De l'optimisation à la décomposition de l'ontologique dans la logique de description

Anh Le Pham

UNIVERSITE DE NICE - SOPHIA ANTIPOLIS

ECOLE DOCTORALE STIC

SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice-Sophia Antipolis

Mention:

présentée et soutenue par

Thi Anh Le PHAM

De l'Optimisation à la Décomposition de l'Ontologie dans la Logique de Description

---

Thèse dirigée par Nhan LE THANH

soutenue le 21 janvier 2008

Jury :

M. Peter Sander	Professeur à l'ESSI Sophia-Antipolis	Président
Mme Parisa Ghodous	Professeur à l'Université C.B Lyon 1	Rapporteur
M. Ollivier Haemmerlé	Professeur à l'Université Toulouse 2	Rapporteur
Mme Rose Dieng-Kuntz	Directeur de recherche à l'INRIA	Examinatrice
M. Franck Morvan	Maître de Conférences à l'Université Toulouse 3	Examinateur
M. Nhan Le Thanh	Professeur à l'UNSA	Directeur

Mis en page avec la classe Phdlasl

Remerciements

Cette thèse a été financée par le gouvernement vietnamien. Je tiens tout d'abord à remercier le gouvernement vietnamien, le Ministère de l'Éducation vietnamien et tous les membres du projet 322.

Je tiens à exprimer ma gratitude à Monsieur Nhan LE-THANH, mon directeur de thèse, pour m'avoir fait profiter de son expérience de recherche. Ses conseils sur l'aspect scientifiques et sur la méthode de recherche qui m'ont été utiles et permis de mener à bien cette thèse.

Je voudrais exprimer toute ma reconnaissance profonde à Monsieur Peter Sander, professeur à l'ESSI Sophia Antipolis, pour tous les suggestions, les remarques et les corrections en anglais de mes contributions scientifiques, et aussi pour m'avoir fait l'honneur de présider le Jury de soutenance.

Madame Parisa Ghodous et Monsieur Ollivier Haemmerlé ont accepté d'être les rapporteurs de cette thèse, et je les en remercie, de même que pour leur participation au jury. Ils ont également contribué par leurs nombreuses remarques et suggestions à améliorer la qualité de ce mémoire, et je leur en suis très reconnaissant.

Mes remerciements vont également à Madame Rose Dieng-Kuntz et Monsieur Franck Morvan qui m'ont fait l'honneur de participer au jury de soutenance.

Je ne sais comment exprimer ma gratitude à Madame Christel Dartigues et Madame Monique Oger pour m'avoir assisté sur les corrections françaises de ma rédaction, relu plusieurs versions de ce mémoire, je les en remercie sincèrement et profondément.

Un grand merci à M. Phu-Hiep Duong, Mme Xuan-Vinh Bui et M. Nhat-Minh Pham pour plusieurs remarques du français de ma rédaction.

Mes sincères remerciements sont adressés également aux collègues, amis et personnels du laboratoire I3S que j'ai côtoyé pendant ce travail, tout particulièrement André L.F. de Almeida, C. Estevao R. Fernandes, Sophiane Boudaoud, Thi-Dieu-Thu Nguyen, pour leur amitié et leur gentillesse.

J'aimerais souligner la participation de M. Viet-Hoang Vu, M. Tuan-Minh Pham et M. Chan Le-Duc à la discussion entourant les travaux de cette thèse.

Je remercie par ailleurs Mlle Fatou Cisse étudiante à l'EPU Sophia Antipolis qui a participé activement à une partie expérimentale de cette thèse.

Je passe ensuite une dédicace spéciale à tous les amis que j'ai eu le plaisir de côtoyer durant ces quelques années à Nice.

Enfin, je voudrais témoigner tous mes remerciements à ma famille pour leur encouragement et leur soutien.

Résumé

Le raisonnement efficace dans une grande base de connaissance en logique de description est un défi actuel en raison des inférences "insurmontables", même pour des langages des logiques de description relativement inexpressifs. En effet, la présence des axiomes dans la terminologie (TBox) est une des raisons importantes causant une augmentation exponentielle de la taille de l'espace de recherche exploré par les algorithmes d'inférence. Le raisonnement dans la logique de description (LD), c'est essentiellement le test de la relation de la subsomption entre les concepts. Par conséquent, on cherche toujours les expédients pour optimiser ce raisonnement. Des techniques d'optimisation pour améliorer la performance du raisonneur d'une LD se divisent donc naturellement en trois niveaux. Le premier est le niveau conceptuel considérant des techniques pour optimiser les structures d'axiomes dans la TBox. Le deuxième, le niveau algorithmique examinant des techniques pour réduire les exigences de stockage de l'algorithme de tableaux et pour optimiser le test de la relation de subsomption (satisfaisabilité). Le troisième, l'optimisation de requête cherchant des stratégies d'exécution optimales d'une requête d'interrogation dans une base de connaissances.

Dans ce mémoire, nous avons étudié une approche de décomposition de l'ontologie s'appelant la "*décomposition overlay*" qui vise à deux objectifs principaux : l'optimisation du raisonnement et la méthodologie de conception des ontologies.

- D'une part l'optimisation pour laquelle nous cherchons à diviser une ontologie dans un ensemble de sous ontologies dont chacune contient une partie de l'ensemble d'axiomes de l'ontologie originelle permet d'obtenir ainsi une réduction relative du temps de raisonnement ;
- D'autre part la méthodologie de conception qui permet de remplacer une ontologie par un ensemble d'ontologies dans une organisation plus ou moins "optimale".

Pour le premier objectif, la décomposition overlay d'une ontologie a comme résultat un ensemble de sous-ontologies regroupées dans une ontologie distribuée, appelée *ontologie-décomposante* (*TBox - décomposante*) et représentée par la logique de description distribuée. Intuitivement, le fait de pouvoir raisonner parallèlement sur ces sous-ontologies ayant chacune un espace de recherche réduit peut conduire à une réduction du temps relatif du raisonnement. Une propriété importante de cette ontologie est d'être interprétée dans le même domaine que l'ontologie originelle. Ceci est une base qui nous suggère la proposition de deux algorithmes de raisonnement pour cette ontologie-décomposante.

Concernant l'objectif de méthodologie de conception, nous introduisons deux méthodes de décomposition de l'ontologie reposant sur la décomposition heuristique des graphes. Une méthode repose sur la décomposition selon les séparateurs minimaux des graphes triangulaires et la seconde sur la décomposition selon la mesure des coupes normalisées des régions d'un graphe.

Mots clés : logique de description, logique de description distribuée, ontologie, optimisation, décomposition de l'ontologie, décomposition de graphe.

Abstract

Reasoning with a large knowledge base of description logics (DLs) is a real challenge because of intractable inferences even with relatively inexpressive logic description languages. Indeed, the presence of axioms in the terminology (TBox) is one of the main reasons causing an exponential increase in the size of the search space explored by the inference algorithms. The reasoning in the description logics is essentially to test subsumption relations between concepts. Therefore, one always seeks to optimize this reasoning. The techniques for improving the performance of a DL reasoner therefore divide naturally into three levels. The first is the conceptual level considering techniques for optimizing the structure of axioms in the TBox. The second examines the algorithmic level techniques to reduce storage requirements of the tableaux algorithms and the number of subsumption tests. The third finds the strategies for optimizing the queries given to a knowledge base.

