteq B,\ B \sqsubseteq A\}$

Test 2 (Aucune transformation). *TBox :* $\{A \sqsubseteq B\}$
Normalisation attendue : $\{A \sqsubseteq B\}$
Normalisation obtenue : $\{A \sqsubseteq B\}$

Test 3 (NR5). *TBox :* $\{A \sqsubseteq B \sqcap C\}$
Normalisation attendue : $\{A \sqsubseteq B, A \sqsubseteq C\}$
Normalisation obtenue : $\{A \sqsubseteq B, A \sqsubseteq C\}$

Test 4 (NR1a avec une conjonction de plus de 2 conjoncts). *TBox :* $\{A \sqcap B \sqcap C \sqsubseteq D\}$
Normalisation attendue : $\{A \sqcap X_1 \sqsubseteq D, B \sqcap C \sqsubseteq X_1\}$
Normalisation obtenue : $\{A \sqcap X_1 \sqsubseteq D, B \sqcap C \sqsubseteq X_1\}$

Test 5 (NR1a avec un rôle en second conjonct). *TBox :* $\{A \sqcap \exists r.B \sqsubseteq C\}$
Normalisation attendue : $\{A \sqcap X_1 \sqsubseteq C, \exists r.B \sqsubseteq X_1\}$
Normalisation obtenue : $\{A \sqcap X_1 \sqsubseteq C, R\_r.B \sqsubseteq X_1, \exists r.B \sqsubseteq R\_r.B, R\_r.B \sqsubseteq \exists r.B\}$

Test 6 (NR1a et NR1b avec un rôle et une conjonction de plus de 2 conjoncts). *TBox :* $\{A \sqcap \exists r.B \sqcap D \sqsubseteq C\}$
Normalisation attendue : $\{A \sqcap X_1 \sqsubseteq D, X_2 \sqcap D \sqsubseteq X_1, \exists r.B \sqsubseteq X_2\}$
Normalisation obtenue : $\{A \sqcap X_2 \sqsubseteq D, X_1 \sqcap D \sqsubseteq X_2, R\_r.B \sqsubseteq X_1, R\_r.B \sqsubseteq \exists r.B, \exists r.B \sqsubseteq R\_r.B\}$

Test 7 (NR1b avec un rôle en premier conjonct). *TBox :* $\{\exists r.B \sqcap D \sqsubseteq C\}$
Normalisation attendue : $\{D \sqcap X_1 \sqsubseteq C, \exists r.B \sqsubseteq X_1\}$
Normalisation obtenue : $\{D \sqcap X_1 \sqsubseteq C, \exists r.BR\_r.B \sqsubseteq X_1, R\_r.B \sqsubseteq \exists r.B, \exists r.B \sqsubseteq R\_r.B\}$

Test 8 (NR2 avec une conjonction). *TBox :* $\{\exists r.(B \sqcap D) \sqsubseteq C\}$
Normalisation attendue : $\{\exists r.X_1 \sqsubseteq C, B \sqcap D \sqsubseteq X_1\}$
Normalisation obtenue : $\{R\_r.X_1 \sqsubseteq C, B \sqcap D \sqsubseteq X_1, R\_r.X_1 \sqsubseteq \exists r.X_1, \exists r.X_1 \sqsubseteq R\_r.X_1\}$

Test 9 (NR2 avec un rôle). *TBox :* $\{\exists r.\exists s.D \sqsubseteq C\}$
Normalisation attendue : $\{\exists r.X_1 \sqsubseteq C, \exists s.D \sqsubseteq X_1C\}$
Normalisation obtenue : $\{R\_r.X_1 \sqsubseteq C, R\_s.D \sqsubseteq X_1, R\_r.X_1 \sqsubseteq \exists r.X_1, \exists r.X_1 \sqsubseteq R\_r.X_1, R\_s.D \sqsubseteq \exists s.D, \exists s.D \sqsubseteq R\_s.DC\}$

Test 10 (NR5 avec un rôle). *TBox :* $\{C \sqsubseteq A \sqcap \exists r.B\}$
Normalisation attendue : $\{C \sqsubseteq A, C \sqsubseteq \exists r.B\}$
Normalisation obtenue : $\{C \sqsubseteq A, C \sqsubseteq R\_r.B, R\_r.B \sqsubseteq \exists r.B, \exists r.B \sqsubseteq R\_r.B\}$

Test 11 (NR5 avec un rôle et une conjonction de plus de deux conjoncts)**.** *TBox :* $\{C \sqsubseteq A \sqcap \exists r.B \sqcap D\}$
Normalisation attendue : $\{C \sqsubseteq A, C \sqsubseteq \exists r.B, C \sqsubseteq D\}$
Normalisation obtenue : $\{C \sqsubseteq A, C \sqsubseteq R\_r.B, C \sqsubseteq D, R\_r.B \sqsubseteq \exists r.B, \exists r.B \sqsubseteq R\_r.B\}$

Test 12 (NR4)**.** *TBox :* $\{C \sqsubseteq \exists r.(B \sqcap D)\}$
Normalisation attendue :$\{ C \sqsubseteq \exists r.X_1, X_1 \sqsubseteq B, X_1 \sqsubseteq D\}$
Normalisation obtenue : $\{C \sqsubseteq R\_r.X_1, X_1 \sqsubseteq B, X_1 \sqsubseteq D, R\_r.X_1 \sqsubseteq \exists r.X_1, \exists r.X_1 \sqsubseteq R\_r.X_1\}$

Test 13 (NR4 avec un rôle)**.** *TBox :* $\{C \sqsubseteq \exists r.\exists s.D\}$
Normalisation attendue : $\{C \sqsubseteq \exists r.X_1, X_1 \sqsubseteq \exists s.D\}$
Normalisation obtenue : $\{C \sqsubseteq R\_r.X_1, X_1 \sqsubseteq R\_s.D, R\_r.X_1 \sqsubseteq \exists r.X_1, \exists r.X_1 \sqsubseteq R\_r.X_1, R\_s.D \sqsubseteq \exists s.D, \exists s.D \sqsubseteq R\_s.D\}$

Test 14 (NR1, NR2, NR3, NR4 et NR5)**.** *TBox :* $\{C \sqcap \exists r.(A \sqcap B) \sqsubseteq \exists r.(\exists s.D \sqcap E) \sqcap F\}$
Normalisation attendue : $\{C \sqcap X_2 \sqsubseteq X_3, A \sqcap B \sqsubseteq X_1, \exists r.X_1 \sqsubseteq X_2, C \sqcap X_2 \sqsubseteq F, X_3 \sqsubseteq r.X_4, X_4 \sqsubseteq \exists s.D, X_4 \sqsubseteq E\}$
Normalisation obtenue : $\{C \sqcap X_2 \sqsubseteq R\_r.X_3, A \sqcap B \sqsubseteq X_1, R\_r.X_1 \sqsubseteq X_2, X_3 \sqsubseteq R\_s.D, C \sqcap X_2 \sqsubseteq F, X_3 \sqsubseteq E, R\_r.X_1 \sqsubseteq \exists r.X_1, \exists r.X_1 \sqsubseteq R\_r.X_1, R\_s.D \sqsubseteq \exists s.D, \exists s.D \sqsubseteq R\_s.D, R\_r.X_3 \sqsubseteq \exists r.X_3, \exists r.X_3 \sqsubseteq R\_r.X_3\}$

B.4.2 Classification

Voir 6.1.2 (p.90) pour la description des tests. Les règles de classification sont celles de la table 5.5 (p.81).

Test 15 (CR1)**.** *TBox :* $\{A \sqsubseteq B\}$
Classification attendue : $S(A) = \{A, B\}, S(B) = \{B\}$
Classification obtenue : $S(A) = \{A, B\}, S(B) = \{B\}$

Test 16 (CR1 avec une équivalence)**.** *TBox :* $\{A \equiv B\}$
Classification attendue : $S(A) = \{A, B\}, S(B) = \{B, A\}$
Classification obtenue : $S(A) = \{A, B\}, S(B) = \{B, A\}$

Test 17 (CR1)**.** *TBox :* $\{A \sqsubseteq B, B \sqsubseteq C\}$
Classification attendue : $S(A) = \{A, B, C\}, S(B) = \{B, C\}, S(C) = \{C\}$
Classification obtenue : $S(A) = \{A, B, C\}, S(B) = \{B, C\}, S(C) = \{C\}$

Test 18 (CR3)**.** *TBox :* $\{A \sqsubseteq \exists x.Y\}$
Classification attendue : $S(A) = \{A\}, S(Y) = \{Y\}, R(x) = \{A, Y\}$
Classification obtenue : $S(A) = \{A, R\_r.Y\}, S(Y) = \{Y\}, S(R\_x.Y) = \{R\_x.Y\}, R(x) = \{R\_x.Y, Y\}$

Test 19 (CR1 due à la normalisation supplémentaire)**.** *TBox :* $\{\exists x.Y \sqsubseteq A\}$
Classification attendue : $S(A) = \{A\}, S(Y) = \{Y\}$
Classification obtenue : $S(A) = \{A\}, S(Y) = \{Y\}, S(R\_x.Y) = \{R\_x.Y, A\}, R(x) = \{R\_x.Y, Y\}$

Test 20 (CR2). *TBox :* $\{A \sqsubseteq B, A \sqcap B \sqsubseteq C\}$
Classification attendue : $S(A) = \{A, B, C\}, S(B) = \{B\}, S(C) = \{C\}$
Classification obtenue : $S(A) = \{A, B, C\}, S(B) = \{B\}, S(C) = \{C\}$

Test 21 (CR4). *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.D, \exists r.D \sqsubseteq E\}$
Classification attendue : $S(A) = \{A, B, E\}, S(B) = \{B, E\}, S(D) = \{D\}, S(E) = \{E\}, R(r) = \{B, D\}$
Classification obtenue : $S(A) = \{A, B, R\_r.D, E\}, S(B) = \{B, R\_r.D, E\}, S(D) = \{D\}, S(E) = \{E\}, S(R\_r.D) = \{R\_r.D, E\}, R(r) = \{R\_r.D, D\}$

Test 22 (CR3, CR4, CR2 et CR1). *TBox :* $\{A \sqsubseteq B, A \sqcap B \sqsubseteq \exists r.D, \exists r.D \sqsubseteq E\}$
Classification attendue : $S(A) = \{A, B, X_1, E\}, S(B) = \{B\}, S(D) = \{D\},$
$S(E) = \{E\}, R(r) = \{X_1, D\}$
Classification obtenue : $S(A) = \{A, B, R\_r.D, E\}, S(B) = \{B\}, S(D) = \{D\},$
$S(E) = \{E\}, S(R\_r.D) = \{R\_r.D, E\}, R(r) = \{R\_r.D, D\}$

Test 23 (CR3, CR4, CR2 et CR1). *TBox :* $\{A \sqsubseteq B, A \sqcap B \sqsubseteq \exists r.D, \exists r.D \sqsubseteq E, E \sqsubseteq F \sqcap G$
Classification attendue : $S(A) = \{A, B, X_1, E, F, G\}, S(B) = \{B\}, S(D) = \{D\},$
$S(E) = \{E, F, G\}, S(F) = \{F\}, S(G) = \{G\}, R(r) = \{X_1, D\}$
Classification obtenue : $S(A) = \{A, B, R\_r.D, E, F, G\}, S(B) = \{B\}, S(D) = \{D\},$
$S(E) = \{E, F, G\}, S(F) = \{F\}, S(G) = \{G\}, S(R\_r.D) = \{R\_r.D, E, F, G\}, R(r) = \{R\_r.D, D\}$

Test 24 (CR3, CR4, CR2 et CR1). *TBox :* $\{A \sqsubseteq B, A \sqcap B \sqsubseteq \exists r.D, \exists r.D \sqsubseteq E, E \sqsubseteq F \sqcap G, B \sqcap G \sqsubseteq H\}$
Classification attendue : $S(A) = \{A, B, X_1, E, F, G, H\}, S(B) = \{B\}, S(D) = \{D\},$
$S(E) = \{E, F, G\}, S(F) = \{F\}, S(G) = \{G\}, S(H) = \{H\}, R(r) = \{X_1, D\}$
Classification obtenue : $S(A) = \{A, B, R\_r.D, E, F, G\}, S(B) = \{B\}, S(D) = \{D\},$
$S(E) = \{E, F, G\}, S(F) = \{F\}, S(G) = \{G\}, S(H) = \{H\},$
$S(R\_r.D) = \{R\_r.D, E, F, G\}, R(r) = \{(R\_r.D, D)\}$

Test 25 (Vérification que $\exists r.(A \sqcap B) \neq \exists r.A \sqcap \exists r.B$). *TBox :* $\{\exists x.(Y \sqcap Z) \equiv A, \exists x.Y \sqcap \exists x.Z \equiv B\}$
Classification attendue : $S(A) = \{A, X_2, X_3, B\}, S(B) = \{B, X_2, X_3\}, S(X_1) = \{X_1\},$
$S(X_2) = \{X_2\}, S(X_3) = \{X_3\}, S(X_4) = \{X_4, Y, Z, X_1\}, S(Z) = \{Z\}, S(Y) = \{Y\},$
$R(x) = \{(A, X_4), (B, Z), (B, Y)\}$
Classification obtenue : $S(A) = \{A, R\_x.X_4, R\_x.X_1, R\_x.Y, R\_x.Z, X_2, X_3, B\},$
$S(B) = \{B, R\_x.Y, R\_x.Z, X_2, X_3\}, S(Z) = \{Z\}, S(Y) = \{Y\}, S(X_1) = \{X_1\}, S(X_2) = \{X_2\}, S(X_3) = \{X_3\}, S(X_4) = \{X_4, Y, Z, X_1\},$
$S(R\_x.1) = \{R\_x.1, A, R\_x.X_4, R\_x.Y, R\_x.Z, X_2, X_3, B\},$
$S(R\_x.4) = \{R\_x.4, R\_x.1, R\_x.Y, R\_x.Z, A, X_2, X_3, B\}, S(R\_x.Y) = \{R\_x.Y, X_2\},$
$S(R\_x.Z) = \{R\_r.Z, X_3\}, R(x) = \{(R\_x.1, X_1), (R\_x.4, X_4), (R\_x.Y, Y), (R\_x.Z, Z)\}$

B.4.3 Fill-up

Voir 6.1.3 (p.90) pour la description des tests.

Test 26. *TBox :* $\{A \sqsubseteq B\}$
Concept : A
Fill-up attendu : $A \sqcap B$
Fill-up obtenu : $A \sqcap B$

Test 27. *TBox :* $\{A \sqsubseteq B\}$
Concept : B
Fill-up attendu : B
Fill-up obtenu : B

Test 28. *TBox :* $\{A \sqsubseteq B\}$
Concept : $\top$
Fill-up attendu : $\top$
Fill-up obtenu : $\top$

Test 29. *TBox :* $\{A \sqsubseteq B\}$
Concept : $\exists r.A$
Fill-up attendu : $\exists r.(A \sqcap B)$
Fill-up obtenu : $\exists r.(A \sqcap B)$

Test 30. *TBox :* $\{A \sqsubseteq B, C \sqsubseteq D\}$
Concept : $\exists r.A \sqcap C$
Fill-up attendu : $\exists r.(A \sqcap B) \sqcap C \sqcap D$
Fill-up obtenu : $\exists r.(A \sqcap B) \sqcap C \sqcap D$

Test 31. *TBox :* $\{A \sqsubseteq B, C \sqsubseteq D\}$
Concept : $\exists r.\exists s.A \sqcap C$
Fill-up attendu : $\exists r.\exists s.(A \sqcap B) \sqcap C \sqcap D$
Fill-up obtenu : $\exists r.\exists s.(A \sqcap B) \sqcap C \sqcap D$

Test 32. *TBox :* $\{A \sqsubseteq \exists r.B, C \sqsubseteq \exists r.D, D \sqsubseteq E, E \sqsubseteq \exists s.F\}$
Concept : $A \sqcap C$
Fill-up attendu : $A \sqcap C \sqcap \exists r.B \sqcap \exists r.(D \sqcap E \sqcap \exists s.F)$
Fill-up obtenu : $A \sqcap C \sqcap \exists r.B \sqcap \exists r.(D \sqcap E \sqcap \exists s.F)$

Test 33. *TBox :* $\{A \sqsubseteq \exists r.B, C \sqsubseteq \exists r.D, D \sqsubseteq E, E \sqsubseteq \exists s.F\}$
Concept : $A \sqcap \exists r.\top \sqcap \exists s.(\top \sqcap H)$
Fill-up attendu : $A \sqcap \exists r.B \sqcap \exists r.\top \sqcap \exists s.(\sqcap H)$
Fill-up obtenu : $A \sqcap \exists r.B \sqcap \exists r.\top \sqcap \exists s.(\sqcap H)$

B.4.4 TSO

Voir 6.1.4 (p.91) pour la description des tests.

Test 34. *Diminuende :* A
Diminiteur : A
TSO attendu : $\top$
TSO obtenu : $\top$

Test 35. *Diminuende :* A
Diminiteur : B
TSO attendu : A
TSO obtenu : A

Test 36. *Diminuende :* $\top$
Diminiteur : $\top$
TSO attendu : $\top$
TSO obtenu : $\top$

Test 37. *Diminuende :* A
Diminiteur : $\top$
TSO attendu : A
TSO obtenu : A

Test 38. *Diminuende :* $A \sqcap B$
Diminiteur : A
TSO attendu : B
TSO obtenu : B

Test 39. *Diminuende :* $A \sqcap B$
Diminiteur : $A \sqcap B$
TSO attendu : $\top$
TSO obtenu : $\top$

Test 40. *Diminuende :* $A \sqcap B$
Diminiteur : $A \sqcap B \sqcap C$
TSO attendu : $\top$
TSO obtenu : $\top$

Test 41. *Diminuende :* $A \sqcap B \sqcap C$
Diminiteur : $A \sqcap B$
TSO attendu : C
TSO obtenu : C

Test 42. *Diminuende :* $\exists r.A$
Diminiteur : A
TSO attendu : $\exists r.A$
TSO obtenu : $\exists r.A$

Test 43. *Diminuende :* $\exists r.A$
Diminiteur : $\exists r.A$
TSO attendu : $\top$
TSO obtenu : $\top$

Test 44. *Diminuende :* $\exists r.(A \sqcap B)$
Diminiteur : $\exists r.A$
TSO attendu : $\exists r.B$
TSO obtenu : $\exists r.B$

Test 45. *Diminuende :* $\exists r.(A \sqcap B) \sqcap C \sqcap D$
Diminiteur : $\exists r.A \sqcap C \sqcap E$
TSO attendu : $\exists r.B \sqcap D$
TSO obtenu : $\exists r.B \sqcap D$

Test 46. *Diminuende :* $\exists r.(A \sqcap B) \sqcap C \sqcap D$
Diminiteur : $\exists s.A \sqcap C \sqcap E$
TSO attendu : $\exists r.(A \sqcap B) \sqcap D$
TSO obtenu : $\exists r.(A \sqcap B) \sqcap D$

Test 47. *Diminuende :* $\exists r.(A \sqcap B \sqcap \exists S.A) \sqcap C \sqcap D$
Diminiteur : $\exists s.A \sqcap C \sqcap E$
TSO attendu : $\exists r.(A \sqcap B \sqcap \exists S.A) \sqcap D$
TSO obtenu : $\exists r.(A \sqcap B \sqcap \exists S.A) \sqcap D$

Test 48. *Diminuende :* $\exists r.(A \sqcap B \sqcap \exists S.A) \sqcap C \sqcap D$
Diminiteur : $\exists r.(A \sqcap \exists S.A) \sqcap C \sqcap E$
TSO attendu : $\exists r.B \sqcap D$
TSO obtenu : $\exists r.B \sqcap D$

Test 49. *Diminuende :* $\exists r.A$
Diminiteur : $\exists r.\top$
TSO attendu : $\exists r.A$
TSO obtenu : $\exists r.A$

Test 50. *Diminuende :* $\exists r.\top$
Diminiteur : $\exists r.A$
TSO attendu : $\top$
TSO obtenu : $\top$

B.4.5 Calcul de subsomption

Voir 6.1.5 (p.92) pour la description des tests.

Test 51. *TBox :* $\{A \sqsubseteq B\}$
Demande : A
Réponse : A
Subsomption attendue : équivalence
Subsomption obtenue : équivalence

Test 52. *TBox :* $\{A \sqsubseteq B\}$
Demande : A
Réponse : B
Subsomption attendue : demande subsumée
Subsomption obtenue : demande subsumée

Test 53. *TBox :* $\{A \sqsubseteq B\}$
Demande : A
Réponse : C
Subsomption attendue : Aucune relation
Subsomption obtenue : Aucune relation

Test 54. *TBox :* $\{A \sqsubseteq \exists r.B\}$
Demande : A
Réponse : $\exists r.B$
Subsomption attendue : demande subsumée
Subsomption obtenue : demande subsumée

Test 55. *TBox :* $\{A \sqsubseteq \exists r.B\}$
Demande : A
Réponse : $\exists r.C$
Subsomption attendue : Aucune relation
Subsomption obtenue : Aucune relation

Test 56. *TBox :* $\{A \sqsubseteq \exists r.B\}$
Demande : $\exists r.(B \sqcap C)$
Réponse : $\exists r.B \sqcap \exists r.C$
Subsomption attendue : demande subsumée
Subsomption obtenue : demande subsumée

B.4.6 Rest et miss

Voir 6.1.6 (p.92) pour la description des tests.

Test 57. *TBox :* $\{A \sqsubseteq B\}$
Demande : A
Réponse : B
Rest attendu : A
Rest obtenu : A
Miss attendu : $\top$
Miss obtenu : $\top$

Test 58. *TBox :* $\{A \sqsubseteq B\}$
Demande : A
Réponse : A
Rest attendu : $\top$
Rest obtenu : $\top$
Miss attendu : $\top$
Miss obtenu : $\top$

Test 59. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq C\}$
Demande : C

Réponse : A
Rest attendu : $\top$
Rest obtenu : $\top$
Miss attendu : $A \sqcap B$
Miss obtenu : $A \sqcap B$

Test 60. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.C\}$
Demande : $\exists r.C$
Réponse : A
Rest attendu : $\top$
Rest obtenu : $\top$
Miss attendu : $A \sqcap B$
Miss obtenu : $A \sqcap B$

Test 61. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.C\}$
Demande : $\exists r.C \sqcap D$
Réponse : A
Rest attendu : D
Rest obtenu : D
Miss attendu : $A \sqcap B$
Miss obtenu : $A \sqcap B$

Test 62. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.C\}$
Demande : $\exists r.C \sqcap D$
Réponse : $A \sqcap E$
Rest attendu : D
Rest obtenu : D
Miss attendu : $A \sqcap B \sqcap E$
Miss obtenu : $A \sqcap B \sqcap E$

Test 63. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.C, A \sqcap E \sqsubseteq D\}$
Demande : $\exists r.C \sqcap D$
Réponse : $A \sqcap E$
Rest attendu : $\top$
Rest obtenu : $\top$
Miss attendu : $A \sqcap B \sqcap E$
Miss obtenu : $A \sqcap B \sqcap E$

Test 64. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.C, A \sqcap E \sqsubseteq D\}$
Demande : $\exists r.C \sqcap D$
Réponse : X
Rest attendu : $\exists r.C \sqcap D$
Rest obtenu : $\exists r.C \sqcap D$
Miss attendu : X
Miss obtenu : X

Test 65. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.C, A \sqcap E \sqsubseteq D, X \equiv Y\}$
Demande : $\exists r.C \sqcap Y$
Réponse : X
Rest attendu : $\exists r.C$
Rest obtenu : $\exists r.C$
Miss attendu : $\top$
Miss obtenu : $\top$

Test 66. *TBox :* $\{A \sqsubseteq B, B \sqsubseteq \exists r.C, A \sqcap E \sqsubseteq D, X \equiv Y, \exists r.X \sqsubseteq Z$
Demande : $\exists r.C \sqcap Y\}$
Réponse : $\exists r.Y$
Rest attendu : $\exists r.C \sqcap Y \sqcap X$
Rest obtenu : $\exists r.C \sqcap Y \sqcap X$
Miss attendu : $\exists r.(X \sqcap Y) \sqcap Z$
Miss attendu : $\exists r.(X \sqcap Y) \sqcap Z$

B.4.7 CCO avec une seul composante

Voir 6.1.7 (p.93) pour la description des tests.

Test 67. *TBox :* $\{A \sqsubseteq B\}$
Demande : B
Réponses : r0 : A, r1 : B
Classement attendu : (r1,1), (r0, -1)
Classement obtenu : (r1,1), (r0, -1)

Test 68. *TBox :* $\{A \sqsubseteq B\}$
Demande : B
Réponses : r0 : A, r1 : C
Classement attendu : (r0,1), (r1, -1)
Classement obtenu : (r0,1), (r1, -1)

Test 69. *TBox :* $\{A \sqsubseteq B, C \sqsubseteq A\}$
Demande : B
Réponses : r0 : A, r1 : C
Classement attendu : (r0,1), (r1, -1)
Classement obtenu : (r0,1), (r1, -1)

Test 70. *TBox :* $\{A \sqsubseteq B, C \sqsubseteq A, B \sqsubseteq D\}$
Demande : B
Réponses : r0 : A, r1 : D
Classement attendu : (r0,1), (r1, -1)
Classement obtenu : (r0,1), (r1, -1)

Test 71. *TBox :* $\{A \sqsubseteq B, C \sqsubseteq A, B \sqsubseteq D\}$
Demande : B

Réponses : r0 : E, r1 : D
Classement attendu : (r1,1), (r0, -1)
Classement obtenu : (r1,1), (r0, -1)

Test 72. *TBox : {*$A \sqsubseteq B, C \sqsubseteq A, B \sqsubseteq D$*}*
Demande : B
Réponses : r0 : $E \sqcap A$*, r1 :* D
Classement attendu : (r0,1), (r1, -1)
Classement obtenu : (r0,1), (r1, -1)

Test 73. *TBox : {*$B \sqsubseteq A, C \sqsubseteq A, D \sqsubseteq B, E \sqsubseteq A$*}*
Demande : B
Réponses : r0 : C*, r1 :* D*, r2 :* A*, r3 :* E*, r4 :* $D \sqcap X$
Classement attendu : (r1,4), (r4, 2), (r2, 0), (r3, -3), (r0, -3)
Classement obtenu : (r1,4), (r4, 2), (r2, 0), (r3, -3), (r0, -3)

Test 74. *TBox : {*$A \sqsubseteq B, C \sqsubseteq A, B \sqsubseteq D$*}*
Demande : B
Réponses : r0 : A*, r1 :* $A \sqcap X$*, r2 :* X*, r3 :* B*, r4 :* $D \sqcap X$
Classement attendu : (r3,4), (r0, 2), (r1, 0), (r4, -2), (r2, -4)
Classement obtenu : (r3,4), (r0, 2), (r1, 0), (r4, -2), (r2, -4)

Test 75. *TBox : {*$A \sqsubseteq B, C \sqsubseteq A, B \sqsubseteq D, Z \sqsubseteq \exists r.X$*}*
Demande : $B \sqcap \exists r.(X \sqcap Y)$
Réponses : r0 : A*, r1 :* $A \sqcap X$*, r2 :* $\exists r.X \sqcap Y$*, r3 :* Z*, r4 :* $D \sqcap \exists s.X$
Classement attendu : (r0,4), (r1, 2), (r3, -1), (r2, -1), (r4, -4)
Classement obtenu : (r0,4), (r1, 2), (r3, -1), (r2, -1), (r4, -4)

B.4.8 CCO avec plusieurs composante

Voir 6.1.8 (p.93) pour la description des tests.

Test 76. *TBox : {*$A2 \sqsubseteq A1, B2 \sqsubseteq B1$*}*
Demande : $\exists a.A1 \sqcap \exists b.B1$
Réponses : r0 : $\exists a.A2 \sqcap \exists b.B2$*, r1 :* $\exists a.A1 \sqcap \exists b.B1$*, r2 :* $\exists a.A1 \sqcap \exists b.B2$*, r3 :* $\exists a.A2 \sqcap \exists b.B1$*,*
r4 : $\exists a.(A2 \sqcap A1) \sqcap \exists b.B1$*, r5 :* $\exists a.A3 \sqcap \exists b.B1$

Classement attendu pour a : (r1,4), (r2, 4), (r4, -1), (r0, -1), (r3, -1), (r5, -5)
Classement obtenu pour a : (r1,4), (r2, 4), (r4, -1), (r0, -1), (r3, -1), (r5, -5)
Classement attendu pour b : (r5,2), (r4, 2), (r3, 2), (r1, 2), (r2, -4), (r0, -4)
Classement obtenu pour b : (r5,2), (r4, 2), (r3, 2), (r1, 2), (r2, -4), (r0, -4)
Classement attendu : (r1,6), (r3, 1), (r4, 1), (r2, 0), (r5, -3), (r0, -5)
Classement obtenu : (r1,6), (r3, 1), (r4, 1), (r2, 0), (r5, -3), (r0, -5)

Test 77. *TBox : {*$A2 \sqsubseteq A1, B2 \sqsubseteq B1$*}*
Demande : $\exists a.A1 \sqcap \exists b.B1 \sqcap \exists c.\top$

Réponses : r0 : $\exists a.A1 \sqcap \exists b.B1 \sqcap \exists c.C2$*, r1 :* $\exists a.A1 \sqcap \exists b.B1 \sqcap \exists c.\top$*, r2 :* $\exists a.A2 \sqcap \exists b.B2 \sqcap \exists c.\top$*, r3 :* $\exists a.A1 \sqcap \exists b.B2 \sqcap \exists c.\top$*, r4 :* $\exists a.A2 \sqcap \exists b.B1 \sqcap \exists c.\top$*, r5 :* $\exists a.(A2 \sqcap A1) \sqcap \exists b.B1 \sqcap \exists c.\top$*, r6 :* $\exists a.A3 \sqcap \exists b.B1 \sqcap \exists c.\top$
Classement attendu pour a : (r0,4), (r1, 4), (r3, 4), (r5, -2), (r4, -2), (r2, -2), (r6, -6)
Classement obtenu pour a : (r0,4), (r1, 4), (r3, 4), (r5, -2), (r4, -2), (r2, -2), (r6, -6)
Classement attendu pour b : (r0,2), (r6, 2), (r5, 2), (r4, 2), (r1, 2), (r3, -5), (r2, -5)
Classement obtenu pour b : (r0,2), (r6, 2), (r5, 2), (r4, 2), (r1, 2), (r3, -5), (r2, -5)
Classement attendu pour c : (r1,1), (r3, 1), (r2, 1), (r4, 1), (r5, 1), (r6, 1), (r0, -6)
Classement obtenu pour c : (r1,1), (r3, 1), (r2, 1), (r4, 1), (r5, 1), (r6, 1), (r0, -6)
Classement attendu : (r1,7), (r4, 1), (r5, 1), (r0, 0), (r3, 0), (r6, -3), (r2, -6)
Classement obtenu : (r1,7), (r4, 1), (r5, 1), (r0, 0), (r3, 0), (r6, -3), (r2, -6)

B.5 Génération aléatoire pour les tests quantitatifs

On présente dans cette section la génération d'une TBox et d'un jeu de réponses associé aléatoire. Cette génération est utilisée lors de la réalisation des tests quantitatifs de la section 6.2 (p.94).

B.5.1 Génération d'une TBox

Pour générer une TBox acyclique et générale de manière aléatoire, on utilise deux fonctions : une fonction d'initialisation (algorithme 18 (p.156)) et une fonction récursive de génération (algorithme 19 (p.157)). La fonction d'initialisation prend en entrées les valeurs des différents facteurs liés à la TBox (identifiés dans le tableau 6.2 (p.94)) qui sont la taille de la TBox (en nombre d'axiomes), la probabilité de générer un axiome composé, la profondeur maximale des axiomes et le nombre de composantes. On fixe également deux autres facteurs : le nombre d'axiomes maximum dans lequel un nom de concept peut apparaître est fixé à 4, et le nombre de rôles maximum est de 3 (r, s et t).

Pour chaque composante, on va créer un nombre égal d'axiomes. C'est-à-dire que pour une TBox de 1000 axiomes et deux composantes, on aura 500 axiomes dans la première composante et 500 dans la seconde. L'algorithme 18 (p.156) génère ensuite les premiers axiomes de la composante, qui sont de type $A_X \sqsubseteq A_0$, où A est le nom de la composante et X un nombre qui commence à 1 et qui incrémente à chaque nouveau concept. On en créé autant de fois que A_0 peut apparaître. Ce nombre arbitraire permet d'obtenir une hiérarchie initiale la plus large possible et c'est le traitement suivant qui ajoutera des axiomes plus variés à ceux de l'initialisation. On effectue ensuite le traitement récursif qui ajoute de nouveaux concepts.

L'algorithme 19 (p.157) génère à chaque itération un nouveau concept et l'insère dans la TBox dans 1 à 4 axiomes, dont la forme dépends des chances d'avoir un axiome composé. Les noms de concepts qui peuvent encore être utilisés dans un axiome sont stockés dans un tableau, tiré aléatoirement lors de la création d'un nouvel axiome et retiré une fois le nombre maximum d'utilisations atteint. L'algorithme continue d'ajouter des concepts et des axiomes jusqu'à ce que la taille souhaitée pour une composante soit atteinte, et on passe à la composante suivante. Une génération de TBox est proposée dans l'exemple 44 (p.158).

Algorithm 18 Algorithme d'initialisation de la génération des axiomes d'une composante d'une TBox pour génération aléatoire.

Require: $Size$ la taille de la composante en nombre d'axiome, Exp les chances (en %) d'avoir un axiome composé lors de la génération d'un axiome, $Depth$ la profondeur maximale des concepts générés et, $Compo$ le nom de la composante. Exp, $Compo$ et $Depth$ sont stockées sous forme de variables de classe.

Ensure: l'ensemble des axiomes de la composante $Compo$ pour la TBox généré aléatoirement, dans le tableau $TBox_{Compo}$.

1: $ConceptTab$ est un tableau vide qui contiendra les noms de concept pouvant encore être utilisés dans les axiomes.
2: $TBox_{Compo}$ est un tableau vide qui contiendra les axiomes de la TBox pour la composante en cours ($Compo$).
3: **for** i de 1..4 **do**
4: Ajout de $Compo_i \sqsubseteq Compo_0$ à $TBox_{Compo}$.
5: Ajout de $Compo_i$ dans $ConceptTab$ 3 fois .
6: **end for**
7: On créé *compteur*, une variable de classe initialisée à 5.
8: $Size$=$Size$-4
9: $TBox$= recursiveOnto($Size$, $ConceptTab$, $TBox_{Compo}$, 0)
10: Retourner $TBox_{Compo}$

Algorithm 19 recursiveOnto(), algorithme de génération aléatoire d'une composante d'une TBox. Notons que les lignes 6 à 8 ne pouvant être exécutées sans problème qu'une fois par récursion, des gardiens ont été ajoutés pour que ce cas n'arrive qu'une fois, et ont été omis dans cet algorithme pour plus de clarté.

Require: $Size$ le nombre d'axiomes restants à ajouter à la TBox pour cette composante, $ConceptTab$ le tableau contenant les noms de concept pouvant encore être utilisés dans les axiomes, $TBox$ le tableau contenant les axiomes de la TBox pour la composante en cours et $LinkModif$ qui indique si le concept B en cours a été utilisé dans un axiome de type $A \sqcap B \sqsubseteq C$ à la récursion précédente. Exp, $Compo$, $Depth$ et $Compteur$ sont disponibles sous forme de variable de classe.

Ensure: $TBox_{Compo}$ avec au moins un nouvel axiome contenant un nouveau nom de concept.

1: On tire aléatoirement un nombre n entre 1 et 4 (qui ne peut pas dépasser $Size$).
2: On créé le nom de concept C de la forme $Compo_{Compteur}$.
3: **for** i de 0 à $n-1$ **do**
4: On tire aléatoirement un concept E qui n'a pas été tiré dans une itération précédente de cette boucle dans le tableau $CompoTab$ et on le retire de celui-ci.
5: On tire un nombre $rand$ entre 0 et 100.
6: **if** $rand < Exp/2$ **then**
7: On créé le nom de concept D de la forme $Compo_{Compteur+1}$ et $LinkModif = 1$.
8: On ajoute $C \sqcap D \sqsubseteq E$ à $TBox_{Compo}$.
9: **else if** $rand >= Exp/2$ et $rand < Exp$ **then**
10: On ajoute $\exists X.C \sqsubseteq E$ à $TBox$ où X est une imbrication de rôle (tiré aléatoirement parmi r, s et t d'une profondeur allant de 1 à $Depth$).
11: **else if** $rand >= Exp$ et $rand <$ (100-Exp)/2 + Exp **then**
12: On ajoute $C \sqsubseteq \exists X.E$ à $TBox_{Compo}$ où X est une imbrication de rôle (tiré aléatoirement parmi r, s et t d'une profondeur allant de 1 à $Depth$).
13: **else**
14: On ajoute $C \sqsubseteq E$ à $TBox_{Compo}$.
15: **end if**
16: **end for**
17: $Size = Size - n$
18: **if** $Size > 0$ **then**
19: $Compteur = Compteur + 1$
20: $TBox_{Compo}$= recursiveOnto($Size$, $ConceptTab$, $TBox_{Compo}$, $LinkModif$)
21: **end if**
22: Retourner $TBox_{Compo}$

Exemple 44. *On veut créer une TBox de 7 axiomes, avec Exp=20% de chance d'avoir un axiome composé, de profondeur maximale 2 et avec une seule composante A. On fixe le nombre de fois qu'un concept peut-être utilisé dans un axiome à 3 seulement (et non 4 comme dans l'algorithme, pour des questions de brièveté de l'exemple).*

- *On initialise d'abord la partie de TBox pour la composante A, $TBox_A$:*
  - $\mathcal{T} = \{A_1 \sqsubseteq A_0, A_2 \sqsubseteq A_0, A_3 \sqsubseteq A_0\}$
  - *on a [A_1, A_1, A_2, A_2, A_3, A_3] dans ConceptTab.*
  - *Size = 4*
- *Itération 1 de recursiveOnto(Size, ConceptTab, $TBox_A$, 0)*
  - *n = 1 et on créé A_4*
  - *Rand= 8 et 8 < 20 / 2 (soit Exp/2), on va créer un axiome de type $A \sqcap B \sqsubseteq C$*
    - *On créé A_5 et on tire A_2 dans ConceptTab*
    - *On ajoute $A_4 \sqcap A_5 \sqsubseteq A_2$ à $TBox_{Compo}$ et $LinkModif$=1*
  - *ConceptTab devient [A_1, A_1, A_2, A_3, A_3, A_4, A_4]*
  - *Size = 3*
- *Itération 2 de recursiveOnto(Size, ConceptTab, $TBox_A$, 1)*
  - *n = 1 et A_5 existe déjà*
  - *Rand= 18 et 18 < 20 (soit Exp) et 18 >= 10, on va crée une axiome de type $\exists r.A \sqsubseteq B$*
    - *On tire A_3, une profondeur de 1 et r comme rôle*
    - *On ajoute $\exists r.A_5 \sqsubseteq A_3$ à $TBox_A$ et $LinkModif$=0*
  - *ConceptTab devient [A_1, A_1, A_2, A_3, A_4, A_4 , A_5, A_5]*
  - *Size = 2*
- *Itération 3 de recursiveOnto(Size, ConceptTab, $TBox_A$, 0)*
  - *n = 2 et on créé A_6*
  - *Première itération i=0 :*
    - *Rand= 35 et 35 >= 20 (Exp) et 35 < 60 (soit (100-Exp)/2+Exp), on va créer un axiome de type $A \sqsubseteq \exists r.B$*
      - *On tire A_3, une profondeur de 2 et r puis t en rôle*
      - *On ajoute $A_6 \sqsubseteq \exists r.\exists t.A_3$ à $TBox_A$*
    - *ConceptTab devient [A_1, A_1, A_2, A_4, A_4]*
  - *Seconde itération i=1 :*
    - *Rand= 95 et 95 >= 40 (soit (100-Exp)/2 + Exp), on va créer un axiome de type $A \sqsubseteq B$*
      - *On tire A_4*
      - *On ajoute $A_6 \sqsubseteq A_4$ à $TBox_A$*
    - *ConceptTab devient [A_1, A_1, A_2, A_4]*
  - *Size = 0*

L'algorithme s'arrête car on a atteint la taille souhaitée et
$TBox_A = \{A_1 \sqsubseteq A_0,$
$A_2 \sqsubseteq A_0,$
$A_3 \sqsubseteq A_0,$
$A_4 \sqcap A_5 \sqsubseteq A_2,$

$\exists r.A_5 \sqsubseteq A_3$,
$A_6 \sqsubseteq \exists r.\exists t.A_3$,
$A_6 \sqsubseteq A_4\}$.