In this thesis, we study an approach to ontology decomposition called the "*overlap decomposition*", which aims at optimizing the reasoning and methodology for ontology design.

- Optimization of reasoning is to find a method that sub-divides the axiom set of an ontology into a set of sub-ontologies. Each sub-ontology contains a subset of axioms of the original ontology. This allows a relative reduction of the reasoning time ;
- Methodology of the ontology design allows an ontology to be replaced by a set of ontologies in a more or less "optimal" organisation.

For the first objective, the overlap decomposition of ontology results in the *decomposing ontology* (*decomposing TBox*) represented by the distributed description logic. Intuitively, implementing the reasoning algorithms simultaneously on these sub-ontologies - each having a reduced search space -could lead to a relative reduction of time. An important property of this ontology is that it is interpreted in the same domain as the original ontology. This is a basis which suggests two reasoning algorithms for the decomposing ontology.

Regarding the goal of design methodology, we introduce two methods of ontology decomposition based on the heuristic graph decomposition. One is based on the minimal separators of graphs and the other is based on the normalised cut of the graph region.

Keywords : description logics, distributed description logics, ontology, optimisation, ontology decomposition, graphe partitioning.

Table des matières

Liste des tableaux

Table des figures

Introduction

Contexte et Motivation

Ces dernières années, avec le développement important du World Wide Web (WWW), les sources d'information deviennent de plus en plus multiformes, très riches et très larges. De plus, la progression des applications de base de données et de base de connaissances ont essentiellement convergé sur les techniques d'intégration qui essayent de surmonter les limites de chaque domaine isolé. Cela amène à la co-existence des grandes bases de données et bases de connaissances qui contiennent plusieurs domaines différents. Certains auteurs dans cette période comme Borgida A., Serafini L., ... [BS03, ST04a, ST04b] ont traité l'intégration de plusieurs sources, et puis ont raisonné en déduisant la connaissance d'une source à partir d'autres. Cependant, la gestion, le maintien et le traitement de telles sources impliquent encore de grandes difficultés pour l'homme et l'ordinateur.

La naissance du Web sémantique vise à désigner des technologies pour rendre le contenu des ressources du World Wide Web accessibles et utilisables par les programmes et les agents logiciels. Ces technologies sont parfois présentées comme des outils de représentation des connaissances adaptés à l'environnement Web, permettant de transformer automatiquement les données en information, et les informations en connaissance. Le Web sémantique favorise l'évolution de la connaissance *humaine* vers un autre type de connaissance *compréhensible* par des ordinateurs qui peut être traitée automatiquement dans les systèmes de gestion des connaissances. La base d'infrastructure du Web sémantique est l'utilisation des ontologies pour décrire les connaissances d'un domaine d'application, comme Baader et al. en ont discuté dans [BHS02].

Les *logiques de description* (LDs) forment une famille de formalismes de représentation de la connaissance terminologique d'un domaine d'application d'une manière structurée et bien formelle. Elles décrivent les notions importantes de domaine par les *descriptions de concept*, i.e., les expressions sont construites à partir des concepts atomiques (prédicats unaires) et des rôles atomiques (prédicats binaires) utilisant les constructeurs de concept et de rôle fournis par une LD particulière.

Les systèmes de représentation de connaissances basés sur les LDs (les systèmes de LDs)

fournissent diverses capacités d'inférence qui déduisent des *connaissances implicites* à partir des *connaissances explicites*. Les services de l'inférence des LDs sont classés sur deux composantes *terminologique* (la TBox) et *assertionnelle* (l'ABox) [DLNS96]. En particulier, le raisonnement avec la TBox se compose de *vérification* des relations de *subsomption* entre deux concepts, et de *satisfaisabilité* d'un concept, tandis qu'avec l'ABox le raisonnement correspond à la *vérification d'instance*. Par exemple, deux concepts PERSONNE et PARENT peuvent avoir une relation de subsomption : "PARENT est subsummé par PERSONNE" si et seulement si toutes les instances de PARENT sont également des instances de PERSONNE.

Par conséquent, le raisonnement dans une base de connaissance (BC) est une des applications les plus importantes des systèmes de la LD. Il est résolu en utilisant un algorithme prouvable correct et complet basé sur les calculs de tableau [Smu68]. Le problème de subsomption peut être transformé en problème de satisfaisabilité : le concept C subsume le concept D si et seulement si la description de concept "D et non C" n'est pas satisfait. C'est à dire que l'approche de l'utilisation de test de satisfaisabilité est limitée aux LDs qui soutiennent la négation générale. Malheureusement, la complexité dans le pire cas du problème de satisfaisabilité pour ces LDs est au moins NP-difficile [BFT95], et pour des LDs plus expressives peut être EXPTIME-complet [Cal96, BS01]. La complexité de calcul augmente selon l'expressivité de la LD utilisée. Ceci mène à la conjecture que l'applicabilité réelle des LDs expressives est limitée. Néanmoins, plusieurs analyses empiriques des applications réelles ont montré que les genres de construction qui conduisent à la difficulté du pire cas se produisent rarement dans la pratique [Neb90, BHN$^+$93]. Et surtout le développement et l'application des techniques d'optimisation de l'algorithme de tableau résultent de la performance du raisonnement pour des LDs expressives [Hor97, HPS99b, THPS07].

D'autre part, dans [Fra03, Cal96, BS01], les auteurs ont indiqué que la complexité de satisfaisabilité par rapport à la terminologie acyclique est PSPACE-complet dans le langage $\mathcal{ALC}$, alors que par rapport à la TBox générale elle est EXPTIME-complet toujours dans le langage $\mathcal{ALC}$. Donc, la complexité du raisonnement dépend encore de la présence des *axiomes terminologiques généraux* (*inclusion de concept général - ICG*) dans la forme $C \sqsubseteq D$, où C et D sont les descriptions de concept. Cela est dû au fait que chaque ICG cause une expression disjonctive qui est ajoutée à l'étiquette de chaque noeud dans l'arbre produit par l'algorithme de tableaux et il mène à une augmentation exponentielle de la taille de l'espace de recherche pour être explorée par la règle - $\bigcup$.

Heureusement, certaines techniques d'optimisation comme le *déploiement paresseux* et l' *absorption* [Hor97] sont connues pour être très efficaces pour réduire la taille de l'espace de recherche. Néanmoins, elles ne traitent qu'avec des ICG qui ont des structures particulières comme indiqué dans [Hor97]. Dans ce mémoire, nous proposons une nouvelle technique pour déduire le nombre d'axiomes qui s'appelle *décomposition de l'ontologie*.