B.5.2 Génération d'un jeu de réponses

Pour générer un jeu de réponses lié à une TBox générée aléatoirement précédemment, on utilise une fonction de génération utilisant les valeurs des différents facteurs liés aux réponses (identifiés dans le tableau 6.2 (p.94)) : le nombre de réponses à générer, le nombre maximal de conjoncts dans la réponse et la profondeur des réponses. L'algorithme 20 (p.160) va créer autant de réponses avec le bon nombre de composantes. Notons que la profondeur des réponses, contrairement à la génération de la TBox, est une valeur fixe et pas une valeur maximale, pour s'assurer de l'influence de la profondeur dans les tests correspondants. On fixe également les chances qu'une restriction existentielle soit générée à 20%.

Algorithm 20 Algorithme de génération d'un jeu de réponses aléatoire associé à une TBox générée aléatoirement précédemment. Notons que la fonction .succ() est une fonction Ruby qui retourne le caractère suivant de l'opérande de gauche. Par exemple, A.succ $= B$. La fonction est utilisée pour générer le nom de la composante de l'itération suivante.

Require: Nb le nombre de réponses à générer, $Compo$ le nombre de composantes, $Depth$ la profondeur des réponses, $maxC$ le nombre de conjoncts maximal par composante, $conceptList$ la liste des concepts présents dans la TBox associée et $rChance$ les chances d'avoir une restriction existentielle comme conjonct en %.

Ensure: Une liste de taille $SizeNb$ contenant des réponses aléatoires liées à la TBox.

On initialise un tableau $repTab$ vide, qui contiendra la liste des réponses.
for i de 0 à Nb-1 **do**
  On initialise un tableau rep vide, qui contiendra la réponse.
  Lettre $=A$
  On initialise un tableau $compTab$ vide qui contiendra le concept de la composante en cours pour la réponse.
  for j de 0 à $Compo$-1 **do**
    On tire aléatoirement un nombre nbC entre 0 et $maxC$.
    if $nbC == 0$ **then**
      $compTab = [\top]$
    else
      for k de 0 à nbC-1 **do**
        On tire aléatoirement un nombre $rand$ entre 0 et 100.
        On tire aléatoirement un nom de concept C dans $conceptList$ (qui n'a pas déjà été tiré pour cette conjonction).
        if $rand < rChance$ **then**
          On ajoute à $compTab$ $\exists X.C$ où X est une imbrication de rôle (tiré aléatoirement parmi r, s et t d'une profondeur $Depth$).
        else
          On ajoute C à $compTab$.
        end if
        if k $\neq$ nbC-1 **then**
          On ajoute $\sqcap$ à $compTab$.
        end if
      end for
    end if
    On ajoute $\exists Lettre.$ et $compTab$ à rep.
    $Lettre$.succ()
    if j $\neq$ $Compo$-1 **then**
      On ajoute $\sqcap$ à rep.
    end if
  end for
  On ajoute rep à $repTab$
end for
Retourner $repTab$

Exemple 45. *On a*
$\mathcal{T} = \{A_1 \sqsubseteq A_0,$
$A_2 \sqsubseteq A_0,$
$A_3 \sqsubseteq A_0,$
$A_4 \sqcap A_5 \sqsubseteq A_2,$
$\exists r.A_5 \sqsubseteq A_3,$
$A_6 \sqsubseteq \exists r.\exists t.A_3,$
$A_6 \sqsubseteq A_4\}.$

On a la profondeur fixée à 2, le nombre de conjoncts fixé à 2, 20% de chance d'avoir une restriction existentielle dans une réponse et on veut obtenir 3 réponses d'une seule composante :
On initialise le tableau des réponses repTab à vide

- *Réponse 1 :*
  - *On initialise le tableau rep=∃compA. .*
  - *On initialise le tableau compTab à vide.*
  - *On tire nbC = 1.*
  - *rand= 5, 5 < 20, on ajoute donc un rôle.*
  - *On tire s et r pour les rôles, A_4 comme nom de concept.*
  - *compTab=$\exists s.\exists r.A_4$ et on ajoute compTab à rep,*
    qui est maintenant $\exists compA.\exists s.\exists r.A_4$
- *On ajoute rep à la liste repTab.*
- *Réponse 2 :*
  - *On initialise le tableau rep=∃compA. .*
  - *On initialise le tableau compTab à vide.*
  - *On tire nbC = 0, on ajoute donc $\mathcal{T}$.*
  - *compTab=$\top$ et on ajoute compTab à rep.*
- *On ajoute rep à la liste repTab.*
- *Réponse 3 :*
  - *On initialise le tableau rep=∃compA. .*
  - *On initialise le tableau compTab à vide.*
  - *On tire nbC = 2.*
  - *Premier conjonct :*
    - *rand= 36, 36 > 20, on ajoute donc un nom de concept.*
    - *On tire A_3 comme nom de concept.*
    - *compTab=$A_3\sqcap$*
  - *Deuxième conjonct :*
    - *rand= 77, 77 > 20, on ajoute donc un nom de concept.*
    - *On tire A_1 comme nom de concept.*
    - *compTab=$A_3 \sqcap A_1$ et on ajoute compTab à rep.*
- *On ajoute rep à la liste repTab.*

La génération est terminée et on a comme liste de réponse :
{ [∃ compA.$\exists s.\exists r.A_4$],

$[\exists$ compA.$\top]$,
$[\exists$ compA.$(A_3 \sqcap A_1)]$ *}*

B.6 Résultats des tests quantitatifs

Dans cette section, on présente les résultats des tests quantitatifs du chapitre 6 (p.89), présenté dans la section 6.2 (p.94).

Dans les tableaux des sections suivantes, on a :

- La première ligne correspond à la variable étudiée,
- Le temps nécessaire à la normalisation de la TBox initiale,
- Le temps nécessaire à la classification de la TBox initiale,
- Le temps nécessaire au calcul des fill-up des réponses,
- Le temps nécessaire au temps de préparation (somme des trois lignes précédentes),
- Le temps nécessaire au calcul du fill-up de la demande,
- Le temps nécessaire à la détermination des liens de subsomption entre la demande et les réponses à partir de la classification,
- Le temps nécessaire au calcul du CCO,
- Le temps nécessaire au calcul du traitement de la demande (somme des trois lignes précédentes),
- Le temps consacrés au calcul de différence pendant le CCO,
- Le nombre de comparaisons effectuées entre réponse,
- Le nombre de différences effectuées entre réponse.

B.6.1 Tests avec variation de la taille de la TBox

On présente dans la table B.3 la moyenne des résultats obtenus pour dix jeux de TBox, réponses et demande générés aléatoirement pour une taille de TBox différente. Les courbes correspondantes et l'analyse des résultats se trouvent dans la section 6.2.1 (p.96).

Taille (nb axiomes)	50	100	150	200	250	300	350	400	450	500
Normalisation	0.01	0.01	0.02	0.02	0.02	0.03	0.03	0.03	0.04	0.04
Classification	0.002	0.004	0.008	0.013	0.019	0.027	0.038	0.051	0.064	0.076
Fup Réponses	0.25	0.40	0.61	0.72	0.98	1.33	1.74	2.23	2.45	3.0
Temps Préparation	0.26	0.42	0.63	0.75	1.02	1.39	1.81	2.31	2.55	3.11
Fup Demande	.0002	.0002	.0002	.0002	.0003	.0003	.0002	.0002	.0002	.0002
Subsomption	0.08	0.10	0.14	0.16	0.25	0.29	0.35	0.36	0.38	0.5
CCO	0.59	1.63	2.64	4.28	6.75	6.53	12.01	15.14	19.27	28.78
Temps Traitement	0.67	1.74	2.78	4.44	7.00	6.82	12.37	15.50	19.65	29.28
Temps Différence	0.58	1.63	2.63	4.27	6.75	6.52	12.01	15.13	19.26	28.77
Nombre de comparaisons	108	108	108	108	108	108	108	108	108	108
Nombre de différences	75.4	84.5	83.6	88.5	88.7	90	92.4	88.4	88.8	92.8

TABLE B.3 – Résultats des tests de l'influence de la taille de la TBox sur le temps de traitement de la demande de l'implémentation du CCO. Toutes les lignes sont en secondes sauf mention contraire.

B.6.2 Tests avec variation de la composition des axiomes

On présente dans la table B.4 la moyenne des résultats obtenus pour dix jeux de TBox, réponses et demande générés aléatoirement pour un pourcentage d'axiomes composés différents. Les courbes correspondantes et l'analyse des résultats se trouvent dans la section 6.2.2 (p.98).

Composition des axiomes (en % de chance d'avoir un axiome composé)	0	10	20	30	40	50	60	70	80	90
Normalisation	0.02	0.04	0.02	0.04	0.02	0.02	0.02	0.02	0.02	0.02
Classification	0.0	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02
Fup Réponses	0.41	1.03	2.45	4.92	7.73	10.80	15.84	22.14	30.30	39.82
Temps Préparation	0.44	1.08	2.45	4.97	7.75	10.82	15.87	22.17	30.32	39.85
Fup Demande	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0003	.0002
Subsomption	0.06	0.27	0.61	1.39	2.48	3.60	5.44	7.49	10.85	14.29
CCO	13.07	3.89	3.56	3.82	4.01	4.10	4.75	5.39	6.64	7.26
Temps Traitement	13.13	4.16	4.17	5.22	6.49	7.72	10.2	12.88	17.39	21.54
Temps Différence	13.06	3.89	3.55	3.81	4.00	4.09	4.74	5.381	6.63	7.24
Nombre de comparaisons	72	144	216	288	360	432	504	576	648	720
Nombre de différences	57.6	109.3	155.8	216.7	258.6	312.0	360.4	416.5	466.8	516.0

TABLE B.4 – Résultats des tests de l'influence de la composition sur le temps de traitement de la demande de l'implémentation du CCO. Toutes les lignes sont en secondes sauf mention contraire.

B.6.3 Tests avec variation du nombre de réponses

On présente dans la table B.5 la moyenne des résultats obtenus pour dix jeux de TBox, réponses et demande générés aléatoirement pour un nombre de réponses différent. Les courbes correspondantes et l'analyse des résultats se trouvent dans la section 6.2.3 (p.100).

Nombre de réponses (avec première réponse générée faisant office de demande de l'utilisateur)	9	19	29	39	49	59	69	79	89	99
Normalisation	0.02	0.02	0.02	0.02	0.02	0.03	0.03	0.03	0.04	0.04
Classification	0.01	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Fup Réponses	0.70	3.25	7.22	15.75	28.26	46.21	72.40	107.1	144.5	209.8
Temps Préparation	0.72	3.28	7.25	15.78	28.29	46.24	72.43	107.2	144.5	209.8
Fup Demande	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002
Subsomption	0.20	0.37	0.67	1.19	1.80	2.72	3.13	4.30	4.56	6.56
CCO	4.49	24.14	71.13	190.3	597.6	622.8	616.0	951.1	1532	3184
Temps Traitement	4.69	24.52	71.80	192	599	626	619.1	955.3	1536	3190
Temps Différence	4.49	24.14	71.1	190.3	597.5	622.7	615.8	950.9	1532	3183
Nombre de comparaisons	108	513	1218	2223	3528	5133	7038	9243	11748	14553
Nombre de différences	84.9	407.8	950.6	1726	2683	4042	5563	7140	8987	10851

TABLE B.5 – Résultats des tests de l'influence du nombre de réponses sur le temps de traitement de la demande de l'implémentation du CCO. Toutes les lignes sont en secondes sauf mention contraire.

B.6.4 Tests avec variation du nombre de composantes

On présente dans la table B.6 la moyenne des résultats obtenus pour dix jeux de TBox, réponses et demande générés aléatoirement pour un nombre de composantes différent. Les courbes correspondantes et l'analyse des résultats se trouvent dans la section 6.2.4 (p.102).

Nombre de composantes	2	4	6	8	10	12	14	16	18	20
Normalisation	0.02	0.04	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Classification	0.02	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02
Fup Réponses	0.83	0.77	0.71	0.71	0.57	0.69	0.80	0.81	0.88	1.18
Temps Préparation	0.87	0.80	0.73	0.76	0.59	0.71	0.83	0.84	0.92	1.22
Fup Demande	.0002	.0002	.0002	.0002	.0003	.0002	.0002	.0003	.0002	.0003
Subsomption	0.17	0.17	0.14	0.16	0.11	0.15	0.20	0.19	0.27	0.28
CCO	24.83	9.91	3.48	3.37	1.49	1.64	1.62	1.62	1.54	1.32
Temps Traitement	25.01	10.09	3.62	3.54	1.60	1.79	1.83	1.81	1.80	1.60
Temps Différence	24.83	9.91	3.48	3.35	1.49	1.61	1.62	1.61	1.53	1.32
Nombre de comparaisons	108	108	108	108	108	108	108	108	108	108
Nombre de différences	87.3	83.3	82.7	84.4	87.2	82	90.8	84.9	90.3	90.2

TABLE B.6 – Résultats des tests de l'influence du nombre de composantes sur le temps de traitement de la demande de l'implémentation du CCO. Toutes les lignes sont en secondes sauf mention contraire.

B.6.5 Tests avec variation du nombre de conjoncts dans les réponses

On présente dans la table B.7 la moyenne des résultats obtenus pour dix jeux de TBox, réponses et demande générés aléatoirement pour un nombre de conjonct dans les réponses différent. Les courbes correspondantes et l'analyse des résultats se trouvent dans la section 6.2.5 (p.104).

Nombre de conjonctions	1	2	3	4	5	6	7	8	9	10
Normalisation	0.02	0.04	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
Classification	0.03	0.01	0.03	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Fup Réponses	0.40	0.56	0.71	1.12	1.48	1.86	2.20	2.51	4.09	4.51
Temps Préparation	0.45	0.61	0.78	1.15	1.48	1.90	2.235	2.54	4.12	4.53
Fup Demande	.0003	.0002	.0002	.0002	.0003	.0002	.0002	.0002	.0002	.0002
Subsomption	0.08	0.13	0.16	0.28	0.43	0.46	0.66	0.76	1.18	1.33
CCO	1.27	3.50	4.21	5.62	9.42	8.42	10.28	13.76	15.98	18.78
Temps Traitement	1.35	3.63	4.38	5.9	9.85	8.87	10.95	14.52	17.17	20.1
Temps Différence	1.23	3.49	4.2	5.62	9.41	8.41	10.28	13.76	15.98	18.77
Nombre de comparaisons	108	108	108	108	108	108	108	108	108	108
Nombre de différences	48.45	73.65	83.25	90.35	93.55	90	96.1	95.75	94.05	98.85

TABLE B.7 – Résultats des tests de l'influence du nombre de conjonctions sur le temps de traitement de la demande de l'implémentation du CCO. Toutes les lignes sont en secondes sauf mention contraire.

B.6.6 Tests avec variation de la profondeur des axiomes et concepts

On présente dans la table B.8 la moyenne des résultats obtenus pour dix jeux de TBox, réponses et demande générés aléatoirement pour une profondeur des axiomes, des réponses et des demandes différente. Les courbes correspondantes et l'analyse des résultats se trouvent dans la section 6.2.6 (p.106).

Profondeur(en nombre d'imbrications de rôle)	1	2	3	4	5	6	7	8	9	10
Normalisation	0.02	0.03	0.03	0.06	0.06	0.03	0.07	0.06	0.04	0.05
Classification	0.01	0.02	0.04	0.05	0.05	0.04	0.05	0.06	0.06	0.04
FuP Réponses	0.71	1.83	2.39	3.11	3.98	4.57	4.42	5.47	4.59	6.02
Temps Préparation	0.74	1.89	2.46	3.22	4.09	4.64	4.55	5.59	4.69	6.12
Fup Demande	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002	.0002
Subsomption	0.16	0.38	0.57	0.58	0.86	1.06	0.87	1.34	1.15	1.47
CCO	4.42	7.68	8.44	8.05	9.1	8.63	10.13	12.82	10.26	11.98
Temps Traitement	4.58	8.01	9.01	8.63	9.96	9.7	11.00	14.16	11.41	13.45
Temps Différence	4.41	7.68	8.44	8.05	9.10	8.63	10.12	12.82	10.26	11.97
Nombre de comparaisons	108	108	108	108	108	108	108	108	108	108
Nombre de différences	84.9	407.8	950.6	1726	2683	4042	5563	7140	8987	10851

TABLE B.8 – Résultats des tests de l'influence de la profondeur sur le temps de traitement de la demande de l'implémentation du CCO.Toutes les lignes sont en secondes sauf mention contraire.

B.7 Comparaison temporelle du CCO avec un CCO sans calcul de subsomption

Pour ce test évoqué dans la section 6.2.7 (p.108), on veut identifier la différence de performance entre un CCO qui va comparer les rest et les miss avec des calculs de subsomption, puis de taille si nécessaire, à un CCO qui va uniquement comparer avec la taille. Les caractéristiques communes aux deux tests sont répertoriées dans le tableau B.9 (p.169). Les performances des temps de traitements sont comparées dans le tableau B.11 (p.169). Les résultats obtenus sont cependant différents. Avec le calcul de subsomption, on obtient l'ordre et le score suivants pour les 9 réponses :
(r3 :15), (r0 :8), (r1 :8), (r7 :5), (r6 :2), (r2 :-2), (r8 :-9), (r5 :-13), (r4 :-14)
Sans calcul de subsomption, on obtient l'ordre et le score suivants pour les 9 réponses :
(r3 :15), (r0 :8), (r7 :7), (r1 :6), (r6 :2), (r2 :-2), (r4 :-12), (r5 :-13), (r8 :-15)
Pour le test avec 99 réponses, les classement sont présentés dans le tableau B.10 (p.169).

On constate des différences dans les résultats dans le tableau B.10, mais le classement général est assez proche du résultat optimal. Par exemple, les dix premières réponses

Variable	Test1	Test 2
Taille de la TBox (en nb d'axiomes)	500	1000
Probabilité d'avoir un axiome contenant, à gauche, des restrictions existentielles ou des conjonctions	20%	20%
Nombre de réponses	9	49
Nombre de composantes	3	3
Nombre de conjoncts dans un axiome ou une réponse	3	3
Profondeur maximale d'un axiome ou d'une réponse (en nombre d'imbrication de rôle)	1	1

TABLE B.9 – Caractéristiques des tests pour la comparaison du CCO avec et sans calcul de subsomption pour comparaison des rest et miss.

Position	Nombre
Même position	27
Une place de différence	14
Jusqu'à trois place de différence	6
Jusqu'à cinq place de différence	1
Jusqu'à dix place de différence	1
Plus de dix place de différence	0

TABLE B.10 – Différence de classement pour la comparaison du CCO avec et sans calcul de subsomption pour comparaison des rest et miss.

proposées (même si dans un ordre légèrement différent) sont les mêmes. C'est donc une approximation potentielle envisageable. Du point de vue des performances on constate dans la table B.11 que l'approximation n'utilisant que les comparaisons de longueurs est bien plus performante pour traiter la demande d'un utilisateur.

Calcul de subsomption	Test 1 : avec	Test 1 : sans	Test 2 : avec	Test 2 : sans
Fup Demande	.0001	.0003	0.0004	0.0002
Subsomption	0.42	0.4	2.28	2.16
CCO	36.72	0.99	1621	6.13
Temps Traitement	37.13	1.39	1623	8.29
Temps Différence	36.72	0.981	1621	6.04

TABLE B.11 – Temps de traitement (en secondes) selon l'utilisation de calcul de subsomption dans la comparaison du rest et du miss pendant le CCO.

Annexe C

Preuves des résultats théoriques

C.1 Preuve de la proposition 2 (p.51)

Dans cette section se trouvent les preuves des propriétés du TSO (voir 23 (p.49)) et de son algorithme (algorithme 3 (p.50)). On rappelle tout d'abord les propriétés du TSO :

Proposition 2. *Soient C et D deux concepts. On a :*
a. $C \vartriangle D$ est unique et existe toujours.
b. L'algorithme 3 (p.50) se termine et produit un résultat unique.
c. $C \vartriangle D = tso(C, D)$ (justesse de l'algorithme 3 (p.50)).
d. Calculer $tso(C, D)$ est PTIME par rapport aux tailles de C et D.

Démonstration. a. **Existence et unicité de** $C \vartriangle D$ (pour le cas $\mathsf{br} \neq \emptyset$)

On montre que $C \vartriangle D$ existe toujours. Puisque br_E est un sous-ensemble de br_C, l est toujours possible de construire E tel que $C \sqsubseteq E$. Alors soit E est minimal par rapport à $\sqsubseteq$, soit on peut le rendre plus petit par rapport à $\sqsubseteq$ en remplaçant ses conjonctions de type $\exists r.A \sqcap \exists r.B$ en $\exists r.(A \sqcap B)$ jusqu'à ce qu'on ne puisse plus garantir $C \sqsubseteq E$. Donc $C \vartriangle D$ existe toujours.

On montre maintenant l'unicité de $C \vartriangle D$ par l'absurde. On suppose qu'il y a deux concepts E_1 et E_2, avec $E_1 \neq E_2$ dans $C \vartriangle D$. Puisque $\mathsf{br}_{E_1} = \mathsf{br}_{E_2}$ et $E_1 \neq E_2$, on a $E_1 \not\equiv E_2$. Alors la seule situation possible est que E_1 et E_2 ne sont pas comparable par rapport $\sqsubseteq$. Cela implique que $E_1 \sqcap E_2 \sqsubseteq E_1$ et $E_1 \sqcap E_2 \sqsubseteq E_2$. Dans le même temps, on a $\mathsf{br}_{E_1 \sqcap E_2} = \mathsf{br}_{E_1} = \mathsf{br}_{E_2}$ et $C \sqsubseteq E_1 \sqcap E_2$. Cela signifie que ni E_1 ni E_2 sont minimaux par rapport à $\sqsubseteq$ en ayant les mêmes propriétés. Alors, $C \vartriangle D$ est unique.

b. **Terminaison**

Par récurrence, on montre que l'algorithme 3 (p.50) se termine toujours et génère un produit dont la taille, appelée s_{out}, est strictement inférieure à la taille combinée des entrées C et D, appelée s_{in} et défini telle que $s_{in} = \mathsf{size}(C) + \mathsf{size}(D)$. La récurrence est faite sur s_{in}.

D'après la définition 1 (p.14), on a $\mathsf{size}(C) \geq 1$ et $\mathsf{size}(D) \geq 1$.
Cas de base : $s_{in} = \mathsf{size}(C) + \mathsf{size}(D) = 1 + 1 = 2$

— Lignes 1 et 2 : l'algorithme s'arrête avec $s_{out} = \mathsf{size}(\top) = 1 < 2 = s_{in}$
— Lignes 4 à 10 : cas impossible quand $\mathsf{size}(C) = 1$.
— Lignes 11 et 12 : cas impossible quand $\mathsf{size}(D) = 1$.
— Lignes 13 à 19 : cas impossible quand $\mathsf{size}(C) = \mathsf{size}(D) = 1$.
— Lignes 20 et 21 : l'algorithme s'arrête avec $s_{out} = \mathsf{size}(C) = 1 < 2 = s_{in}$

Cas général : L'hypothèse de récurrence (HR) dit qu'il y a $n \geq 3$ tel que l'algorithme s'arrête avec $s_{out} < s_{in}$ pour tout $s_{in} = \mathsf{size}(C) + \mathsf{size}(D) \leq n$.

On suppose maintenant que $s_{in} = \mathsf{size}(C) + \mathsf{size}(D) = n+1$. Donc $\mathsf{size}(C) = n+1-\mathsf{size}(D)$ et $\mathsf{size}(D) = n+1-\mathsf{size}(C)$ (avec $\mathsf{size}(C) \geq 1$ et $\mathsf{size}(D) \geq 1$).

— Ligne 1 et 2 : l'algorithme s'arrête avec $s_{out} = \mathsf{size}(\top) = 1 < n+1 = s_{in}$ (puisque $n \geq 3$).
— Lignes 4 à 10
  — Puisque $C = C_1 \sqcap \ldots \sqcap C_k$, avec $k \geq 2$, il y a (d'après la définition 1 (p.14)) :
    $\mathsf{size}(C) = (\sum_{i=1}^{k} \mathsf{size}(C_i)) + k - 1$, avec $\mathsf{size}(C_i) \geq 1, \forall 1 \leq i \leq k$
    Puisque $\mathsf{size}(C) = n+1-\mathsf{size}(D)$, on a $\forall 1 \leq i \leq k$:
    $\mathsf{size}(C_i) = n+1-\mathsf{size}(D)$
    $-\mathsf{size}(C_1) - \ldots - \mathsf{size}(C_{i-1}) - \mathsf{size}(C_{i+1}) - \ldots - \mathsf{size}(C_k) - k + 1$
    Puisque $\forall 1 \leq i \leq k, \mathsf{size}(C_i) \geq 1$, on a :
    $\mathsf{size}(C_i) \leq n+1-\mathsf{size}(D)-(k-1)-k+1 = n-\mathsf{size}(D)-2.k+3$
  — A la ligne 5, on a $k \geq 2$ appels récursifs $\mathrm{tso}(C_i, D)$ avec $\mathsf{size}(C_i) \leq n-\mathsf{size}(D)-2.k+3$ et $\mathsf{size}(D) \geq 1$. Pour chaque appel récursif $\mathrm{tso}(C_i, D)$, on note la taille combinée de ses entrées s_{in}^{rc} et la taille de sa sortie s_{out}^{rc}. Ensuite, pour chaque appel récursif $\mathrm{tso}(C_i, D)$ on a,
    ◦ $s_{in}^{rc} = \mathsf{size}(C_i) + \mathsf{size}(D) \leq n - \mathsf{size}(D) - 2.k + 3 + \mathsf{size}(D) = n - 2.k + 3 \leq n - 1 \leq n$ (puisque $k \geq 2$).
    ◦ Ainsi, avec l'HR, l'appel récursif s'arrête et $s_{out}^{rc} < s_{in}^{rc} \leq n - 2.k + 3$, c'est-à-dire $s_{out}^{rc} \leq n - 2.k + 2$
  — A la ligne 7, puisque tous les appels récursifs s'arrêtent, l'algorithme s'arrête avec $s_{out} \leq k * (n - 2.k + 2) + k - 1$ (puisqu'il y a k concepts C_i et $k-1$ constructeurs $\sqcap$ in C).
    ◦ Maintenant, $k \leq (n+1)/2$. En effet, $\mathsf{size}(C) \leq n$ (puisque $\mathsf{size}(C) + \mathsf{size}(D) = n+1$ et $\mathsf{size}(D) \geq 1$). Et $\mathsf{size}(C) \geq 2.k - 1$ (puisque $\mathsf{size}(C) = (\sum_{i=1}^{k} \mathsf{size}(C_i)) + k - 1$ et $\mathsf{size}(C_i) \geq 1, \forall 1 \leq i \leq k$).
    ◦ Donc $s_{out} \leq (n+1)/2 * (n - 2.(n+1)/2 + 2) + (n+1)/2 - 1 = n$
    So $s_{out} < n + 1 = s_{in}$.
  — A la ligne 9, l'algorithme s'arrête avec $s_{out} = \mathsf{size}(\top) = 1 < n+1 = s_{in}$ (puisque $n \geq 3$).
— Lignes 11 et 12 :
  — Ici on se focalise sur $D = D_1 \sqcap \ldots D_m$, avec $m \geq 2$. Le même raisonnement que celui utilisé pour la ligne 4 avec D à la place de C amène :
    ◦ $\forall 1 \leq j \leq m$:
    $\mathsf{size}(D_j) = n+1-\mathsf{size}(C)$
    $-\mathsf{size}(D_1) - \ldots - \mathsf{size}(D_{j-1}) - \mathsf{size}(D_{j+1}) - \ldots - \mathsf{size}(D_m) - m + 1,$

- $\forall 1 \leq j \leq m : \mathsf{size}(D_j) \leq n - \mathsf{size}(C) - 2.m + 3$ et
- $m \leq (n+1)/2$

— Maintenant nous montrons que l'ensemble des appels récursifs imbriqués
$\mathrm{tso}(\ldots(\mathrm{tso}(\mathrm{tso}(C, D_1), D_2), \ldots), D_m)$
se termine avec $s_{out} < s_{in}$.

Formellement, cela nécessiterait une preuve par récurrence. Cependant, pour des raisons de simplicité, nous proposons une ébauche de preuve. On utilise les notations $s_{in}^{rc,j}$ et $s_{out}^{rc,j}$ pour la taille combinée des entrées et la taille de la sortie de l'appel récursif imbriqué possédant D_j comme second argument.

- Tout d'abord l'appel récursif $\mathrm{tso}(C, D_1)$ est tel que $s_{in}^{rc,1} = \mathsf{size}(C) + \mathsf{size}(D_1)$.
  Alors : $s_{in}^{rc,1} \leq \mathsf{size}(C) + n - \mathsf{size}(C) - 2.m + 3 = n - 2.m + 3$
  Alors : $s_{in}^{rc,1} \leq n - 1 \leq n$ (since $m \geq 2$).
  D'après l'HR, l'appel récursif s'arrête avec $s_{out}^{rc,1} < s_{in}^{rc,1} \leq n - 2.m + 3$.
- L'appel récursif $\mathrm{tso}(\mathrm{tso}(C, D_1), D_2)$ est tel que
  $s_{in}^{rc,2} = \mathsf{size}(s_{out}^{rc,1}) + \mathsf{size}(D_2)$
  Et ensuite :
  $s_{in}^{rc,2} < \mathsf{size}(s_{in}^{rc,1}) + \mathsf{size}(D_2)$
  $s_{in}^{rc,2} < \mathsf{size}(s_{in}^{rc,1}) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_3) - \ldots - \mathsf{size}(D_m) - m + 1$
  $s_{in}^{rc,2} \leq \mathsf{size}(s_{in}^{rc,1}) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_3) - \ldots - \mathsf{size}(D_m) - m$
  $s_{in}^{rc,2} \leq \mathsf{size}(C) + \mathsf{size}(D_1) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_3) - \ldots - \mathsf{size}(D_m) - m$
  $s_{in}^{rc,2} \leq n + 1 - \mathsf{size}(D_3) - \ldots - \mathsf{size}(D_m) - m$
  $s_{in}^{rc,2} \leq n + 1 - (m-2) - m = n - 2.m + 3$ (puisque $\mathsf{size}(D_j) \geq 1, \forall 1 \leq j \leq m$)
  D'après l'HR, l'appel récursif s'arrête avec $s_{out}^{rc,2} < s_{in}^{rc,2} \leq n - 2.m + 3$.
- L'appel récursif $\mathrm{tso}(\mathrm{tso}(\mathrm{tso}(C, D_1), D_2), D_3)$est tel que
  $s_{in}^{rc,3} = \mathsf{size}(s_{out}^{rc,2}) + \mathsf{size}(D_3)$
  Et ensuite :
  $s_{in}^{rc,3} < \mathsf{size}(s_{in}^{rc,2}) + \mathsf{size}(D_3)$
  $s_{in}^{rc,3} < \mathsf{size}(s_{in}^{rc,2}) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_2) - \mathsf{size}(D_4) - \ldots - \mathsf{size}(D_m) - m + 1$
  $s_{in}^{rc,3} \leq \mathsf{size}(s_{in}^{rc,2}) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_2) - \mathsf{size}(D_4) - \ldots - \mathsf{size}(D_m) - m$
  $s_{in}^{rc,3} \leq \mathsf{size}(s_{out}^{rc,1}) + \mathsf{size}(D_2) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_2) - \mathsf{size}(D_4) - \ldots - \mathsf{size}(D_m) - m$
  $s_{in}^{rc,3} < \mathsf{size}(s_{in}^{rc,1}) + \mathsf{size}(D_2) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_2) - \mathsf{size}(D_4) - \ldots - \mathsf{size}(D_m) - m$
  $s_{in}^{rc,3} < \mathsf{size}(C) + \mathsf{size}(D_1) + \mathsf{size}(D_2) + n + 1 - \mathsf{size}(C) - \mathsf{size}(D_1) - \mathsf{size}(D_2) - \mathsf{size}(D_4) - \ldots - \mathsf{size}(D_m) - m$
  $s_{in}^{rc,3} \leq \mathsf{size}(C) + \mathsf{size}(D_1) + \mathsf{size}(D_2)$

$+n+1-\mathsf{size}(C)-\mathsf{size}(D_1)-\mathsf{size}(D_2)-\mathsf{size}(D_4)-\ldots-\mathsf{size}(D_m)-m-1$
$s_{in}^{rc,3} \leq n+1-\mathsf{size}(D_4)-\ldots-\mathsf{size}(D_m)-m-1$
$s_{in}^{rc,3} \leq n+1-(m-2)-m = n-2.m+3$ (since $\mathsf{size}(D_j) \geq 1, \forall 1 \leq j \leq m$)
D'après l'HR, l'appel récursif s'arrête avec $s_{out}^{rc,3} < s_{in}^{rc,3} \leq n-2.m+3$.

- Ainsi de suite.

Une sous-preuve formelle par récurrence montrerait que chaque appel récursif de $\mathrm{tso}(\ldots(\mathrm{tso}(\mathrm{tso}(C, D_1), D_2), \ldots), D_m)$
s'arrête avec $s_{out}^{rc,j} < s_{in}^{rc,j} \leq n-2.m+3 < n+1$, $\forall 1 \leq j \leq m$. Ainsi, à la ligne 12, l'algorithme s'arrête avec $s_{out} < s_{in} = n+1$.

— Lignes 13 à 19 :
  — Ici on a $C = \exists r.C'$ et $D = \exists r.D'$. Donc $s_{in} = n+1 = \mathsf{size}(C) + \mathsf{size}(D) \geq 4$.
  — A la ligne 14, l'appel récursif $\mathrm{tso}(C', D')$ est tel que $s_{in}^{rc} = \mathsf{size}(C') + \mathsf{size}(D') = n+1-2 = n-1 \leq n$. D'après l'HR, l'appel récursif s'arrête avec $s_{out}^{rc} < s_{in}^{rc} = n-1$. c'est-à-dire $s_{out}^{rc} \leq n-2$.
  — Puisque l'instruction de la lignes 15 à 19 s'arrête, alors l'algorithme s'arrête. De plus :
    - A la ligne 16, $s_{out} = \mathsf{size}(\top) = 1 < s_{in}$ (puisque $s_{in} \geq 4$).
    - A la ligne 18, $s_{out} = 1 + s_{out}^{rc} \leq 1+n-2 = n-1$. Donc $s_{out} < s_{in} = n+1$.
— Lignes 20 et 21 : l'algorithme s'arrête avec $s_{out} = \mathsf{size}(C) < \mathsf{size}(C) + \mathsf{size}(D) = s_{in}$ (puisque $\mathsf{size}(D) \geq 1$).

Enfin, le fait que l'algorithme 3 (p.50) génère un résultat unique provient du fait que chaque sortie est définie de manière unique (voir lignes 2, 7, 9, 12, 16, 18 et 21) et qu'il n'y a aucune étape indéterminée (c'est-à-dire aucune instruction impliquant un choix)

c. **Justesse**

La preuve de justesse s'effectue en trois étapes

Étape 1 À partir de la caractérisation de la subsomption dans $\mathcal{EL}$ sans TBox donné dans Baader et al. (1999), nous dérivons une caractérisation de la subsomption dans $\mathcal{EL}$ en terme de sous description (dans le corollaire 1 (p.175)).

Étape 2 à partir du corollaire 1 (p.175) est dérivée une caractérisation du TSO en terme de sous description.

Étape 3 une preuve par récurrence de la justesse est donnée en utilisant la caractérisation obtenue à l'étape 2.

Étape 1

Rappelons la caractérisation de la subsomption dans $\mathcal{EL}$ par rapport à une TBox vide (voir Baader et al. (1999)). Commençons par l'arbre de description d'un concept.

Definition 35 (Arbre de description (voir Baader et al. (1999)))**.** *Soit C un concept $\mathcal{EL}$. Son arbre de description $\mathcal{G}_C$ est un arbre représenté par le quadruplé (V, E, v_0, l) où V est un ensemble de sommet, $E \subseteq V \times \mathbf{r} \times V$ est l'ensemble des arrêtes, $v_0 \in V$ est la racine et $l : V \to 2^{\mathbf{C}}$ une fonction qui nomme les sommets avec des ensembles de nom de concept (L'ensemble vide étant $\top$).*

Baader et al. (1999) explique que $\mathcal{G}_C$ est unique, comment obtenir $\mathcal{G}_C$ à partir de C, ou C à partir de $\mathcal{G}_C$, et que $C_{\mathcal{G}_C} \equiv C$. On y trouve également la définition d'un

homomorphisme entre deux arbres de description rappelée ci-dessous. Ils caractérisent également la subsomption de concept exprimé avec seulement des concepts primitifs.