Problématique

Les conditions en temps d'exécution et d'espace mémoire sont les deux facteurs significatifs qui influencent directement l'exécution d'un algorithme de raisonnement. Pour les grandes TBox ou les TBox arbitraires, le raisonnement efficace est un grand défi en raison des inférences insurmontables, même pour des logiques relativement inexpressives. Le raisonnement dans la logique de description repose essentiellement sur le test de la relation de subsomption entre les concepts. Par conséquent, on cherche toujours des moyens pour optimiser ce raisonnement. L'optimisation du raisonnement dans les LDs est naturellement divisée en trois niveaux :

- Le *niveau conceptuel* : Les techniques d'optimisation seront considérées dans l'étape de conception de l'ontologie. Elles peuvent être exécutées comme une phase de prétraitement qui vise à réduire le nombre des vérifications de subsomption exigées pour calculer l'ordre partiel [BHN+93]. Des éléments de l'ontologie sont structurés de façon à augmenter la performance de l'algorithme. Certaines techniques ont été développées à ce niveau dans le système FaCT [Hor97] comme la *simplification*, la *normalisation et l'encodage*, et l'*absorption* de IGCs. En particulier, la simplification, la normalisation et l'encodage restructurent des expressions de concept pour fournir la découverte plus rapide d'inconsistances. L'absorption consiste à réduire le nombre d'ICG en les absorbant en des axiomes d'introduction de concept. De plus, FACT réalise la classification des concepts qui est indépendante de langages terminologiques de LDs présentant l'ontologie. Cette classification vise à calculer une présentation complète d'un rangement partiel en faisant la comparaison explicite entre les éléments de l'ensemble fondamental (l'ensemble de tous les concepts se produisant dans la terminologie). Cette classification permet de détecter plus rapidement des relations de subsomption entre les concepts [BHS02].
- Le *niveau algorithmique* : l'algorithme de subsomption réel est changé de sorte qu'il peut utiliser des informations fournies par les relations de subsomption qui ont été précédemment calculées. Les techniques d'optimisation dans ce niveau optimisent des étapes de l'algorithme, par exemple, elles peuvent sous-diviser des essais de subsomption en utilisant un essai moins coûteux. Des techniques sont développées à ce niveau comme le *déploiement paresseux*, le *retour sur la direction de dépendance*, la *recherche de l'embranchement sémantique*, la *propagation de contrainte booléenne*, l'*heuristique* et la *mise en cache* [Hor97].
- Le *niveau d'optimisation de requête* [BBLS94] : pour le problème du raisonnement sur la TBox $\mathcal{T}$, une requête posée est une demande :"vérifier la satisfaisabilité d'une description de concept C ou la subsomption de deux descriptions de concept C et D ($C \sqsubseteq D$) par rapport à la $\mathcal{T}$?". En résumé, une requête posée est vue comme "vérifier la satisfaisabilité de l'expression Q qui est dans la forme C ou $C \sqsubseteq D$". En conséquence, l'optimisation de

requête est nécessaire dans le cas où elle est une expression complexe. On peut traiter la requête en la restructurant dans des expressions moins complexes, ou en la décomposant en des sous-requêtes qui seront vérifiées au lieu de l'expression complexe originelle. En particulier, la requête peut être ré-écrite dans une expression conjonctive, i.e., $Q \equiv Q_1 \sqcap ... \sqcap Q_n$ (chaque Q_i ($i \in \{1, .., n\}$) est une expression simple). L'essai de satisfaisabilité de Q sera remplacé par les vérifications de satisfaisabilité des $\{Q_i\}$ et une combinaison de résultats. Néanmoins, les techniques d'optimisation dans ce niveau sont très difficiles à effectuer parce que les requêtes ne sont pas toujours ré-écrites dans une expression conjonctive.

Notre problématique de recherche dans le domaine de l'optimisation démarre avec les questions suivantes :

1. ***Comment faire une optimisation de conception de l'ontologie*** ?
2. ***Quels sont les critères pour cette optimisation conceptuelle*** ?
3. ***L'ontologie obtenue de cette optimisation (ontologie optimale) préserve-t-elle la sémantique de l'ontologie originelle*** ?
4. ***Comment faire le raisonnement sur l'ontologie optimale et est-ce que les résultats d'inférence sur l'ontologie originelle sont préservés dans l'ontologie optimale*** ?
5. ***Est-ce que le raisonnement sur l'ontologie optimale est plus efficace que sur l'ontologie originelle*** ?
6. ***Avec cette optimisation, comment faire une optimisation de requête*** ?

Approches et Contributions

Dans ce mémoire, nous proposons une approche s'appelant la "décomposition overlay" qui se concentre sur deux niveaux d'optimisation conceptuel et algorithmique. Notre but est de réduire le nombre d'axiomes en sous-divisant l'ensemble des axiomes. Particulièrement, l'idée principale est de diviser l'ensemble des axiomes d'une ontologie en plusieurs sous-ensembles disjoints des axiomes dans les sous-ontologies respectivement. Tous les concepts et les rôles de l'ontologie originelle sont conservés dans toutes les sous-ontologies. Si nous examinons seulement les axiomes dans notre décomposition, c'est parce que leur présence rend difficile le raisonnement. Cette décomposition résulte en une *TBox-décomposante* qui est représentée comme une TBox distribuée, où les sous-ontologies qui correspondent aux ontologies composantes et aux concepts et rôles qui apparaissent dans les paires des axiomes de deux sous-ontologies différentes font des règles de pont entre ces deux sous-ontologies.

Avec cette approche de décomposition, nous souhaitons répondre au plus grand nombre possible des questions proposées. En particulier, dans le cadre de cette thèse, nous procédons comme suit :

- D'abord, à partir de la définition de décomposition overlay et de ses propriétés, nous prouvons la préservation de la sémantique de l'ontologie originelle (pour répondre à la question 3).
- Pour raisonner avec une TBox-décomposante, nous avançons deux algorithmes de tableau parallèle et distribué. La préservation des services d'inférence de la TBox originelle est montrée dans ces algorithmes (pour répondre à la question 4).
- L'analyse primaire de la complexité des algorithmes du raisonnement sur la TBox- décomposante nous permet de proposer des critères pour une "bonne décomposition" (pour répondre à la question 2). Nous considérons deux exigences principales qui sont dans l'ordre de priorité dégressif : les TBox composantes sont plus distinguées que possible et le cardinal d'axiomes entre les TBox composantes est quasi à l'équilibre. Autrement dit, *minimiser la partie commune entre les TBox composantes (minimiser le nombre des règles de pont)* et *équilibrer le cardinal d'axiomes entre les TBox composantes*. La première exigence fait allusion au fait que l'*association* entre les axiomes dans chaque TBox composante est maximale, et minimale entre les axiomes dans les TBox composantes différentes. En conséquence, notre approche est considérée comme une technique d'optimisation au niveau algorithmique.
- La théorie des graphes avec plusieurs outils et les "belles propriétés" de *coupe* nous aident à répondre à la question 1. Dans ce mémoire, nous proposons deux techniques de décomposition basées sur le problème de partition du graphe, elles reposent sur la décomposition basée sur le séparateur minimal et la décomposition basée sur la coupe normalisée. Les algorithmes ont également montré l'adaptation aux critères d'une bonne décomposition donnés ci-dessus. Par conséquent, notre décomposition est vue comme une étape d'optimisation dans le processus de conception de l'ontologie.

Par la décomposition, le nombre d'axiomes dans les TBox composantes est réduit significativement par rapport à la TBox originelle. Donc les requêtes sont exécutées très efficacement, car elles peuvent être traitées indépendamment sur les TBox composantes. Le raisonnement sur la TBox-décomposante dans le pire cas est vu comme sur la TBox originelle. L'efficacité du raisonnement dépend d'une "bonne décomposition".

Nous choisissons le langage $\mathcal{ALC}$ pour la recherche dans ce mémoire. $\mathcal{ALC}$ est un sous-ensemble de presque toutes les logiques de description expressives. Il fournit donc une expressivité pour plusieurs applications réelles. De plus, l'algorithme de test de satisfaisabilité est relativement simple et facile d'implémentation avec un champ des techniques d'optimisation.

Organisation du mémoire

Ce mémoire de thèse se compose de neuf chapitres répartis en trois parties. La première partie représente les notions principales et les techniques d'optimisation du raisonnement sur les logiques de description. La deuxième partie apporte une contribution principale à nos travaux, une approche de décomposition de l'ontologie qui capture la préservation de la sémantique et la préservation des services d'inférence de l'ontologie originelle. La troisième partie décrit deux méthodes particulières de décomposition de l'ontologie basées sur la théorie des graphes qui satisfont les critères de notre approche de décomposition.

Partie 1 - Logiques de description et techniques d'optimisation

Le **chapitre 1** étudie des approches de représentation des connaissances et indique que les logiques de description sont une famille de langages qui capturent la capacité de représentation et de raisonnement des connaissances terminologiques. La syntaxe et la sémantique de la LD standard sont également présentées dans ce chapitre.

Le **chapitre 2** présente les services d'inférence et les approches du raisonnement sur les deux composantes TBox et ABox, et se concentre sur le raisonnement sur la TBox. Ce chapitre décrit aussi les principes élémentaires de l'algorithme de tableau pour exécuter le problème de vérification de satisfaisabilité de concept.

Le **chapitre 3** introduit un champ des techniques d'optimisation qui ont été conçues pour améliorer la performance empirique des algorithmes de tableau.

Partie 2 - Théories de décomposition de l'ontologie

Dans le **chapitre 4** nous énonçons le problème de décomposition générale. Ensuite, nous présentons l'application de décomposition dans la théorie logique et dans le schéma de base de données relationnelle qui est la base pour la décomposition d'une base de connaissances (ontologie) dans les LDs. Nous donnons deux visions de la décomposition de base de connaissances *intra-ontologie* et *inter-ontologie* dans laquelle nous nous intéressons à la décomposition inter-ontologie. La fin du chapitre présente des formalismes hybrides pour la représentation des connaissances dans un système d'intégration de plusieurs ontologies ainsi que des conclusions.

Le **chapitre 5** débute par des notions principales dans la logique de description distribuée. Ensuite, nous proposons une méthode de décomposition de l'ontologie au niveau terminologique s'appelant *décomposition overlay*. La décomposition overlay d'une TBox est représentée comme une TBox distribuée qui s'appelle la *TBox-décomposante*. Dans ce chapitre, nous présentons les propriétés de TBox-décomposante, la distinction avec la TBox distribuée générale, et surtout la préservation de la sémantique de la TBox originelle est démontrée.

Le **chapitre 6** présente une contribution importante de nos travaux, y compris les deux mé-

thodes du raisonnement sur la TBox-décomposante qui sont décrites via deux algorithmes de tableau distribué et parallèle. Ces algorithmes nous permettent de prouver la préservation de services d'inférence de TBox originelle.

Partie 3 - Méthodes de décomposition

Le **chapitre 7** introduit des notions principales et quelques algorithmes utilisés pour la partition de graphe dans la théorie des graphes et la théorie de graphe spectral qui nous aident à chercher des méthodes appropriées pour notre décomposition.

Le **chapitre 8** réalise une autre contribution principale de ce mémoire par deux techniques particulières de décomposition de l'ontologie basées sur le graphe, la décomposition basée sur le *séparateur minimal* et la décomposition basée sur la *coupe normalisée*. Ce chapitre commence par une vue d'ensemble de la conception de l'ontologie qui conduit à une étape d'optimisation dans cette conception. Les critères de la décomposition de l'ontologie déduits de l'analyse de complexité des algorithmes du raisonnement dans le chapitre précédent sont proposés. Nous indiquons que ces critères sont appropriés à l'application de partition du graphe pour notre problème de décomposition, donc notre décomposition est nommée *G-Décomposition*. Dans ce chapitre, une méthodologie de G-Décomposition et les algorithmes de décomposition seront présentés. Nous discutons aussi des évaluations primaires de la complexité et quelques évaluations par des expériences de ces deux techniques.

Pour conclure les travaux présentés tout le long du mémoire, nous résumons la problématique, nos propositions, des approches et les résultats principaux. Nous discutons également les limites et les points restant à approfondir dans le futur.

Première partie

Logique de description et techniques d'optimisation

Dans cette partie, nous rappelons des notions de base du domaine de représentation de connaissances et présentons la Logique de Description (LD). Nous choisissons le formalisme de la logique de description à cause de sa capacité double de représentation et de raisonnement sur les connaissances. Cependant, la puissance de l'expressivité d'un langage de LD limite la performance des algorithmes du raisonnement corrects et complets. Concrètement, un langage plus expressif conduit à une complexité algorithmique élevée. Pour résoudre partiellement ce déséquilibre, plusieurs techniques d'optimisation des systèmes qui emploient les algorithmes de tableau sont étudiées et développées. Dans le dernier chapitre de cette partie, nous présentons quelques techniques d'optimisation qui peuvent être appliquées dans le champ large des logiques de description.

Chapitre 1

Représentation de connaissances et logiques de description

La recherche dans le domaine de la représentation de connaissances (RC) se concentre toujours sur les méthodes qui fournissent les bonnes descriptions du monde où elles peuvent être efficacement utilisées pour construire des applications intelligentes. Dans ce contexte, "*intelligent*" se rapporte à la capacité d'un système qui peut trouver des conséquences *implicites* de ses connaissances représentées *explicitement*. Les tels systèmes sont donc caractérisés comme des systèmes basés sur la connaissance.

D'autre part, les informations stockées sur le Web sont maintenant très importantes et très complexes avec des structures sémantiques diversifiées. Elles exigent des langages de modélisation, des méthodologies et des services de raisonnement plus *intelligents* pour soutenir les problèmes de conception, de gestion, de récupération, et d'intégration. Dans plusieurs applications réelles, un système moderne de RC est utile s'il a la possibilité non seulement de traiter de grands ensembles de données mais encore de fournir des langages de requête expressives.

Pour ces objectifs, la Logique de Description (LD) montre un formalisme approprié. Les logiques de description sont un domaine de recherche qui s'impose dans la RC avec plusieurs applications dans les bases de connaissances. L'effort principal de la recherche dans les LD est le développement de théories et de systèmes pour représenter les connaissances structurées et pour raisonner avec elles.

Ce chapitre commence par une vue générale de la représentation de connaissances avec deux formalismes basés sur la logique et non-logique. Nous introduisons brièvement la représentation de connaissances par une ontologie, le rôle des langages de l'ontologie dans le Web sémantique et discutons ensuite pourquoi les logiques de description sont bien adaptées en tant que langages de l'ontologie. Enfin, nous présentons successivement la syntaxe et la sémantique de la logique de description.

1.1 Représentation de connaissances

L'augmentation d'informations disponibles et de l'usage du Web ont créé de nouveaux besoins utilisateurs. L'invention du World Wide Web permet aux utilisateurs d'accéder facilement à l'information, tandis que les problèmes du traitement et de l'interprétation d'information récupérée restent encore un thème de recherche important qui s'appelle l'Intégration d'Information Intelligente.