Definition 36 (homomorphisme des arbres de description (voir Baader et al. (1999)))**.** *Un homomorphisme à partir d'un arbre de description $\mathcal{G}_D = (V_D, E_D, w_0, l_D)$ vers un arbre de description $\mathcal{G}_C = (V_C, E_C, v_0, l_C)$ est le mapping $\varphi : V_D \to V_C$ tel que (1) $\varphi(w_0) = v_0$, (2) $l_D(w) \subseteq l_C(\varphi(w)), \forall w \in V_D$, et (3) $(\varphi(w_1), r, \varphi(w_2)) \in E_C, \forall (w_1, r, w_2) \in E_D$.*

Theoreme 1 (Voir Baader et al. (1999))**.** *Soient C et D deux concepts et $\mathcal{G}_C$ et $\mathcal{G}_D$ leur arbre de description correspondant. Alors :*

$C \sqsubseteq D$ ssi il existe un homomorphisme φ de $\mathcal{G}_D$ vers $\mathcal{G}_C$.

Notons que, avec une TBox $\mathcal{T}$, le théorème 1 (p.175) est également valide avec avec $\sqsubseteq_{\mathcal{T}}$ à la place de $\sqsubseteq$ quand $(C, D) \in (\mathcal{T}_{\mathcal{EL}}^{\mathsf{prim}})^2$. Le corollaire 1 (p.175) reformule le théorème 1 (p.175) en terme de sous-description.

Corollaire 1. *Soient C et D deux concepts. Alors :*

$C \sqsubseteq D$ ssi il existe $D' \in \mathsf{subd}_C$ tel que D' puisse être obtenu en appliquantdans D n'importe où, zéro ou plusieurs fois les deux règles syntaxiques suivantes (r1) et (r2) qui remplace le côté gauche par le côté droit :

(r1) $\exists r.G \sqcap \exists r.H \rightsquigarrow \exists r.(G \sqcap H)$

(r2) $\top \rightsquigarrow H$

avec r n'importe quel rôle, et G et H n'importe quel concept.

Démonstration. On note :

Ⓐ $C \sqsubseteq D$

Ⓑ il existe un homomorphisme φ de $\mathcal{G}_D$ vers $\mathcal{G}_C$.

Ⓒ il existe $D' \in \mathsf{subd}_C$ tel que D' peut être obtenu en appliquant dans D n'importe où zéro ou plusieurs fois les règles (r1) et (r2).

Nous montrons maintenant la preuve du corollaire 1 (p.175) en montrant que Ⓒ implique Ⓐ et Ⓑ implique Ⓒ.

Preuve que Ⓒ implique Ⓐ

Cela provient des deux faits suivants :

— $C \sqsubseteq D'$ puisque $D' \in subd_C$

— et $D' \sqsubseteq D$ puisque appliquer (r1) ou (r2) à D produit clairement un concept plus spécifique.

Preuve que Ⓑ implique Ⓒ

On prouve ce résultat par récurrence sur $\mathsf{size}(D)$.

— Cas initial : On suppose $\mathsf{size}(D) = 1$ (c'est-à-dire $D \in \mathbf{C} \cup \{\top\}$) et Ⓑ. Montrons Ⓒ.

Par définition de $\mathcal{G}_D = (V_D, E_D, w_0, l_D)$, il y a : $l_D(w_0) = \{D\}$ si $D \in \mathbf{C}$ ou $l_D(w_0) = \emptyset$ si $D = \top$. D'après Ⓑ et la définition 36 (p.175), on a (avec φ un homomorphisme de $\mathcal{G}_D$ vers $\mathcal{G}_C$) :

— si $D \in \mathbf{C}$: $l_D(w_0) = \{D\} \subseteq l_C(\varphi(w_0)) = l_C(v_0)$. Donc $D' = D$ assure Ⓒ (c'est-à-dire D' peut être obtenu sans appliquer ni (r1) ni (r2) et $D' \in \mathsf{subd}_C$).

— si $D = \top$: toute sous-description de C peut définir D' (en appliquant la règle (r2) to D) de manière à ce que $\textcircled{\diamondsuit}$ soit vrai.

— Cas général : on suppose $\mathsf{size}(D) = n + 1$ avec $n \geq 1$. L'hypothèse de récurrence (HR) est que pour tout concept dont la taille est au maximum n, on suppose : $\textcircled{\square}$ implique $\textcircled{\diamondsuit}$. On suppose alors $\textcircled{\square}$ pour D. Montrons $\textcircled{\diamondsuit}$.

— Cas 1 : $D = D_1 \sqcap D_2$ On a $\textcircled{\star}$ $\mathsf{size}(D_1) \leq n - 1$ et $\mathsf{size}(D_2) \leq n - 1$. As $D_1 \in \mathsf{subd}_D$ et $D_2 \in \mathsf{subd}_D$ et il existe un homomorphisme φ de $\mathcal{G}_D$ vers $\mathcal{G}_C$ (puisqu'on suppose $\textcircled{\square}$ pour D), alors $\textcircled{\star\star}$ φ est un homomorphisme de $\mathcal{G}_{D_1}$ vers $\mathcal{G}_C$ et un homomorphisme de $\mathcal{G}_{D_2}$ vers $\mathcal{G}_C$. Grâce à $\textcircled{\star}$ et $\textcircled{\star\star}$, l'HR peut être appliquée : il existe $D'_1 \in \mathsf{subd}_C$ (resp. $D'_2 \in \mathsf{subd}_C$) qui peut être obtenu en appliquant (r1) ou (r2) zéro plusieurs fois dans D n'importe où. D' peut ensuite être construit de manière à ce qu'il soit un arbre de description $\mathcal{G}_{D'} = (V_{D'}, E_{D'}, v_0, l_{D'})$ tel que :

$V_{D'} = V_{D'_1} \cup V_{D'_2}$,
$E_{D'} = E_{D'_1} \cup E_{D'_2}$,
et $l_{D'} : V_{D'} \to 2^{\mathbf{C}}$

$$v' \mapsto l_{D'}(v') \text{ avec}$$

$$l_{D'}(v') = \begin{cases} l_{D'_1}(v') & \text{si } l_{D'_2}(v') \text{ non défini} \\ l_{D'_2}(v') & \text{si } l_{D'_1}(v') \text{ non défini} \\ l_{D'_1}(v') \cup l_{D'_2}(v') & \text{snon} \end{cases}$$

Dans $\mathcal{G}_{D'}$, on a :
$V_{D'} \subseteq V_C$ puisque $V_{D'_1} \subseteq V_C$ et $V_{D'_2} \subseteq V_C$,
$E_{D'} \subseteq E_C$ puisque $E_{D'_1} \subseteq E_C$ et $E_{D'_2} \subseteq E_C$,
et $\forall v' \in V_{D'}, l'_{D'}(v') \subseteq l_c(v')$.

Donc $D' \in \mathsf{subd}_C$. Les deux arguments suivants montre que D' peut être obtenu en appliquant (r1) ou (r2) zéro plusieurs fois dans D n'importe où :

- D'_1 (resp. D'_2) est une sous-description de D' et peut être obtenue en appliquant (r1) ou (r2) zéro plusieurs fois dans D_1 (resp. in D_2) n'importe où, qui est elle-même même une sous-description de of D. Alors, toutes les branches $\mathcal{G}_{D'}$ dont les arrêtes $(v_1, r, v_2) \in E_{D'}$ viennent uniquement des arrêtes de $\mathcal{G}_{D'_1}$ ou $\mathcal{G}_{D'_2}$ qui correspondent aux sous-descriptions de D'_1 ou D'_2 qui peuvent être obtenues en appliquant (r1) ou (r2) zéro plusieurs fois dans D n'importe où.
- Maintenant pour tout $(v_1, r, v_2) \in E_{D'}$, si (v_1, r, v_2) est à la fois dans $E_{D'_1}$ et $E_{D'_2}$, cela signifie qu'il existe $(w_1, r,\ w_2) \in E_{D_1}$ et $(w_3, r, w_4) \in E_{D_2}$ (avec $w_2 \neq w_4$) tels que $(v_1, r, v_2) = (\varphi(w_1), r, \varphi(w_2)) = (\varphi(w_3), r, \varphi(w_4))$. Maintenant, en terme de concept, cela équivaut appliquer la règle (r2) dans D aux paires de restrictions existentielles qui correspondent à $(w_1, r,$ $w_2)$ et (w_3, r, w_4). Alors, toutes les branches $\mathcal{G}_{D'}$ pour lesquels une arrête $(v_1, r, v_2) \in E_{D'}$ vient à la fois de $\mathcal{G}_{D'_1}$ et $\mathcal{G}_{D'_2}$ correspondent aux sous-descriptions de D_1 ou D_2 de D sur lesquelles (r2) a été appliquée. Comme D_1 et D_2 ont été obtenu par l'application de (r1) ou (r2) zéro plusieurs

fois dans D n'importe où, alors il en va de même pour D'.

— Cas 2 : $D = \exists r.D_1$.

Clairement, $\circledast$ $\mathsf{size}(D) = n$ et $D_1 \in \mathsf{subd}_D$.

D'après $\boxed{\square}$, il existe un homomorphisme φ de $\mathcal{G}_D$ vers $\mathcal{G}_C$. Alors, il existe un concept C' tel que $\exists r.C' \in \mathsf{subd}_C$ avec $\circledast\circledast$ φ qui est un homomorphisme de $\mathcal{G}_{D_1}$ vers $\mathcal{G}_{C'}$. Supposons que v_0 est la racine de $\mathcal{G}_C$ et (v_0, r, v_1) est l'arrête de $\mathcal{G}_C$ commençant au sommet v_0 et correspondant à la restriction existentielle $\exists r.C'$ (c'est-à-dire v_1 est à la racine de $\mathcal{G}_{C'}$).

Avec $\circledast$ et $\circledast\circledast$, on peut appliquer l'HR : il existe $D'_1 \in \mathsf{subd}_{C'}$ qui peut être obtenu en appliquant (r1) ou (r2) zéro ou plusieurs fois dans D_1 n'importe où. D' peut alors être construit de manière à être un arbre de description $\mathcal{G}_{D'} = (V_{D'}, E_{D'}, v_0, l_{D'})$ tel que :

$V_{D'} = V_{D'_1} \cup \{v_0\}$,

$E_{D'} = E_{D'_1} \cup (v_0, r, v_1)$,

et $l_{D'} : V_{D'} \to 2^{\mathbf{C}}$

$$v' \mapsto l_{D'}(v') = \begin{cases} l_{D'_1}(v') & \text{si } v' \in V_{C'} \\ l_D(w_0) & \text{si } v' = v_0 = \varphi(w_0) \end{cases}$$

En d'autres termes $D' = \exists r.D'_1$. Il est alors facile de voir que $D' \in \mathsf{subd}_C$.

De plus, D' peut être obtenu en appliquant (r1) or (r2) zéro plusieurs fois dans D n'importe où puisque D'_1 peut être obtenu en appliquant (r1) or (r2) zéro plusieurs fois dans D_1 n'importe où, c'est alors une sous-description de D.

□

Étape 2

Rappelons la définition du TSO :

$$C \vartriangle D = \begin{cases} Min_{\sqsubseteq}\{\text{concept } E \mid ① \; C \sqsubseteq E \text{ et } ② \; \mathsf{br}_E = \mathsf{br}\} & \text{if } \mathsf{br} \neq \emptyset \\ \top & \text{if } \mathsf{br} = \emptyset \end{cases}$$

avec $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2 ... \exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid$
$\exists S' = \exists r_1 ... \exists r_n.\exists r_{n+1} ... \exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

Dans le cas où $\mathsf{br} \neq \emptyset$, ② implique $\mathsf{br}_E \subseteq \mathsf{br}_C$. Maintenant le corollaire 1 (p.175) dit que : ① $C \sqsubseteq E$ ssi il existe $E' \in \mathsf{subd}_C$ tel que E' peut être obtenu en appliquant (r1) or (r2) zéro plusieurs fois dans E n'importe où , avec :

(r1) $\exists r.G \sqcap \exists r.H \rightsquigarrow \exists r.(G \sqcap H)$

(r2) $\top \rightsquigarrow H$

Puisque $\mathsf{br}_E \subseteq \mathsf{br}_C$, cela signifie que E' est obtenu depuis E sans appliquer (r2). Supposons maintenant que E' a été obtenu à partir de E en appliquant au moins une fois (r1). Cela signifie qu'il y a dans E une sous-description $S_1 = \exists r_1 ... \exists r_{i-1}.(\exists r_i.G \sqcap \exists r_i.H)$ et il y a dans C une sous-description $S_2 = \exists r_1 ... \exists r_i.J$, avec J obtenu à partir de $G \sqcap H$ en appliquant (r1) zéro plusieurs fois. Il est alors clair que $S_2 \sqsubseteq S_1$. Mais cela contredit le fait que E est minimal par rapport à $\sqsubseteq$ tel que ① et ②. Donc $E' = E$ et alors $E \in \mathsf{subd}_C$. Puisque $E \in \mathsf{subd}_C$ implique $C \sqsubseteq E$, on a alors :

$$C \vartriangle D = \begin{cases} Min_{\sqsubseteq}\{E \mid \text{①'}\ E \in \mathsf{subd}_C \text{ et } \text{②}\ \mathsf{br}_E = \mathsf{br}\} & \text{if } \mathsf{br} \neq \emptyset \\ \top & \text{if } \mathsf{br} = \emptyset \end{cases}$$
avec $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid$
$\exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

Puisque un ensemble de branches détermine une sous-description unique pour le concept associé, on a :

$$C \vartriangle D = \begin{cases} E \mid \text{①'}\ E \in \mathsf{subd}_C \text{ et } \text{②}\ \mathsf{br}_E = \mathsf{br} & \text{if } \mathsf{br} \neq \emptyset \\ \top & \text{if } \mathsf{br} = \emptyset \end{cases}$$
avec $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid$
$\exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

Étape 3

Sur la base de la précédente caractérisation du TSO, nous montrons maintenant que $C \vartriangle D = \text{tso}(C, D)$ par récurrence sur la taille combinée des entrées C et D : cette taille combinée est nommée s_{in} et initialisée à $\mathsf{size}(C) + \mathsf{size}(D)$.

Cas de base : $s_{in} = \mathsf{size}(C) + \mathsf{size}(D) = 1 + 1 = 2$

- Lignes 1 et 2 :
  - Si $C = D = \top$, $\mathsf{br} = \{\top\} \setminus \{\top\} = \emptyset$ et ainsi $C \vartriangle D = \top$, et $\text{tso}(C, D) = \top$ également.
  - Si $C = \top$ (et $C \neq D$), $\mathsf{br} = \{\top\} \setminus (\mathsf{br}_D \cup \{\top\}) = \emptyset$ et ainsi $C \vartriangle D = \top$, et $\text{tso}(C, D) = \top$ également.
  - Si $C = D$ (et $C \neq \top$), $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_C \cup \emptyset) = \emptyset$ et ainsi $C \vartriangle D = \top$, et $\text{tso}(C, D) = \top$ également.
- Lignes 4 à 10 : ce cas ne s'applique pas car il s'applique uniquement quand $\mathsf{size}(C) \geq 3$.
- Lignes 11 et 12 : ce cas ne s'applique pas car il s'applique uniquement quand $\mathsf{size}(D) \geq 3$.
- Lignes 13 à 19 : ce cas ne s'applique pas car il s'applique uniquement quand $\mathsf{size}(C) + \mathsf{size}(D) \geq 4$.
- Lignes 20 et 21 :

  Dans ce cas, $C \neq \top$, $C \neq D$, C et D ne sont pas des conjonctions et C et D ne sont pas des restrictions existentielles avec le même rôle initial. Alors C et D peuvent être soit des restrictions existentielles avec un rôle initial différent et/ou des noms de concepts. Dans tous les cas, cela mène à $\mathsf{br}_C \cap \mathsf{br}_D = \emptyset$, $\mathsf{subd}_C \cap \mathsf{subd}_D = \emptyset$ et il n'y a aucun couple de branches qui commencent par le même rôle. Alors $\mathsf{br} = \mathsf{br}_C$ avec

  $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid$
  $\exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

  So $C \vartriangle D = C$. De plus, $\text{tso}(C, D) = C$.

Cas général : $s_{in} = \mathsf{size}(C) + \mathsf{size}(D) = n_r + 1$, avec $n_r \geq 3$
On rappelle que dans ce cas, l'hypothèse de récurrence (HR) est : $\forall C'$ et D' avec $s'_{in} =$

$\mathsf{size}(C') + \mathsf{size}(D') \leq n_r$, $\mathrm{tso}(C', D') = C' \vartriangle D'$.

— Lignes 1 et 2 :
  — Si $C = D = \top$, ce cas ne s'applique pas car $\mathsf{size}(C) + \mathsf{size}(D) = 2$.
  — Si $C = \top$ (et $C \neq D$), $\mathsf{br} = \{\top\} \setminus (\mathsf{br}_D \cup \{\top\}) = \emptyset$ et donc $C \vartriangle D = \top$, et $\mathrm{tso}(C, D) = \top$ également.
  — Si $C = D$ (et $C \neq \top$), $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_C \cup \emptyset) = \emptyset$ et donc $C \vartriangle D = \top$, et $\mathrm{tso}(C, D) = \top$ également.

— Lignes 4 à 10 : dans ce cas, on a $C = \bigsqcap_{i=1}^{k} C_i$, avec $k \geq 2$ et tous les C_i ne sont pas des conjonctions. Alors ⓐ $\mathsf{br}_C = \bigcup_{i=1}^{k} \mathsf{br}_{C_i}$. De plus, comme $\mathsf{size}(C_i) \leq \mathsf{size}(C) - 2$ on a $\mathsf{size}(C_i) + \mathsf{size}(D) \leq n_r - 2$. Alors d'après l'HR, il y a :
ⓑ $\forall i \in \{1, \ldots, k\}$,
$\mathrm{tso}(C_i, D) = C_i \vartriangle D$

$$= \begin{cases} E_i \mid ①' \ E_i \in \mathsf{subd}_{C_i} \text{ et } ② \ \mathsf{br}_{E_i} = \mathsf{br}_i & \text{if } \mathsf{br}_i \neq \emptyset \\ \top & \text{if } \mathsf{br}_i = \emptyset \end{cases}$$

avec $\mathsf{br}_i = \mathsf{br}_{C_i} \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2 ... \exists r_n.\top \in \mathsf{br}_{C_i}, n \geq 0 \mid$
$\exists S' = \exists r_1 ... \exists r_n.\exists r_{n+1} ... \exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$
Regardons $E = \bigsqcap_{i=1}^{k} \mathrm{tso}(C_i, D)$ puisque ce résultat de l'algorithme en est issu (modulo lignes 7 et 9). Alors $\mathsf{br}_E = \mathsf{br}_{\bigsqcap_{i=1}^{k} \mathrm{tso}(C_i, D)}$. D'après ⓑ, il y a :
$\forall 1 \leq i \leq k$,

$$\mathsf{br}_{\mathrm{tso}(C_i, D)} = \begin{cases} \mathsf{br}_i & \text{if } \mathsf{br}_i \neq \emptyset \\ \{\top\} & \text{if } \mathsf{br}_i = \emptyset \end{cases}$$

avec
$\mathsf{br}_i = \mathsf{br}_{C_i} \setminus (\mathsf{br}_D \cup \{S = \exists r_1 ... \exists r_n.\top \in \mathsf{br}_{C_i} \mid \exists S' = \exists r_1 ... \exists r_{n+m}.P \in \mathsf{br}_D\})$
Alors :
$\mathsf{br}_E = \mathsf{br}_{\bigsqcap_{i=1}^{k} \mathrm{tso}(C_i, D)} = \bigcup_{i=1}^{k} \mathsf{br}_{\mathrm{tso}(C_i, D)}$

$$= \begin{cases} \bigcup_{i=1}^{k} \mathsf{br}_i & \text{if } \bigcup_{i=1}^{k} \mathsf{br}_i \neq \emptyset \\ \{\top\} & \text{if } \bigcup_{i=1}^{k} \mathsf{br}_i = \emptyset \end{cases}$$

Notons que $\bigcup_{i=1}^{k} \mathsf{br}_i = \mathsf{br}$. On a :
$\mathsf{br} = \bigcup_{i=1}^{k} \Big(\mathsf{br}_{C_i} \setminus (\mathsf{br}_D \cup \{S = \exists r_1 ... \exists r_n.\top \in \mathsf{br}_{C_i} | \exists S' = \exists r_1 ... \exists r_{n+m}.P \in \mathsf{br}_D\})\Big)$
$= (\bigcup_{i=1}^{k} \mathsf{br}_{C_i}) \setminus (\mathsf{br}_D \cup \bigcup_{i=1}^{k} \{S = \exists r_1 ... \exists r_n.\top \in \mathsf{br}_{C_i} | \exists S' = \exists r_1 ... \exists r_{n+m}.P \in \mathsf{br}_D\})$
$= (\bigcup_{i=1}^{k} \mathsf{br}_{C_i}) \setminus (\mathsf{br}_D \cup \{S = \exists r_1 ... \exists r_n.\top \in (\bigcup_{i=1}^{k} \mathsf{br}_{C_i}) | \exists S' = \exists r_1 ... \exists r_{n+m}.P \in \mathsf{br}_D\})$
Et puisque ⓐ, on a :
$\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1 ... \exists r_n.\top \in \mathsf{br}_C \mid \exists S' = \exists r_1 ... \exists r_{n+m}.P \in \mathsf{br}_D\})$
et

$$\mathsf{br}_E = \begin{cases} \mathsf{br} & \text{if } \mathsf{br} \neq \emptyset \\ \{\top\} & \text{if } \mathsf{br} = \emptyset \end{cases}$$

C'est la preuve de ② pour E.
Puisque $E = \bigsqcap_{i=1}^{k} \mathrm{tso}(C_i, D)$, $C = \bigsqcap_{i=1}^{k} C_i$ et, d'après ⓑ, $\mathrm{tso}(C_i, D) \in \mathsf{subd}_{C_i}$, on obtient $E \in \mathsf{subd}_C$. C'est la preuve de ①' pour E (quand $\mathsf{br} \neq \emptyset$).

— Lignes 11 et 12 : dans ce cas, on a $D = \bigsqcap_{j=1}^{m} D_j$, avec $m \geq 2$ et tous les D_j ne

sont pas des conjonctions. On a alors $\mathsf{size}(D) = (\sum_{j=1}^{m} \mathsf{size}(D_j) + (m-1))$. De plus, le résultat de $\mathrm{tso}(C, D)$ est alors :

$E = \mathrm{tso}(\mathrm{tso}((..\mathrm{tso}(\mathrm{tso}(C, D_1), D_2), ..), D_{m-1}), D_m)$.

On rappelle que nous avons montré que l'algorithme 3 (p.50) termine toujours avec $\mathsf{size}(\mathrm{tso}(C, D)) < \mathsf{size}(C) + \mathsf{size}(D)$. Appliqué à E cela signifie que :

$\mathsf{size}(\mathrm{tso}(C, D_1)) \leq \mathsf{size}(C) + \mathsf{size}(D_1) - 1$

$\mathsf{size}(\mathrm{tso}(\mathrm{tso}(C, D_1), D_2)) \leq \mathsf{size}(\mathrm{tso}(C, D_1)) + \mathsf{size}(D_2) - 1$

...

$\mathsf{size}(\mathrm{tso}(\mathrm{tso}((...), D_{m-2}), D_{m-1})) \leq \mathsf{size}(\mathrm{tso}((...), D_{m-2})) + \mathsf{size}(D_{m-1}) - 1$

Donc, $\forall k \in \{2, .., m-1\}$:

$\mathsf{size}(\mathrm{tso}(\mathrm{tso}(..., D_{k-1}), D_k)) \leq \mathsf{size}(C) + (\sum_{j=1}^{k} \mathsf{size}(D_j)) - k$

Puisque $\mathsf{size}(D) = (\sum_{j=1}^{m} \mathsf{size}(D_j) + (m-1))$, $m \geq 2$ et $s_{in} = \mathsf{size}(C) + \mathsf{size}(D) = n_r + 1$, on a $\forall k \in \{2, .., m-1\}$:

$$\begin{aligned}\mathsf{size}(\mathrm{tso}(\mathrm{tso}((...), D_{k-1}), D_k)) + size(D_{k+1}) \\ \leq \mathsf{size}(C) + (\textstyle\sum_{j=1}^{k+1} \mathsf{size}(D_j)) - k \\ < \mathsf{size}(C) + \mathsf{size}(D) = s_{in} = n_r + 1\end{aligned}$$

On peut alors appliquer l'HR pour obtenir :

$\mathrm{tso}(C, D_1) = C \triangle D_1$

$\mathrm{tso}(\mathrm{tso}(C, D_1), D_2) = \mathrm{tso}(C, D_1) \triangle D_2$

...

$\mathrm{tso}(\mathrm{tso}((...), D_{m-2}), D_{m-1}) = \mathrm{tso}((...), D_{m-2}) \triangle D_{m-1}$

$E = \mathrm{tso}(\mathrm{tso}((...), D_{m-1}), D_m) = \mathrm{tso}((...), D_{m-1}) \triangle D_m$

Pour plus de clarté, on renomme $\mathrm{tso}(\mathrm{tso}((...), D_{k-1}), D_k)$ as tso_k, pour tout $k \in \{2, .., m\}$. On a alors :

$\mathrm{tso}_1 = C \triangle D_1$

$\mathrm{tso}_2 = \mathrm{tso}_1 \triangle D_2$

...

$\mathrm{tso}_{m-1} = \mathrm{tso}_{m-2} \triangle D_{m-1}$

$E = \mathrm{tso}_m = \mathrm{tso}_{m-1} \triangle D_m$

Avec la caractérisation de $C \triangle D$ obtenue à l'étape deux, on a :

$$\mathrm{tso}_1 = \begin{cases} E_1 \mid \text{①'} \ E_1 \in \mathsf{subd}_C \text{ et } \text{②} \ \mathsf{br}_{E_1} = \mathsf{br}_1 & \text{if } \mathsf{br}_1 \neq \emptyset \\ \top & \text{if } \mathsf{br}_1 = \emptyset \end{cases}$$

avec $\mathsf{br}_1 = \mathsf{br}_C \setminus (\mathsf{br}_{D_1} \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_{D_1}, m \geq 0\})$

$$\mathrm{tso}_2 = \begin{cases} E_2 \mid \text{①'} \ E_2 \in \mathsf{subd}_{\mathrm{tso}_1} \text{ et } \text{②} \ \mathsf{br}_{E_2} = \mathsf{br}_2 & \\ & \text{if } \mathsf{br}_2 \neq \emptyset \\ \top & \text{if } \mathsf{br}_2 = \emptyset \end{cases}$$

avec $\mathsf{br}_2 = \mathsf{br}_{\mathrm{tso}_1} \setminus (\mathsf{br}_{D_2} \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_{\mathrm{tso}_1}, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_{D_2}, m \geq 0\})$

...

$$E = \mathrm{tso}_m$$
$$= \begin{cases} E_m \mid \text{①'} E_m \in \mathsf{subd}_{\mathrm{tso}_{m-1}} \text{ et ② } \mathsf{br}_{E_m} = \mathsf{br}_m & \text{if } \mathsf{br}_m \neq \emptyset \\ \top & \text{if } \mathsf{br}_m = \emptyset \end{cases}$$
avec $\mathsf{br}_m = \mathsf{br}_{\mathrm{tso}_{m-1}} \setminus (\mathsf{br}_{D_m} \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_{\mathrm{tso}_{m-1}}, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_{D_m}, m \geq 0\})$

Puisque $\mathsf{br}_D = \bigcup_{j=1}^{m} \mathsf{br}_{D_j}$, il s'ensuit que, if $\mathsf{br} \neq \emptyset$, alors ①' $E \in \mathsf{subd}_C$ et ② $\mathsf{br}_E = \mathsf{br}$, et if $\mathsf{br} = \emptyset$, donc $E = \top$, avec $\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$, i.e. $\mathrm{tso}(C, D) = E = C \vartriangle D$.

— Lignes 13 à 19 : dans ce cas, $C = \exists r.C'$ et $D = \exists r.D'$.

Clairement, $\mathsf{size}(C') + \mathsf{size}(D') = \mathsf{size}(C) + \mathsf{size}(D) - 2 \leq n_r$. Donc on peut appliquer l'HR :

$$\mathrm{tso}(C', D') = C' \vartriangle D' = \begin{cases} E' \mid \text{①'} E' \in \mathsf{subd}_{C'} \text{ et ② } \mathsf{br}_{E'} = \mathsf{br}' & \text{if } \mathsf{br}' \neq \emptyset \\ \top & \text{if } \mathsf{br}' = \emptyset \end{cases}$$
avec $\mathsf{br}' = \mathsf{br}_{C'} \setminus (\mathsf{br}_{D'} \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_{C'}, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_{D'}, m \geq 0\})$

De plus, il est clair que $\mathsf{br}_C = \{\exists r.S \mid S \in \mathsf{br}_{C'}\}$ et $\mathsf{br}_D = \{\exists r.S \mid S \in \mathsf{br}_{D'}\}$. On définit alors :

$\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

et $\mathsf{br}' = \emptyset$ ssi $\mathsf{br} = \emptyset$.

De plus, on a également $\mathsf{subd}_C = \{\exists r.S \mid S \in \mathsf{subd}_{C'}\}$ et $\mathsf{subd}_D = \{\exists r.S \mid S \in \mathsf{subd}_{D'}\}$. Alors, quand $\mathsf{br}' \neq \emptyset$, on en conclut que le concept E' qui suit ①' $E' \in \mathsf{subd}_{C'}$ et ② $\mathsf{br}_{E'} = \mathsf{br}'$ définit un unique concept $E = \exists r.E$ qui suit ①' $E \in \mathsf{subd}_C$ et ② $\mathsf{br}_E = \mathsf{br}$. Cela montre que $\mathrm{tso}(C, D) = C \vartriangle D$ pour les lignes 16 et 18.

— Lignes 20 et 21 :

Dans ce cas, $C \neq \top$, $C \neq D$, C et D ne sont pas des conjonctions et C et D ne sont pas des restrictions existentielles avec le même rôle initial. Alors C et D peuvent être soit des restrictions existentielles avec un rôle initial différent soit des noms de concept différents soit l'un et l'autre. Dans tous les cas, cela mène à $\mathsf{br}_C \cap \mathsf{br}_D = \emptyset$, $\mathsf{subd}_C \cap \mathsf{subd}_D = \emptyset$ et il n'y a pas de couple de branche qui commence avec le même rôle. Alors $\mathsf{br} = \mathsf{br}_C$ avec

$\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid \exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

Alors $C \vartriangle D = C$ et $\mathrm{tso}(C, D) = C$ également.

d. **Complexité polynomiale**

Dans cette preuve, on considère que k est le temps d'exécution constant des opérations élémentaires suivantes : la concaténation de deux concepts A et B avec un symbole $\sqcap$ pour obtenir $A \sqcap B$, la concaténation d'une restriction existentielle $\exists r.$ avec un concept A pour obtenir $\exists r.A$, l'assignation d'une variable, la comparaison de deux concepts ou

noms de rôle et l'appel récursif de la fonction. On considère également que vérifier que deux concepts de taille n_1 et n_2 sont syntaxiquement identique peut être effectué en temps linéaire $\mathcal{O}(Min(n_1, n_2))$. Enfin, vérifier qu'un concept est une conjonction ou une restriction existentielle peut être effectué en temps constant car on suppose que les concepts sont représentés/stockés sous forme d'arbre syntaxique.

Soit C et D deux concepts $\mathcal{EL}$. Pour des raisons de clarté, on note $\mathsf{size}(C) = n_C$ et $\mathsf{size}(D) = n_D$. On note également $n = n_C + n_D$. Quand C est une conjonction, on considère qu'il a p conjoncts (et pas n comme dans l'algorithme 3 (p.50) puisque n est déjà égal à $n_C + n_D$). Quand D est une conjonction, on considère qu'il a m conjoncts (comme dans l'algorithme 3 (p.50)). Cherchons la borne supérieur pour $T(n)$ qui est le temps d'exécution de l'algorithme tso pour une entrée de taille n, c'est-à-dire l'appel $\text{tso}(C, D)$.

Rappelons l'algorithme 3 (p.50) qui calcule $\text{tso}(C, D)$.

Require: C et D deux concepts $\mathcal{EL}$.
Ensure: $C \vartriangle D$ (voir définition 23 (p.49))
1: **if** $C = D$ ou $C = \top$ **then**
2: $Result := \top$ {Cas 1 et cas 2}
3: **else**
4: **if** $C = C_1 \sqcap \ldots \sqcap C_n$ avec $n \geq 2$ **then**
5: $Result1 := \text{tso}(C_1, D) \sqcap \ldots \sqcap \text{tso}(C_n, D)$
6: **if** Il y a au moins un conjonct $\neq \top$ dans $Result1$ **then**
7: $Result := Result1$ sans aucun conjonct $\top$. {Cas 3}
8: **else**
9: $Result := \top$ {Cas 4}
10: **end if**
11: **else if** $D = D_1 \sqcap \ldots \sqcap D_m$ avec $m \geq 2$ **then**
12: $Result := \text{tso}(\ldots(\text{tso}(\text{tso}(C, D_1), D_2), \ldots), D_m)$ {Cas 5}
13: **else if** $C = \exists r.C'$ et $D = \exists r.D'$ **then**
14: $Result1 := \text{tso}(C', D')$
15: **if** $Result1 = \top$ **then**
16: $Result := \top$ {Cas 6}
17: **else**
18: $Result := \exists r.Result1$ {Cas 7}
19: **end if**
20: **else**
21: $Result := C$ {Cas 8}
22: **end if**
23: **end if**
24: Retourner $Result$

Puisque l'algorithme 3 (p.50) est construit avec des tests imbriqués menant vers 8 cas possible (tel que montré dans l'algorithme au dessus), on a $T(n)$ inférieur ou égal au temps maximum atteint par les 8 cas suivants :

$Min(n_C, n_D) * k$	L1 : test $C = D$	cas 1
$+k$,	L2 : assignation de variable	
$(Min(n_C, n_D) + 1) * k$	L1 : test $C = D$ et $C = \top$	cas 2
$+k$,	L2 : assignation de variable	
$(Min(n_C, n_D) + 1) * k$	L1 : $C = D$ et $C = \top$	cas 3
$+k$	L4 : test $C = C_1 \sqcap ... \sqcap C_p$	
$+p * k$	L5 : p appels recursifs	
$+\sum_{i=1}^{p} T(\mathsf{size}(C_i) + n_D)$	L5 : temps d'execution des appels recursifs	
$+(p-1) * k$	L5 : construction de la réponse en concaténant avec $\sqcap$	
$+k$	L5 : assignation de variable	
$+p * k$	L6 : p tests que les conjoncts $\neq \top$	
$+(p-1) * k$	L7 : retrait des au pire $p-1$ conjoncts	
$+k$,	L7 : assignation de variable	
$(Min(n_C, n_D) + 1) * k$	L1 : $C = D$ et $C = \top$	cas 4
$+k$	L4 : test $C = C_1 \sqcap ... \sqcap C_p$	
$+p * k$	L5 : p appels recursifs	
$+\sum_{i=1}^{p} T(\mathsf{size}(C_i) + n_D)$	L5 : temps d'execution des appels recursifs	
$+(p-1) * k$	L5 : construction de la réponse en concaténant avec $\sqcap$	
$+k$	L5 : assignation de variable	
$+p * k$	L6 : p tests que les conjoncts $\neq \top$	
$+k$,	L9 : assignation de variable	
$(Min(n_C, n_D) + 1) * k$	L1 : $C = D$ et $C = \top$	cas 5
$+k$	L4 : test $C = C_1 \sqcap ... \sqcap C_p$	
$+k$	L11 : $D = D_1 \sqcap ... \sqcap D_m$	
$+m * k$	L12 : m appels recursifs	
$+T(n_C + \mathsf{size}(D_1))$	L12 : temps d'execution des appels recursifs	
$+T(\mathsf{size}(tso(C, D_1)) + \mathsf{size}(D_2))$		
$+...$		
$+T(\mathsf{size}(tso(...)) + \mathsf{size}(D_{m-1}))$		
$+T(\mathsf{size}(tso(...)) + \mathsf{size}(D_m))$		
$+k$,	L12 : assignation de variable	

$(Min(n_C, n_D) + 1) * k$	L1 : $C = D$ et $C = \top$	cas 6
$+k$	L4 : test $C = C_1 \sqcap ... \sqcap C_p$	
$+k$	L11 : $D = D_1 \sqcap ... \sqcap D_m$	
$+2 * k$	L13 : $C = \exists r.C'$ et $D = \exists r.D'$	
$+k$	L14 : appel recursif	
$+T(n_C - 1 + n_D - 1)$	L14 : temps d'execution de l'appel recursif	
$+k$	L14 : assignation de variable	
$+k$	L15 : test $Result = \top$	
$+k$,	L16 : assignation de variable	
$(Min(n_C, n_D) + 1) * k$	L1 : $C = D$ et $C = \top$	cas 7
$+k$	L4 : test $C = C_1 \sqcap ... \sqcap C_p$	
$+k$	L11 : $D = D_1 \sqcap ... \sqcap D_m$	
$+2 * k$	L13 : $C = \exists r.C'$ et $D = \exists r.D'$	
$+k$	L14 : appel recursif	
$+T(n_C - 1 + n_D - 1)$	L14 : temps d'execution de l'appel recursif	
$+k$	L14 : assignation de variable	
$+k$	L15 : test $Result = \top$	
$+k$	L18 : construction de la réponse en concaténant avec $\exists r$.	
$+k$	L18. assignation de variable	
$(Min(n_C, n_D) + 1) * k$	L1 : $C = D$ et $C = \top$	cas 8
$+k$	L4 : test $C = C_1 \sqcap ... \sqcap C_p$	
$+k$	L11 : $D = D_1 \sqcap ... \sqcap D_m$	
$+2 * k$	L13 : $C = \exists r.C'$ et $D = \exists r.D'$	
$+k$	L21. assignation de variable	

Après simplification, on a :

$$
\begin{array}{lll}
T(n) & \leq & \\
Max(& & \left.\begin{array}{l}(Min(n_C, n_D) + 1) * k, \\ \text{cas 1}\end{array}\right\} \\
 & & \left.\begin{array}{l}(Min(n_C, n_D) + 2) * k, \\ \text{cas 2}\end{array}\right\} \\
 & & \left.\begin{array}{l}(Min(n_C, n_D) + 2 + 4p) * k + \sum_{i=1}^{p} T(\mathsf{size}(C_i) + n_D), \\ \text{cas 3}\end{array}\right\} \\
 & & \left.\begin{array}{l}(Min(n_C, n_D) + 3 + 3p) * k + \sum_{i=1}^{p} T(\mathsf{size}(C_i) + n_D), \\ \text{cas 4}\end{array}\right\}
\end{array}
$$

$$
\begin{array}{l}
\left.\begin{array}{l}
(Min(n_C, n_D) + 4 + m) * k \\
\quad + T(n_C + \mathsf{size}(D_1)) \\
\quad + T(\mathsf{size}(\mathrm{tso}(C, D_1)) + \mathsf{size}(D_2)) \\
\quad + ... \\
\quad + T(\mathsf{size}(\mathrm{tso}(...)) + \mathsf{size}(D_{m-1})) \\
\quad + T(\mathsf{size}(\mathrm{tso}(...)) + \mathsf{size}(D_m))
\end{array}\right\} \\
\text{cas 5} \\
\left.\begin{array}{l}
(Min(n_C, n_D) + 9) * k + T(n-2)
\end{array}\right\} \\
\text{cas 6} \\
\left.\begin{array}{l}
(Min(n_C, n_D) + 10) * k + T(n-2)
\end{array}\right\} \\
\text{cas 7} \\
\left.\begin{array}{l}
(Min(n_C, n_D) + 6) * k
\end{array}\right\} \\
\text{cas 8}
\end{array}
$$

)

On peut voir qu'il y a des cas avec des appels récursifs et d'autres sans. Les cas sans appel récursif s'exécutent en temps linéaire $\mathcal{O}(n)$ puisque $n_C \leq n$ et $n_D \leq n$. Les cas avec des appels récursifs ont une partie linéaire et une partie avec des appels récursifs. Alors, le pire cas se produit quand le nombre d'appels récursifs est maximal. Parmi les cas avec des appels récursifs, on a :

— les cas 3 et 4 ont p appels récursifs (p est le nombre de conjoncts C),
— les cas 5 ont m appels récursifs (m est le nombre de conjoncts D)
— et les cas 6 et 7 ont un seul appel récursif.