Une connaissance est considérée comme une collection d'informations appropriées et relatives à un sujet particulier, et est employée pour réaliser un "comportement intelligent". Dans le domaine de l'intelligence artificielle (IA), le but primaire de la représentation de connaissances est de stocker la connaissance de sorte que les programmes puissent traiter et réaliser la vraisemblance de l'intelligence humaine. Autrement dit, les connaissances doivent être représentées d'une manière qui soit facile à inférer. Donc, un système de représentation de base de connaissances nécessite un langage *bien spécifié*, autrement dit un *formalisme*, pour coder les connaissances. Ce système est capable de fournir des mécanismes permettant d'inférer les *connaissances implicites* à partir des *connaissances explicites* stockées dans la base.

Autour des années 70, les approches de représentation de la connaissance peuvent être classées en deux catégories : les formalismes basés sur la logique et les présentations non-logiques.

- Les *représentations non-logiques* sont développées en construisant des notions plus cognitives permettant de manipuler des connaissances représentées dans des structures de données *ad hoc* comme les *réseaux sémantiques*, les *frames* etc. Le raisonnement sur ces structures est réalisé par des procédures *ad hoc*. Comme leur signification à l'origine est de converger vers les êtres humains, selon un point de vue pratique, on considère les systèmes basés sur les réseaux plus efficaces et plus attirants que les systèmes logiques. Malheureusement, ces représentations manquent d'une *caractérisation sémantique* précise, ceci ne nous permet pas de déterminer la correction et la complétude des conclusions du système.
- Les formalismes de représentation basés sur la logique [Baa96] ont dépassé l'intuition dans les représentations non-logiques. Ils utilisent clairement les calculs du prédicats du *premier-ordre* pour capturer des faits au sujet du monde. Dans cette approche, le langage de représentation est habituellement une variante du calcul du prédicat du premier-ordre, et le raisonnement permet de vérifier les conséquences logiques. Une base de connaissances (BC) contient un ensemble de données, souvent sous la forme de règles qui décrivent la connaissance d'une manière consistante logique. Des opérateurs logiques comme *et* (conjonction), *ou* (disjonction) et la *négation* peuvent être employés pour établir des connaissances plus complexes que des connaissances atomiques. Cette approche est vraiment efficace pour le traitement et l'exploitation automatique des connaissances.

Le *Web sémantique* est un nouveau domaine de recherche visant au développement des techniques et des outils de traitement automatique des informations sur le web. Plusieurs nouvelles technologies de représentation de connaissances sont développées dans ce domaine. Dans ce mémoire, nous nous intéressons aux formalismes de représentation de connaissances qui ont une sémantique déclarative, généralement logique, et des mécanismes d'inférence fondés par rapport à cette sémantique, particulièrement, une variante de la logique du premier-ordre qui s'appelle la *Logique de Description* (introduit dans la section 1.6), où les connaissances sont représentées par des objets logiques qui sont reliés par les propriétés, les axiomes et les règles.

1.2 Web Sémantique

Les sources de données du Web, comme nous le savons, constituent une collection énorme d'informations représentées sous des formes utiles pour l'être humain mais difficiles pour des traitements automatiques par un ordinateur. Le Web sémantique a été créé par Tim Berners-Lee [BL99] pour désigner les techniques qui peuvent rendre le contenu des ressources du Web accessible et utilisable par les processus automatisés des programmes, des agents logiciels. Le contenu des ressources du Web est décrit en utilisant les *méta-données* [1] formelles. Quand les méta-données sont structurées dans un ordre hiérarchique, elles sont habituellement appelées sous le nom *ontologie* ou schéma. En raison de l'évolution du Web sémantique, la notion de ressource s'étend de son sens original de "document publié sur le Web" à des sens plus généraux et plus abstraits. Par conséquent, les informations dans ces documents nécessitent d'être formalisées par les langages d'*ontologie* ou le langage SKOS[2]. Pour ces langages, les ressources sont décrits par les termes comme des concepts (classes) et des propriétés. Pour s'assurer que les différentes ressources ont une connaissance commune de ces termes, on a besoin d'ontologies dans lesquelles ces termes sont décrits. À ce titre, les langages et technologies du Web sémantique sont parfois présentés comme des outils de représentation de connaissances adaptés à l'environnement Web, permettant de transformer automatiquement les données en informations, et les informations en connaissances.

[1] Dans le cadre du Web sémantique, les méta-données sont des données signifiantes qui permettent de faciliter l'accès au contenu informationnel d'une ressource informatique

[2] SKOS est l'abréviation de Simple Knowledge Organisation System, voir le détail dans http ://www.w3.org/2004/02/skos/

1.2.1 Ontologies

Les ontologies sont utilisées et développées pour préciser la signification des ressources du Web. Plusieurs auteurs, y inclus humain et logiciel, peuvent communiquer à travers les ontologies. Une des définitions qui est largement utilisée dans l'Intelligence Artificielle, "*une ontologie est la spécification explicite et formelle d'une conceptualisation partagée d'un domaine d'application*" [Gru93]. Une *conceptualisation* est une abstraction d'un domaine d'application, elle identifie les concepts pertinents de ce domaine. Une *conceptualisation partagée* signifie que les concepts dans l'ontologie sont acceptés par une communauté d'utilisateurs. L'ontologie est la *spécification explicite* parce que tous les concepts et les contraintes utilisées de ces concepts sont explicitement définis. La *spécification formelle* implique que l'ontologie doit être représentée par un langage formel avec lequel la machine peut opérer. Pour cette définition, une ontologie fournit un vocabulaire partagé, incluant les concepts importants, les propriétés, les définitions, et les contraintes qui sont employés dans un domaine communicatif entre les personnes et les systèmes d'application hétérogènes et distribués.

L'objectif premier d'une ontologie est de modéliser un ensemble de connaissances dans un domaine donné. Donc, les ontologies jouent un rôle important dans l'intelligence artificielle, le Web sémantique, etc., elles sont employées comme une forme de représentation de la connaissance au sujet d'un monde ou d'une certaine partie du monde. Concrètement, dans le contexte du Web sémantique, les ontologies sont utilisées pour, d'une part, modéliser des ressources du Web à partir de représentations conceptuelles des domaines concernés et, d'autre part, l'objectif de permettre à des programmes de faire des inférences dessus. Les recherches à leur sujet sont donc indispensables. Une fois construite et acceptée par une communauté particulière, une ontologie doit en effet traduire un consensus explicite et un certain niveau de partage, deux aspects essentiels pour permettre l'exploitation des ressources du Web par différentes applications ou agents logiciels. D'autre part, la formalisation, autre facette des ontologies, est nécessaire pour qu'il soit possible de raisonner automatiquement dessus afin de décharger les utilisateurs d'une partie de leur tâche d'exploitation et de combinaison des ressources du Web. En résumé, les ontologies sont utilisées pour fournir le vocabulaire et la structure des méta-données associées aux ressources annotées, ou comme représentations pivot pour l'intégration de sources de données hétérogènes, ou pour décrire les services Web ou, en général, partout où il va être nécessaire d'appuyer des modules logiciels sur des représentations sémantiques.

1.2.2 Langages de l'ontologie du Web

En science de l'information et en intelligence artificielle, les langages de l'ontologie sont des langages formels utilisés pour construire les ontologies. Ils permettent l'encodage des connais-

sances sur des domaines spécifiques et comprennent souvent les règles de raisonnement qui supportent le traitement de ces connaissances. Les langages de l'ontologie sont généralement des langages déclaratifs, et presque toujours des généralisations de langages de *frame*, généralement fondés sur la *logique du premier ordre* [Smu68] ou la *logique de description*.