Comme p et m sont tous les deux plus grands que 2, alors maximiser le nombre d'appels récursifs implique de passer dans les cas 3, 4 ou 5. En d'autres termes, pour une taille en entrée n, on doit maximiser p et m, ce qui implique de minimiser le nombre de restrictions existentielles. Cela signifie que le pire cas pour l'algorithme 3 (p.50) se produit quand C et D sont tous les deux des conjonctions de noms de concept, sans aucune restriction existentielle. De plus, dans le pire cas, les noms de concepts dans C et D doivent tous être différents les uns des autres et différent de $\top$ pour que le cas 1 ne se produise jamais. Il est alors facile de voir qu'on obtiendra $\mathrm{tso}(C, D) = C$.

Dans le pire cas, l'exécution se déroule ainsi :

— l'appel principal de $\mathrm{tso}(C, D)$ entre dans le cas 3, c'est-à-dire le passage dans les lignes 4, 5, 6 et 7.
— ligne 5 déclenche p appels récursifs $\mathrm{tso}(C_i, D)$, $1 \leq i \leq p$.
  — chaque appel récursif de $\mathrm{tso}(C_i, D)$ entre dans le cas 5, c'est-à-dire le passage dans les lignes 1, 4, 11 et 12,
  — ligne 12 déclenche m appels récursifs $\mathrm{tso}(C_i, D_j)$, $1 \leq j \leq m$. Pour tout m appels récursifs, le premier argument reste C_i puisque C n'est pas modifié pendant le processus entier.
    — chaque appel récursif $\mathrm{tso}(C_i, D_j)$ entre dans le cas 8, i.e. c'est-à-dire le passage dans les lignes 1, 4, 11, 13 et 21.

Comme C est une conjonction de p nom de concept (de taille 1), il y a : $n_C =$

$\mathsf{size}(C) = (\sum_{i=1}^{p} 1) + p - 1 = 2p - 1$. De la même manière : $n_D = \mathsf{size}(D) = (\sum_{j=1}^{m} 1) + m - 1 = 2m - 1$. Comme $n = n_C + n_D$, on a $n = 2p + 2m - 2$ et alors $p + m = n/2 + 1$. On peut alors évaluer le temps d'exécution de $\mathrm{tso}(C, D)$ dans le pire des cas décrit plus haut : $T(n) = (Min(n_C, n_D) + 2 + 4p) * k + \sum_{i=1}^{p} T(\mathsf{size}(C_i) + n_D)$
avec

— pour tout i dans $\{1, .., p\}$:

$T(\mathsf{size}(C_i) + n_D) = (Min(\mathsf{size}(C_i), \mathsf{size}(D)) + 4 + m) * k$
$+T(\mathsf{size}(C_i) + \mathsf{size}(D_1))$
$+T(\mathsf{size}(C_i) + \mathsf{size}(D_2))$
$+...$
$+T(\mathsf{size}(C_i) + \mathsf{size}(D_{m-1}))$
$+T(\mathsf{size}(C_i) + \mathsf{size}(D_m))$

— pour tout j dans $\{1, .., m\}$:

$T(\mathsf{size}(C_i) + \mathsf{size}(D_j)) = (Min(\mathsf{size}(C_i), \mathsf{size}(D_j)) + 6) * k$

En remplaçant $\mathsf{size}(C_i)$ et $\mathsf{size}(D_j)$ par 1 puisque ce sont tous des noms de concept, on a :

$T(n) = (Min(n_C, n_D) + 2 + 4p) * k + p * (5 + 8m) * k$
$= k * Min(2p - 1, 2m - 1) + 4k * p + 2k + 5k * p + 8k * mp$

Puisque k est constant, $T(n)$ est dans $\mathcal{O}(mp)$. Comme $m + p = n/2 + 1$, mp est maximal quand $m = p = n/4 + 1/2$. On en déduit que $T(n)$ est dans $\mathcal{O}(n^2)$.

□

C.2 Preuve de la proposition 3 (p.51)

Dans cette section se trouvent les preuves des propriétés du CSO (voir 3.1.3 (p.51)) et de son algorithme (algorithme 4 (p.51)). On rappelle tout d'abord les propriétés du CSO :

Proposition 3. *Soient une TBox $\mathcal{T}$ et une paire de concepts $(C, D) \in (\mathcal{T}_{\mathcal{EL}})^2$. On a :*

a. $C \ominus_{\mathcal{T}} D$ existe et est unique (modulo $\equiv_{\mathcal{T}}$).

b. $cso(\mathcal{T}, C, D)$ se termine.

c. $C \ominus_{\mathcal{T}} D = \mathcal{T}^*(C) \triangle \mathcal{T}^*(D) = cso(\mathcal{T}, C, D)$ (Justesse de l'algorithme 4 (p.51)).

d. Calculer le $cso(\mathcal{T}, C, D)$ est EXPTIME par rapport aux tailles de $\mathcal{T}$ (car la complexité de l'expansion complète est EXPTIME (Voir Nebel (1990))), C et D et PTIME par rapport aux tailles de $\mathcal{T}^*(C)$ et $\mathcal{T}^*(D)$.

Démonstration. a. **Existence et unicité**

Dans le cas où $\mathsf{char}_C^{\mathcal{T}} \subseteq \mathsf{dcom}_{C,D}^{\mathcal{T}}$, l'existence et l'unicité de $C \ominus_{\mathcal{T}} D$ sont triviales. Dans les autres cas, $\mathsf{char}_C^{\mathcal{T}} \not\subseteq \mathsf{dcom}_{C,D}^{\mathcal{T}}$ est équivalent à $\exists S \in \mathsf{char}_C^{\mathcal{T}} \mid D \not\sqsubseteq_{\mathcal{T}} S$.Il y a donc toujours un $E = S$ tel que $C \sqsubseteq_{\mathcal{T}} E$ et $\mathsf{dcom}_{E,D}^{\mathcal{T}} = \emptyset$. Donc $C \ominus_{\mathcal{T}} D$ existe. Puisque $C \ominus_{\mathcal{T}} D$ doit être minimal par rapport à $\sqsubseteq_{\mathcal{T}}$, il est facile de voir qu'il est unique car il suffit de prendre la conjonction de tous les éléments de $\{E \in \mathcal{T}_{\mathcal{EL}} \mid C \sqsubseteq_{\mathcal{T}} E \text{ et } \mathsf{dcom}_{E,D}^{\mathcal{T}} = \emptyset\}$

pour avoir le concept minimal. Puisque les concepts définis peuvent être étendus par rapport à $\mathcal{T}$, le résultat est unique modulo $\equiv_{\mathcal{T}}$.

b. **Terminaison**

La terminaison est justifiée car l'expansion complète et l'algorithme 3 (p.50) se terminent également.

c. **Justesse**

Montrons maintenant $C \ominus_{\mathcal{T}} D = \mathcal{T}^*(C) \vartriangle \mathcal{T}^*(D)$ avec :

$$C \ominus_{\mathcal{T}} D = \begin{cases} Min_{\sqsubseteq_{\mathcal{T}}}\{E \in \mathcal{T}_{\mathcal{EL}} \mid C \sqsubseteq_{\mathcal{T}} E \text{ et } \mathsf{dcom}^{\mathcal{T}}_{E,D} = \emptyset\} & \text{if } \mathsf{char}^{\mathcal{T}}_C \not\subseteq \mathsf{dcom}^{\mathcal{T}}_{C,D} \\ \top & \text{if } \mathsf{char}^{\mathcal{T}}_C \subseteq \mathsf{dcom}^{\mathcal{T}}_{C,D} \end{cases}$$

et

$$\mathcal{T}^*(C) \vartriangle \mathcal{T}^*(D) = \begin{cases} Min_{\sqsubseteq}\{E \in \mathcal{T}^{\mathsf{prim}}_{\mathcal{EL}} \mid \mathcal{T}^*(C) \sqsubseteq E \text{ et } \mathsf{br}_E = \mathsf{br}\} & \text{if } \mathsf{br} \neq \emptyset \\ \top & \text{if } \mathsf{br} = \emptyset \end{cases}$$

avec $\mathsf{br} = \mathsf{br}_{\mathcal{T}^*(C)} \setminus (\mathsf{br}_{\mathcal{T}^*(D)} \cup \{S = \exists r_1.\exists r_2...\exists r_n.\top \in \mathsf{br}_{\mathcal{T}^*(C)}, n \geq 0 \mid$
$\exists S' = \exists r_1...\exists r_n.\exists r_{n+1}...\exists r_{n+m}.P \in \mathsf{br}_{\mathcal{T}^*(D)}, m \geq 0\})$
et avec P n'importe quel nom de concept ou $\top$ (sauf si m=0).
Remarque : $E \in \mathcal{T}^{\mathsf{prim}}_{\mathcal{EL}}$ est une consequence de $\mathcal{T}^*(C) \sqsubseteq E$.

Montrons : $\circledast$ $\mathsf{char}^{\mathcal{T}}_C \subseteq \mathsf{dcom}^{\mathcal{T}}_{C,D} \Leftrightarrow \mathsf{br} = \emptyset$ $\circledast\circledast$.
$\Rightarrow$:
Soit $S \in \mathsf{br}_{\mathcal{T}^*(C)}$.

— Cas 1 : $S \in \mathsf{char}^{\mathcal{T}}_C$.
Alors $S \in \mathsf{br}^{\mathsf{prim}}_{\mathcal{T}}$. Puisque $\circledast \Leftrightarrow \forall S \in \mathsf{char}^{\mathcal{T}}_C, D \sqsubseteq_{\mathcal{T}} S$, on a $D \sqsubseteq_{\mathcal{T}} S$. Puisque $S \in \mathsf{br}^{\mathsf{prim}}_{\mathcal{T}}$, on a $\mathcal{T}^*(D) \sqsubseteq S$. D'après le corollaire 1 (p.175), cela équivaut à l'existence d'un $S' \in \mathsf{subd}_{\mathcal{T}^*(D)}$ tel que S' peut être obtenu depuis S en appliquant dans S n'importe où zéro ou plusieurs fois les règles syntaxiques suivantes (r1) et (r2) qui remplacent un nom de concept qui est le côté gauche d'un axiome par son côté droit :
(r1) $\exists r.G \sqcap \exists r.H \rightsquigarrow \exists r.(G \sqcap H)$
(r2) $\top \rightsquigarrow H$
avec r n'importe quel rôle, et G et H n'importe quels concepts. Pour des raisons de simplicité, nous notons ces transformations : $S \overset{(r1),(r2)}{\rightsquigarrow} S'$. Comme S est une branche, S' est également une branche. Donc il y a : $\exists S' \in \mathsf{br}_{\mathcal{T}^*(D)} \mid S \overset{(r1),(r2)}{\rightsquigarrow} S'$. Puisque S est une branche, (r1) ne peut pas s'appliquer. Donc $\exists S' \in \mathsf{br}_{\mathcal{T}^*(D)} \mid S \overset{(r2)}{\rightsquigarrow} S'$. Puisque (r1) ne peut être appliqué qu'une seule fois sur une branche, on a les deux cas suivants :
— soit $S = S'$
— soit $S = \exists r_1...\exists r_n.\top$ et $S' = \exists r_1...\exists r_n.H$, avec $n \geq 0$ et $H \in \mathsf{br}^{\mathsf{prim}}_{\mathcal{T}}$.
D'après la définition de br, $S \notin \mathsf{br}$ dans les deux cas.

— Cas 2 : $S \notin \mathsf{char}^{\mathcal{T}}_C$.
$\forall G \in \mathcal{T}_{\mathcal{EL}} \mid G \equiv_{\mathcal{T}} C$ et $S \in \mathsf{br}_G$, $\exists H \in \mathcal{T}_{\mathcal{EL}} \mid \mathsf{br}_H = \mathsf{br}_G \setminus \{S\}$ et $H \equiv_{\mathcal{T}} C$. As $S \in \mathsf{br}_{\mathcal{T}^*(C)}$, cela signifie que que le concept $\mathcal{T}^*(C)^{-S}$, défini avec $\mathsf{br}_{\mathcal{T}^*(C)^{-S}} =$

$\mathsf{br}_{\mathcal{T}^*(C)} \setminus \{S\}$ et $\mathcal{T}^*(C)^{-S} \equiv \mathcal{T}^*(C)$, existe, est unique et est tel que $\mathcal{T}^*(C)^{-S} \sqsubseteq S$. Avec le même raisonnement que le cas précédent, on obtient :

$\exists S' \in \mathsf{br}_{\mathcal{T}^*(C)^{-S}} \mid$

— soit $S = S'$

— soit $S = \exists r_1 ... \exists r_n.\top$ et $S' = \exists r_1 ... \exists r_n.H$, avec $n \geq 0$ et $H \in \mathsf{br}_{\mathcal{T}}^{\mathsf{prim}}$.

Puisque l'expansion complète d'un concept C par rapport à $\mathcal{T}$ est un concept unique $\mathcal{T}^*(C)$, on a $\mathsf{br}_{\mathcal{T}^*(C)} = \mathsf{char}^{\emptyset}_{\mathcal{T}^*(C)} \subseteq \mathsf{char}^{\mathcal{T}}_{C}$. Cela signifie que $S' \in \mathsf{char}^{\emptyset}_{\mathcal{T}^*(C)^{-S}}$. Donc le cas où $S = S'$ n'est pas possible, car on aurait $S \in \mathsf{char}^{\mathcal{T}}_{C}$. Donc $S = \exists r_1 ... \exists r_n.\top$ et $S' = \exists r_1 ... \exists r_n.H$, avec $n \geq 0$ et $H \in \mathsf{br}_{\mathcal{T}}^{\mathsf{prim}}$. Puisque $S' \in \mathsf{char}^{\emptyset}_{\mathcal{T}^*(C)^{-S}}$ on a $S' \in \mathsf{char}^{\mathcal{T}}_{C}$. Supposons $\circledast$ $\mathsf{char}^{\mathcal{T}}_{C} \subseteq \mathsf{dcom}^{\mathcal{T}}_{C,D}$. Alors $S' \in \mathsf{dcom}^{\mathcal{T}}_{C,D}$ et $D \sqsubseteq_{\mathcal{T}} S'$. Puisque $S' \in \mathsf{br}_{\mathcal{T}}^{\mathsf{prim}}$, on a $\mathcal{T}^*(D) \sqsubseteq S'$. Avec le même raisonnement que précédemment, d'après le corollaire 1 (p.175), on obtient : $\exists S'' \in \mathsf{br}_{\mathcal{T}^*(D)} \mid S' = S''$. Et et on en dérive $S \notin \mathsf{br}$.

Dans les deux cas, $S \in \mathsf{br}_{\mathcal{T}^*(C)}$ implique $S \notin \mathsf{br}$. Donc $\mathsf{br} = \emptyset$.

$\Leftarrow$:

On suppose maintenant que $\mathsf{br} = \emptyset$. Cela signifie que $\forall S \in \mathsf{br}_{\mathcal{T}^*(C)}$, soit ① $S \in \mathsf{br}_{\mathcal{T}^*(D)}$, soit ② $S = \exists r_1 ... \exists r_n.\top$, avec $n \geq 0$, et $\exists S' = \exists r_1 ... \exists r_n ... \exists r_{n+m}.P \in \mathsf{br}_{\mathcal{T}^*(D)}$, avec $m \geq 0$.

Soit $S \in \mathsf{char}^{\mathcal{T}}_{C}$. On doit montrer que $S \in \mathsf{dcom}^{\mathcal{T}}_{C,D}$. C'est-à-dire que $D \sqsubseteq_{\mathcal{T}} S$. Si ① est vrai, on a $\mathcal{T}^*(D) \sqsubseteq S$ et donc $D \sqsubseteq_{\mathcal{T}} S$. Si ② est vrai, on a $\mathcal{T}^*(D) \sqsubseteq S' \sqsubseteq S$ et donc $D \sqsubseteq_{\mathcal{T}} S$.

On a alors montré que $\mathsf{char}^{\mathcal{T}}_{C} \subseteq \mathsf{dcom}^{\mathcal{T}}_{C,D} \Leftrightarrow \mathsf{br} = \emptyset$.

Ce lemme est utile pour la seconde partie de la démonstration de la justesse :

Lemme 1. *Soient C et D deux concepts. L'ensemble* br *est défini tel que :*

$\mathsf{br} = \mathsf{br}_C \setminus (\mathsf{br}_D \cup \{S = \exists r_1.\exists r_2 ... \exists r_n.\top \in \mathsf{br}_C, n \geq 0 \mid$
$\exists S' = \exists r_1 ... \exists r_n.\exists r_{n+1} ... \exists r_{n+m}.P \in \mathsf{br}_D, m \geq 0\})$

est tel que :

$$\mathsf{br} = \{S \in \mathsf{br}_C \mid D \not\sqsubseteq S\}$$

Démonstration.

$S \in \mathsf{br}$

$\Leftrightarrow$ • $S \in \mathsf{br}_C$ et
• $S \notin \mathsf{br}_D$ et
• si $S = \exists r_1 ... \exists r_n.\top$, avec $n \geq 0$
alors $\forall S' \in \mathsf{br}_D, S' \neq \exists r_1 ... \exists r_{n+m}.P$, pour tout $m \geq 0$ et P un nom de concept ou $\top$.

$\Leftrightarrow$ • $S \in \mathsf{br}_C$ et
$\forall S' \in \mathsf{br}_D$, S' ne peut pas être obtenu en appliquant zéro ou une fois la règle (r2) à S.

$\overset{S \text{ is a branch}}{\Leftrightarrow}$ • $S \in \mathsf{br}_C$ et
$\forall S' \in \mathsf{subd}_D$, S' ne peut pas être obtenu en appliquant zéro ou plusieurs fois les règles (r1) et (r2) to S.

$\overset{\text{corollary 1 (p.175)}}{\Leftrightarrow}$ • $S \in \mathsf{br}_C$ et
$D \sqsubseteq_{\mathcal{T}} S$

$\Leftrightarrow$ $S \in \{T \in \mathsf{br}_C \mid D \not\sqsubseteq T\}$

□

Avant le lemme 1 (p.188), on a montré que $\mathsf{char}_C^{\mathcal{T}} \subseteq \mathsf{dcom}_{C,D}^{\mathcal{T}} \Leftrightarrow \mathsf{br} = \emptyset$. Maintenant, supposons le contraire, c'est-à-dire que $\mathsf{char}_C^{\mathcal{T}} \not\subseteq \mathsf{dcom}_{C,D}^{\mathcal{T}} \Leftrightarrow \mathsf{br} \neq \emptyset$, on doit montrer :

$$Min_{\sqsubseteq_{\mathcal{T}}}\{E \in \mathcal{T}_{\mathcal{EL}} \mid C \sqsubseteq_{\mathcal{T}} E \text{ et } \mathsf{dcom}_{E,D}^{\mathcal{T}} = \emptyset\} = Min_{\sqsubseteq}\{F \in \mathcal{T}_{\mathcal{EL}}^{\mathsf{prim}} \mid \mathcal{T}^*(C) \sqsubseteq F \text{ et } \mathsf{br}_F = \mathsf{br}\}$$

Soit $F \in \mathcal{T}_{\mathcal{EL}}^{\mathsf{prim}}$ tel que :
① $\mathcal{T}^*(C) \sqsubseteq F$ et
② $\mathsf{br}_F = \mathsf{br}$ et
③ F est minimal par rapport à $\sqsubseteq$ tel que ① et ②.
On doit montrer que :
ⓐ $C \sqsubseteq_{\mathcal{T}} F$ et
ⓑ $\mathsf{dcom}_{F,D}^{\mathcal{T}} = \emptyset$ et
ⓒ F est minimal par rapport à $\sqsubseteq_{\mathcal{T}}$ tel que ⓐ et ⓑ.
D'abord, montrons que $C \equiv_{\mathcal{T}} \mathcal{T}^*(C)$, ① implique $C \sqsubseteq_{\mathcal{T}} F$. C'est ⓐ.
Ensuite, en supposant que $S \in \mathsf{char}_F^{\mathcal{T}}$, on doit montrer que $D \not\sqsubseteq_{\mathcal{T}} S$. Comme $F \in \mathcal{T}_{\mathcal{EL}}^{\mathsf{prim}}$, on a $\mathsf{char}_F^{\mathcal{T}} \subseteq \mathsf{br}_F$. Donc $S \in \mathsf{br}_F = \mathsf{br}$. D'après le lemme 1 (p.188), $S \in \mathsf{br}_{\mathcal{T}^*(C)}$ et $\mathcal{T}^*(D) \not\sqsubseteq S$. Alors $D \not\sqsubseteq_{\mathcal{T}} S$. C'est ⓑ.
Enfin, ③ $\Leftrightarrow$ ⓒ s'obtient de $F \in \mathcal{T}_{\mathcal{EL}}^{\mathsf{prim}}$.

d. **Complexité**

Le résultat s'obtient trivialement de la complexité de l'expansion complète et de l'algorithme 3 (p.50). □

C.3 Preuve de la proposition 4 (p.57)

Dans cette section se trouvent les preuves des propriétés du fill-up (voir 3.2.2 (p.57)) et de son algorithme (algorithme 5 (p.56)). On rappelle tout d'abord les propriétés du

fill-up :

Proposition 4 (Propriétés du fill-up et des algorithmes 5 (p.56) et 6 (p.56))**.** *Soit $\mathcal{T}$ une TBox normalisée et acyclique $\mathcal{EL}$ TBox, et C un concept $\mathcal{EL}$. On a :*

a. $Fup_{\mathcal{T}}(C)$ existe toujours.

b. Les algorithmes 5 (p.56) et 6 (p.56) se terminent et donnent un résultat unique.

c. fill-up$(\mathcal{T}, C)$ respecte les propriétés 1. et 2. de la définition de $Fup_{\mathcal{T}}(C)$.

d. Le calcul de $Fup_{\mathcal{T}}(C)$ est dans EXPTIME par rapport aux tailles de $\mathcal{T}$ et C.

Démonstration. **Preuve de a. (existence)** Il existe toujours au moins un fill-up car le concept suivant existe toujours : $C \sqcap (\bigsqcap_{S \in \mathsf{br}_{\mathcal{T}} \mid C \sqsubseteq_{\mathcal{T}} S} S)$ et il est facile de voir que les points 1. et 2. de la définition 24 (p.55) sont vérifiés.

Ébauche de preuve de b. (terminaison) Il est possible de montrer la terminaison de l'algorithme 6 (p.56) par récurrence sur la taille du concept d'entrée grâce à l'acyclicité de la TBox d'entrée et car les appels récursifs sont toujours effectués sur des concepts plus profondément imbriqués (et donc strictement plus petits que l'entrée initiale).

La terminaison de l'algorithme 5 (p.56) est assurée par :

— l'absence d'appel récursif et de boucle,

— la terminaison des fonctions rename-input(), rename-existential() et clean-up() (qui consistent à explorer des concepts pour en renommer ou supprimer certaines parties),

— la terminaison de la classification de Baader et al. (2005),

— et la terminaison de l'algorithme 6 (p.56).

L'unicité du résultat vient du fait qu'il n'y a pas d'indéterminisme dans les algorithmes, modulo la conjonction.

Ébauche de preuve de c. (justesse) Montrons que fill-up$(\mathcal{T}, C)$ a les propriétés de tout $Fup_{\mathcal{T}}(C)$ comme données à la définition 24 (p.55) :

1. $C \equiv_{\mathcal{T}}$ fill-up$(\mathcal{T}, C)$
2. $\mathsf{br}_{\text{fill-up}(\mathcal{T},C)} = \{S \in \mathsf{br}_{\mathcal{T}} \mid C \sqsubseteq_{\mathcal{T}} S\}$

D'abord on commence par montrer que les propriétés 1. et 2. sont vraies pour l'algorithme 6 (p.56), c'est à dire que :

1. $C' \equiv_{\mathcal{T}'}$ build-fill-up$(\mathcal{T}', Cl_{\mathcal{T}'}, C')$
2. $\mathsf{br}_{\text{build-fill-up}(\mathcal{T}', Cl_{\mathcal{T}'}, C')} = \{S \in \mathsf{br}_{\mathcal{T}'} \mid C' \sqsubseteq_{\mathcal{T}'} S\}$

On montre ce résultat par récurrence sur n qui est la valeur maximale telle que $C' \sqsubseteq_{\mathcal{T}'} \exists r_1.\exists r_2 ... \exists r_n.D$, avec D un nom de concept de $\mathcal{T}'$ ou $\top$.

— Cas de base $n = 0$:

Dans ce cas, il n'existe aucun nom de concept de forme $X_{\exists ...}$ dans $\mathbf{S}_{\mathcal{T}'}(C')$. Ainsi :

— Propriété 1. build-fill-up$(\mathcal{T}', Cl_{\mathcal{T}'}, C') = E_1 = \bigsqcap_{Z \in \mathbf{S}_{\mathcal{T}'}(C') \mid Z \neq X_{\exists r.D}} Z \equiv_{\mathcal{T}'} C'$ car soit $Z = C'$ soit $C' \sqsubseteq_{\mathcal{T}'} Z$ par définition de $\mathbf{S}_{\mathcal{T}'}(C')$.

— Propriété 2. On a facilement :
$\mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},C')} = \mathsf{br}_{E_1} \subseteq \{S \in \mathsf{br}_{\mathcal{T}'} \mid C' \sqsubseteq_{\mathcal{T}'} S\}$
De plus, on n'a dans $\mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},C')}$ aucune branche S avec au moins un rôle, car alors on aurait C' subsumé par $X_{\exists\ldots}$, et on est dans le cas où on n'a pas cela (car $n = 0$). Comme la classification est complète pour les relations de subsomption entre noms de concepts, on a l'inclusion dans l'autre sens : $\mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},C')} \supseteq \{S \in \mathsf{br}_{\mathcal{T}'} \mid C' \sqsubseteq_{\mathcal{T}'} S\}$

— Cas général : on suppose 1. et 2. vraies jusqu'à un certain $n > 0$. On veut savoir si 1. et 2. sont vraies pour $n+1$. Donc on suppose que $n+1$ est la valeur maximale telle que $C' \sqsubseteq_{\mathcal{T}'} \exists r_1.\exists r_2...\exists r_{n+1}.D$, avec D un nom de concept de $\mathcal{T}'$ ou $\top$.
— Propriété 1. On rappelle que $\mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},C')} = E_1 \sqcap E_2$. Or $E_1 \equiv_{\mathcal{T}'} C'$ pour les mêmes raisons que dans le cas de base.
— Propriété 2.
On pose $S = \exists r_1.\exists r_2...\exists r_{n+1}.D$ et
$X_{n+2} = D$,
$X_{n+1} = X_{\exists r_{n+1}.X_{n+2}}$,
...,
$X_2 = X_{\exists r_2.X_3}$, et
$X_1 = X_{\exists r_1.X_2}$.

— $\subseteq$
Toutes les branches de E_1 subsument C' par définition de $\mathbf{S}_{\mathcal{T}'}(C')$.
Soient $S' \in \mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},C')}$ et $S' \in E_2$. Donc S' est de forme $\exists r.S''$ avec $S'' \in \mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},F)}$, avec F un nom de concept et, par définition de $\mathbf{S}_{\mathcal{T}'}(C')$, $C' \sqsubseteq_{\mathcal{T}'} X_{\exists r.F}$. Par construction de $\mathcal{T}'$ on a $X_{\exists r.F} \equiv_{\mathcal{T}'} \exists r.F$. Il n'est donc pas possible que F soit subsumé par une branche avec strictement plus de n rôles car alors C' serait subsumé par une branche avec strictement plus de $n+1$ rôles, ce qui n'est pas notre hypothèse. Donc on peut appliquer l'hypothèse de récurrence à F qui implique que $F \sqsubseteq_{\mathcal{T}'} S''$. Et donc $C' \sqsubseteq_{\mathcal{T}'} \exists r.S'' = S'$.
— $\supseteq$
Soit une branche S' telle que $C' \sqsubseteq_{\mathcal{T}'} S'$.
Montrons que $S' \in \mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},C')}$.

— Cas où S' est un nom de concept ou $\top$. Par définition de $\mathbf{S}_{\mathcal{T}'}(C')$ et parce que la classification est complète, on a facilement que $S' \in E_1$ et donc $S' \in \mathsf{br}_{\text{build-fill-up}(\mathcal{T}',Cl_{\mathcal{T}'},C')}$.
— Cas où $S' = \exists r.S''$. D'après la construction de C' et $\mathcal{T}'$ dans fill-up$(\mathcal{T}, C)$, on a dans $\mathcal{T}'$ les axiomes suivants : $X_{\exists r.Y} \sqsubseteq S'$ et $X_{\exists r.Y} \sqsupseteq S'$ et $Y \equiv_{\mathcal{T}'} S''$. Donc $C' \sqsubseteq_{\mathcal{T}'} X_{\exists r.Y}$. Donc par le calcul de E_2, on a : $\exists r.$build-fill-up$(\mathcal{T}', Cl_{\mathcal{T}'}, Y)$ est une sous-description de build-fill-up$(\mathcal{T}', Cl_{\mathcal{T}'}, C')$. Pour les mêmes arguments que précédem-

ment, Y ne peut pas être subsumé par une branche ayant strictement plus de n rôles. Donc on peut appliquer l'hypothèse de récurrence à Y : comme S'' est une branche (car S' est une branche) et $Y \sqsubseteq_{\mathcal{T}'} S''$, on a par récurrence $S'' \in \mathsf{br}_{\text{build-fill-up}(\mathcal{T}', Cl_{\mathcal{T}'}, Y)}$. Et donc $S' = \exists r.S'' \in \mathsf{br}_{\text{build-fill-up}(\mathcal{T}', Cl_{\mathcal{T}'}, C')}$.

La preuve des propriétés 1. et 2. pour l'algorithme 5 (p.56) provient du fait que les transformations syntaxiques effectuées par les fonctions rename-input() et rename-existential(), ainsi que la normalisation, ne modifient pas la sémantique des concepts de $\mathcal{T}$ et des axiomes de $\mathcal{T}$. Ces transformations permettent le calcul du fill-up sur des noms de concept uniquement. Comme des concepts équivalents ont des fill-up ayant le même ensemble de branches, les fill-up calculés pour des noms de concepts sont attribués aux concepts d'origine en dernière instruction. La fonction clean-up() permet de retirer des fill-up calculés les noms des concepts intermédiaires créés par rename-input(), rename-existential() et la normalisation.

Preuve de d. (complexité) Calculer $Fup_{\mathcal{T}}(C)$ est dans EXPTIME par rapport aux tailles de $\mathcal{T}$ et C. En effet la taille de $Fup_{\mathcal{T}}(C)$ peut être exponentielle en la taille de C et $\mathcal{T}$ étant donné que la taille de $\mathcal{T}^*(C)$ peut l'être et que $Fup_{\mathcal{T}}(C)$ contient toutes les branches de $\mathcal{T}^*(C)$.

□

C.4 Trace de l'algorithme 5 (p.56) du fill-up

Exemple d'application de l'algorithme 5 (p.56) avec une TBox acyclique définitionnelle :

Exemple 46. *Soit la TBox $\mathcal{T}$ et la demande Dem suivantes :*
$\mathcal{T} = \{A \equiv B \sqcap C, C \equiv D \sqcap \exists r.B, D \equiv E, F \equiv \exists r.D\}$
$Dem = B \sqcap \exists r.D.$
Calculons fill-up$(\mathcal{T}, Dem)$:

- *Appel de l'algorithme 5 (p.56) : fill-up$(\mathcal{T}, B \sqcap \exists r.D)$*
  - $\mathcal{T}^0 := \{A \equiv B \sqcap C, C \equiv D \sqcap \exists r.B, D \equiv E, F \equiv \exists X_{r.D}\} \cup \{X_{B \sqcap X_{\exists r.D}} \sqsubseteq B \sqcap \exists r.D, B \sqcap \exists r.D \sqsubseteq X_{B \sqcap X_{\exists r.D}}\} \cup \{X_{\exists r.D} \sqsubseteq \exists r.D, \exists r.D \sqsubseteq X_{\exists r.D}\}$
  - $C^0 := X_{B \sqcap X_{\exists r.D}}$
  - *Calcul de $\mathcal{T}^0_{Bnf}$ la normalisation de $\mathcal{T}^0$*
  - $\mathcal{T}'$:= *rename-existential$(\mathcal{T}^0_{Bnf})$: $\mathcal{T}'$ est obtenue depuis $\mathcal{T}^0_{Bnf}$ en remplaçant $\exists r.B$ par $X_{\exists r.B}$ (et en ajoutant les deux axiomes de subsomption associés).*
  - *Calcul de la classification $Cl_{\mathcal{T}'}$ de $\mathcal{T}'$.*
    On obtient : $\boldsymbol{S}_{\mathcal{T}'}(X_{B \sqcap X_{\exists r.D}}) = \{\top, B, F, X_{\exists r.D}, Y_1, X_{B \sqcap X_{\exists r.D}}\}$ et $\boldsymbol{S}_{\mathcal{T}'}(D) = \{\top, E, D\}$. Y_1 a été généré pendant la normalisation.
  - *Appel de l'algorithme 6 (p.56) : build-fill-up$(\mathcal{T}', Cl_{\mathcal{T}'}, X_{B \sqcap X_{\exists r.D}})$*
    - $E_1 := \top \sqcap B \sqcap F \sqcap Y_1 \sqcap X_{B \sqcap X_{\exists r.D}}$
    - *Appel récursif : build-fill-up$(\mathcal{T}', Cl_{\mathcal{T}'}, D)$*
      - $E_1 := \top \sqcap E \sqcap D$

- — E_2 *est vide.*
- — *Alors build-fill-up*$(\mathcal{T}', D) = \top \sqcap E \sqcap D$.
- $E_2 := \exists r.(\top \sqcap E \sqcap D)$
- — *Alors fill-up*$(\mathcal{T}', X_{B \sqcap X_{\exists r.D}}) = \top \sqcap B \sqcap F \sqcap Y_1 \sqcap X_{B \sqcap X_{\exists r.D}} \sqcap \exists r.(\top \sqcap E \sqcap D)$

— *clean-up() enlève* Y_1 *et* $X_{B \sqcap X_{\exists r.D}}$ *(qui ne sont ni dans* $\mathcal{T}$ *ni dans* C*)*

— *Alors fill-up*$(\mathcal{T}, B \sqcap \exists r.D) = \top \sqcap B \sqcap F \sqcap \exists r.(\top \sqcap E \sqcap D)$

Résumé

Dans un contexte de recherche de ressources sur une plateforme documentaire, nous proposons le CCO, un opérateur de matchmaking sémantique multicritère se basant sur une ontologie du domaine exprimée en logique de description et sur une structuration en composantes des descriptions des ressources et des requêtes. Chaque composante permet de décrire un aspect particulier des ressources de la plateforme. Lors d'une recherche d'un utilisateur, le CCO utilise ces composantes pour obtenir un score de proximité sémantique d'une réponse potentielle à la demande par rapport à une autre. Ce score sert à obtenir le classement des meilleures réponses. Le CCO nécessite un opérateur de différence entre concept de l'ontologie capable d'identifier les informations manquantes et superflues de chaque réponse par rapport à la demande de l'utilisateur. Nous proposons pour cela le CSO, un opérateur syntaxique et sémantique pour les TBox définitionnelles et acycliques. Enfin, une implémentation du CCO et d'une approximation du CSO pour TBox générale et acyclique sont présentées, avec les tests associés.

Mots-clés: matchmaking, logique de description, ontologie, opérateur de différence, composante, recommandation, CCO, CSO, TSO.

Abstract

In a context of resources retrieval on a documentary platform, we propose the CCO, a multicriteria semantic matchmaking operator based on the domain ontology expressed in description logic and a component structure of the resource and request descriptions. Each component describe a specific aspect of the platform resources. Upon receiving a user's query, the CCO use those components to compute a proximity score for an answer in comparison to another. This score is used to compute the ordering of the best answers. The CCO requires a difference opérator between concept of an ontology, able to identify missing and unnecessary information from each answer with respect to the user's demand. In that regard, we propose the CSO, a semantic and syntaxic operator for definitional acyclic TBox. Finally, we present an implementation of the CCO and of an approximation of CSO for general acyclic TBox, with the associated tests.

Keywords: matchmaking, description logic, ontology, difference operator, component, recommendation, CCO, CSO, TSO.