Le rôle important des ontologies dans la perspective du Web Sémantique a mené à l'extension de "langages majoratifs du Web" (Web markup languages) pour satisfaire la description du contenu et le développement du Web basé sur les ontologies, e.g., XML Schema[3], RDF[4], RDF Schema (RDFS) [BKD+02], OWL etc. RDFS a été recommandé par le W3C (World Wide Web Consortium). Il peut être considéré comme un langage primitif de l'ontologie dans le Web sémantique supportant la définition de classes (concepts), de ressources et de propriétés (rôles), et déterminant des relations de sous-classes et de sous-propriétés (subsomption).

Les descriptions dans une ontologie devraient être suivies d'un raisonnement automatisé pour détecter le rapport sémantique entre les termes syntaxiques différents. Ces existences ont mené au développement de DAML+OIL [Hor02], un langage expressif de l'ontologie Web, qui est conçu pour décrire la structure d'un domaine. Il adopte une approche orienté-objet, décrivant la structure en termes de classes et de propriétés. Une ontologie se compose d'un ensemble d'axiomes qui affirment, par exemple, des relations de subsomption entre les classes ou les propriétés.

Parmi les langages de l'ontologie du Web, OWL est un langage particulièrement important. Il est conçu comme une extension de RDF et de DAML+OIL. Le langage OWL peut être employé pour permettre la représentation explicite des vocabulaires des termes et des relations entre les entités dans ces vocabulaires. De cette façon, OWL dépasse XML, RDF et RDF-S en permettant un plus grand contenu compréhensible par une machine sur le Web. La conception du OWL a été influencée par une variété des paradigmes de représentation de connaissance, par des langages d'ontologie existants et par des langages du Web. Une des influences les plus importantes sur la conception de OWL est les spécifications formelles des langages de LDs. En particulier, il y a trois sous-langages de OWL : OWL Lite, OWL DL et OWL Full. OWL Lite et OWL DL sont des logiques de description fondamentalement très expressives avec une syntaxe de RDF. OWL Full fournit une intégration plus complète de RDF, mais l'inférence dans OWL Full constitue ainsi un problème indécidable. Les ontologies basées sur les LD peuvent être exploitées par des outils de raisonnement puissants de LD, afin de faciliter l'arrangement des ressources du Web. La possibilité de représentation de connaissances de la LD dans une manière structurée et bien formelle et d'inférence sera montrée dans les sections suivantes et le chapitre suivant.

[3]http ://www.w3.org/XML/Schema/

[4]http ://www.w3.org/RDF/

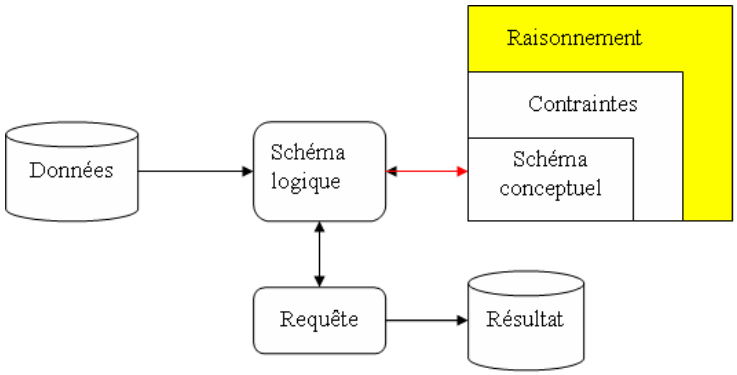


FIG. 1.1 : Modélisation conceptuelle et logique

1.3 Modélisation conceptuelle de l'ontologie

Par une vue d'ensemble, une ontologie est considérée comme un *schéma conceptuel* [Fra02] exprimé dans un modèle conceptuel de données (i.e., un langage de l'ontologie). Les *bons modèles de données* soulignent la représentation riche en sémantique et correcte. Ils devraient tenir compte d'une représentation abstraite qui capture la signification d'un domaine d'application comme la perception humaine. Ceci permet d'écourter la distance entre le domaine et sa représentation. La modélisation conceptuelle résout le problème : "comment décrire un domaine d'application avec son vocabulaire pertinent d'une manière déclarative et réutilisable, et comment contraindre l'utilisation des données, en reconnaissant ce qui peut être sorti de lui" [Fra02]. Par conséquent, les tâches de modélisation conceptuelle sont toujours présentées dans la tendance de recherche de représentation de connaissances et peuvent être considérées comme une des applications principales des langages de représentation de connaissances et des techniques du raisonnement [BB02]. La figure 1.1 illustre le rôle du schéma conceptuel (modèle de donnée conceptuel) dans la modélisation. Les ontologies sont utilisées pour la modélisation conceptuelle de données. Parce qu'une ontologie est une conceptualisation formelle du monde, c'est à dire qu'elle est adressée pour caractériser le monde réel d'une manière formelle. De plus, l'ontologie spécifie un ensemble de contraintes qui déclarent ce qui doit nécessairement retenir dans ce monde réel. Il y a plusieurs méthodes de modélisation conceptuelle comme Entité/Relation (ER) pour le modèle de données relationnel, UML et ODMG pour le modèle de donnée orienté-objet, et XML, RDF(S), DAML+OIL et OWL pour le modèle de donnée semi-structuré du web.

Les LD peuvent être considérées comme un formalisme unifié [BB02], parce qu'elles permettent la reconstruction logique et l'extension des outils représentatifs comme les modèles de donnée orienté-objet (UML), les modèles de donnée (ER, ORM), les langages de l'ontologie

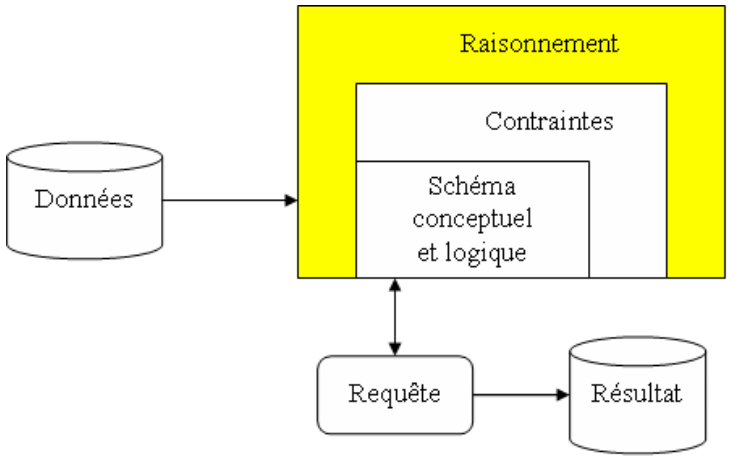


FIG. 1.2 : Conception de l'ontologie dans la logique de description

basés sur frame (DAML+OIL, OWL) [Fra02, CLN98]. En conséquence, une ontologie dans la LD est vue comme une association du schéma logique et schéma conceptuel (voir la figure 1.2). La recherche de la logique de description est très active en fournissant les langages expressifs de l'ontologie pour capturer les aspects différents de l'information [BB02, Fra02]. Enfin, l'utilisation des LD expressives permet de décrire les données complexes et semi-structurées dans le web sémantique. Ce qui donne à la logique de description un statut de candidat favorable pour un langage de modélisation conceptuelle des ontologies.

1.4 Réseaux sémantiques et frames

L'idée de développer des systèmes de représentation de connaissances basés sur une représentation structurée a été réellement exploitée dans les *réseaux sémantiques* et les *frames*.

1.4.1 Réseaux sémantiques

En 1968, Quillain a proposé les réseaux sémantiques [Qui68, Woo75] qui encodent les connaissances taxonomiques d'objets ainsi que leurs propriétés sous la forme d'un *graphe orienté étiqueté*. Les *sommets* dénotent des concepts et des objets, et les *arcs* représentent les diverses relations entre les concepts (voir la figure 1.3). Un tel graphe a deux types d'arcs : les arcs de propriété qui assignent des propriétés (par exemple, *avoir, travailler*) à des concepts (par exemple, *MERE*) et à des objets (par exemple, *UNIVERSITE*), et les arcs IS-A qui introduisent les relations hiérarchiques entre des concepts et les relations d'instance entre des objets et des

concepts.

Considérons un exemple simple, illustré dans la figure 1.3, qui représente la connaissance concernant des personnes, des parents, etc. La structure dans la figure est aussi désignée comme une *terminologie*, elle représente la généralité/spécificité des concepts impliqués. Par exemple, un arc entre MERE et PARENT dit que "les mères sont des parents" - une relation "IS-A". La relation IS-A définit une hiérarchie sur des concepts et fournit la base pour l'*héritage des propriétés* : quand un concept est plus spécifique qu'un autre, il hérite les propriétés du concept plus général. Par exemple, si un parent a un âge, alors une mère a aussi un âge. L'insuffisance

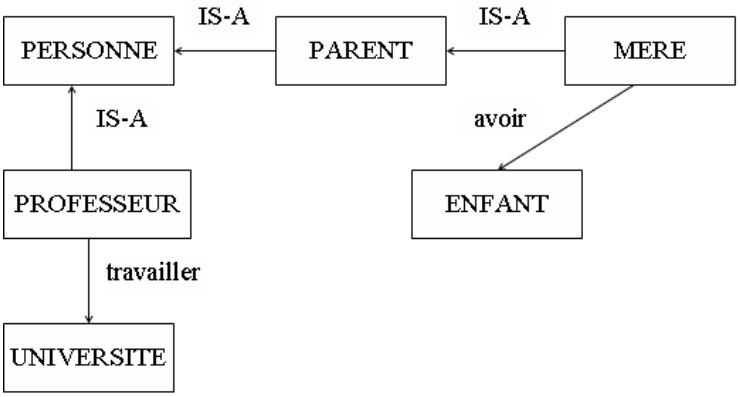


Fig. 1.3 : Un réseau sémantique

de réseaux sémantique est qu'ils n'ont pas de sémantique formelle, les arcs dans le réseau sémantique peuvent représenter différents genres de relations entre les sommets. La signification d'un réseau sémantique donné est décidée par l'intuition des utilisateurs et des programmeurs. Par conséquent, les différents systèmes de représentation de connaissances reposent sur le même réseau sémantique pourraient avoir des interprétations différentes. Nous pouvons illustrer ce problème par l'exemple dans la figure 1.3 : l'arc de propriété "travailler" de "PROFESSEUR" à "UNIVERSITE" peut indiquer que l'université est le seul établissement pour des professeurs, en outre, il peut vouloir dire qu'un professeur travaille dans au moins une université et aussi dans d'autres établissements.

Le raisonnement dans un réseau sémantique consiste à déterminer si un objet, représenté par un sommet A, est membre d'un ensemble, représenté par un sommet B. Pour cela, il faut suivre tous les arcs allant de A vers le haut (arcs IS-A et arcs d'instances) pour voir si on rencontre le sommet B. Afin de déterminer la valeur de certaines propriétés d'un objet représenté par le sommet A, on suit les arêtes allant de A vers le haut (comme précédemment) jusqu'à ce que l'on trouve un sommet ayant cette propriété (arc de propriété).

1.4.2 Frames

Les formalismes *frames* [Min75] peuvent également être considérés comme une extension des réseaux sémantiques, parce que l'exigence de la représentation est devenue plus complexe et plus structurée. L'idée principale est de collecter toutes les informations nécessaires pour traiter une situation dans une place. Les frames présentent des classes et des instances. Les relations entre frames sont présentées en utilisant des "slots". Chaque slot correspond à une propriété de la classe (ou de l'instance) présentée ou à une relation avec l'autre frame. Si une frame f est dans la relation r avec une frame g, alors on met la valeur g dans le slot r de f. Les systèmes de frame proposent une manière puissante de coder l'information pour soutenir le raisonnement. Ils ont largement été employés dans le traitement du langage naturel. Les frames sont bien connues parce que la modélisation basée sur les frames, comme la modélisation par l'objet est intuitive pour l'homme.

Néanmoins, les frames manquent également d'une sémantique formelle [Hay79]. Par exemple, ils ne donnent pas toujours bien des renseignements sur comment on peut interpréter une assertion où un slot est rempli avec une valeur particulière. Par exemple, la frame PERSONNE a un slot "avoirNom", c.-à-d., on veut dire que toutes les personnes doivent avoir un nom, mais, en effet quand on remplit une valeur "Frank" dans ce slot, c'est seulement une possibilité qu'une certaine personne ait le nom "Frank". En outre, les langages de représentation basés sur les frames ont des limites expressives. Par exemple, on ne peut pas représenter une nouvelle classe HOMME, où homme est une personne non femelle, des classes existantes PERSONNE et FEMME. Hayes [Hay79] a également montré que pour une formalisation appropriée, les frames peuvent être vus comme une variante syntaxique de la logique des prédicats du premier ordre.

Dans [Bra78], Brachman a proposé une nouvelle représentation, appelée *réseaux d'héritage structurels*. Cette représentation est pourvue d'une sémantique formelle permettant de capturer précisément la signification du formalisme qui est exprimée indépendemment des programmes attachés. Ce formalisme a été constaté la première fois dans le système KL-ONE [BS85] et est vu comme un prédécesseur d'une famille de langages plus expressifs capturant complètement les sémantiques, appelée *Logiques de Description* (LD). Les LD décrivent des connaissances en des termes de concepts et de relations qui sont utilisés pour déterminer automatiquement la classification de taxonomies. Une caractéristique d'une LD est que les concepts sont définis dans les termes de descriptions en employant les autres rôles et concepts. Une LD fournit un certain nombre de services de raisonnement qui permettent la construction des hiérarchies de classification et la vérification de la consistance de ces descriptions. Ces services de raisonnement peuvent alors être rendus disponibles aux applications qui souhaitent utiliser de la connaissance représentée dans l'ontologie.

Au fil du développement des langages de la représentation des connaissances avec la disponibilité d'une sémantique bien définie et d'outils de raisonnement puissants, les graphes conceptuels et les logiques de description ont été apparus. Ils répondent à ces deux besoins et se développent pour devenir actuellement les formalismes dominants dans le Web sémantique.

1.5 Graphes conceptuels

Les graphes conceptuels introduits par Sowa [Sow84] sont un formalisme expressif qui permet de représenter formellement les connaissances d'un domaine d'application. Un graphe conceptuel n'a pas de sens propre. C'est au travers des réseaux sémantiques que concepts et liens sont rattachés au contexte. Un graphe conceptuel est différent d'un réseau sémantique, parce qu'il contient une proposition. Concrètement, les graphes conceptuels décrivent les concepts (organisés en treillis), les relations (entre les concepts et entre les objets).

De façon générale, un graphe conceptuel est défini comme un graphe qui a deux sortes de noeuds :

- Les noeuds concepts qui représentent des entités, des attributs, des états, des événements,...
- Les noeuds relations conceptuelles qui symbolisent les liens existant entre deux concepts.

[CONCEPT]⟶(RELATION CONCEPTUELLE)⟶[CONCEPT]

Un arc entrant relie un concept à une relation conceptuelle, un arc sortant relie une relation conceptuelle à un concept.

- Chaque relation conceptuelle possède un arc ou plus, chacun d'entre eux doit être lié à un concept,
- Si une relation a n arcs, elle est dite n-adique.
- Un unique concept peut former un graphe conceptuel, mais chaque arc de chaque relation conceptuelle doit être lié à un concept quelconque.

Les graphes conceptuels sont donc des graphes finis, connectés et bipartis.

En plus, les graphes conceptuels sont munis d'un formalisme formel en les transformant en des formules de la logique du premier ordre. Ils peuvent exprimer toute la logique des prédicats du premier ordre. Certains services d'inférence sur ce formalisme ont été développés comme la validation d'un graphe et la subsomption entre deux graphes. Pour les graphes conceptuels généraux, ces problèmes d'inférence sont donc indécidables. Cependant, des fragments décidables de ce formalisme ont été précisés, par exemple le *graphe conceptuel simple*.

- **Concepts** : Les concepts représentent les objets de l'univers du discours. Il y a deux types de concept. Premièrement, le concept *individuel* représente un objet déterminé. Dans le

langage naturel, il correspond à un nom propre ou un nom commun précédé par un article défini. Par exemple, la France est représentée par le concept [PAYS :France]. Deuxièmement, le concept *générique* représente un objet indéfini. Il introduit le quantificateur existentiel $\exists$ et correspond à "Il existe un objet ...", par exemple, le concept [PERSONNE] représente *une* personne.

- **Relations conceptuelles** : Les relations conceptuelles permettent d'assembler les concepts pour construire une phrase, une idée, une proposition. Elles sont la deuxième sorte de noeuds qui constituent un graphe conceptuel. Par exemple, le graphe suivant représente la phrase "La France a pour capitale Paris" :

[PAYS : France]⟶(CAP)⟶[VILLE : Paris]

Les graphes conceptuels comme les autres types de représentation de connaissances sont utilisés dans diverses disciplines notamment dans le domaine biomédical [VJF98], plus précisément pour l'imagerie, l'instrumentation médicale, l'analyse et le traitement du langage naturel.

1.6 Logiques de description

Les Logiques de Description (LD) ont été d'abord développées pour fournir une signification formelle, déclarative aux réseaux sémantiques et aux frames, et pour montrer comment de telles représentations structurées peuvent être pourvues d'outils effectifs de raisonnement. Elles forment une famille de formalismes de représentation de connaissances qui peuvent être employés pour représenter et raisonner sur les connaissances d'un domaine d'application d'une manière structurée et formellement bien comprise. Une BC présentée par une LD contient deux composantes s'appelant *TBox* et *ABox*. La TBox introduit le vocabulaire d'un domaine d'application (terminologie) tandis que l'ABox contient les individus (d'assertions) apparaissant dans le domaine (la description du monde).

1.6.1 Syntaxe

Dans les langages de LD, la base de connaissances est représentée par les individus qui peuvent être groupés dans les classes, *noms de concept* (dénotés par les lettres A, B), et les relations binaires entre les individus, *noms de rôle* (dénotés par les lettres R, P).

Les langages différents fournissent des ensembles de *constructeurs* différents. Autrement dit, un langage particulier de concept est uniquement identifié par son ensemble de constructeurs qui permet d'établir des *descriptions de concept* plus complexes (dénotées par les lettres C, D), c.-à-d., les expressions de concepts sont construites par les noms de concept, les noms de rôle

et les constructeurs de concept et de rôle. Les noms de langages sont identifiés par les lettres associées à chaque ensemble des constructeurs [HNSS90], comme dans la table 1.1.

Un concept (rôle) obtenu en utilisant les constructeurs de $\mathcal{L}$ s'appelle un $\mathcal{L}$-concept ($\mathcal{L}$-rôle). Par exemple, la conjonction de deux concepts A et B, écrite comme $A \sqcap B$; la quantification universelle (restriction de valeur), écrite comme $\forall R.C$; et la quantification existentielle, écrite comme $\exists R.C$.

Les exemples suivants fournissent le sens intuitif pour les trois constructeurs ci-dessus, et les possibilités de la combinaison dans une LD :

Exemple 1.1 :

- Noms de concepts : PERSONNE, HOMME, FEMME, ETUDIANT, UNIVERSITE
- Noms de rôles : avoirEnfant, avoirParent

On peut donc définir les nouveaux concepts (définitions de concept) dans des termes d'une description intentionnelle des propriétés de ses membres. Par exemple, on définit un nouveau concept *PARENT* en utilisant les noms de concepts et les noms de rôles donnés :

$$\text{PARENT} \equiv \exists \text{ avoirEnfant.PERSONNE.}$$

De plus, on peut spécifier qu'une description, D, subsume ou est plus générale que l'autre, C, ce qui s'écrit $C \sqsubseteq D$, signifiant que quelque chose qui satisfait les conditions de C doit nécessairement, satisfaire les conditions de D, par exemple, PARENT $\sqsubseteq$ PERSONNE.

Une terminologie (TBox) de LD (base de connaissances terminologique) comprend un ensemble fini d'*axiomes terminologiques* (écrire en abréviation "axiome"). L'axiome est une introduction des noms de nouveaux concepts et de nouveaux rôles, ou une affirmation des relations de subsomption entre les descriptions de concept et de rôle.

Soient un langage $\mathcal{L}$, un nom de concept A, et deux $\mathcal{L}$-concepts C, D , un axiome dans $\mathcal{L}$ est de formes suivant :

1. $A \sqsubseteq C$, spécification de concept primitif
2. $A \equiv C$, définition de concept (introduction de concept primitif)
3. $C \sqsubseteq D$, inclusion de concept général (ICG)
4. $C \equiv D$, équation de concept

Essentiellement, les formes (1), (2), (4) sont les cas particuliers de la forme (3). En effet, si C est un nom de concept, alors (3) devient (1) et (4) devient (2). Par ailleurs, une équation de concept de la forme (4) peut être présentée par deux inclusions de concepts : $C \sqsubseteq D$ et $D \sqsubseteq C$.

Pour les buts d'utilisation différents, on classifie la terminologie en quelques types :

- *TBox primitive* : TBox se compose du type d'axiome (1).

Constructeurs du concept	$\mathcal{FL}^-$	$\mathcal{FL}$	$\mathcal{AL}$	$\mathcal{ALC}$	$\mathcal{ALCR}^+(\mathcal{S})$	$\mathcal{SHIQ}$
universel $\top$			×	×	×	×
vide $\bot$			×	×	×	×
conjonction	×	×	×	×	×	×
disjonction ($\mathcal{U}$)				×	×	×
négation ($\mathcal{C}$)			×	×	×	×
quantification existentielle inqualifiée	×	×	×	×	×	×
quantification existentielle ($\mathcal{E}$)		×		×	×	×
quantification universelle	×	×	×	×	×	×
rôle transitif (R^+)					×	×
rôle hiérarchie ($\mathcal{H}$)						×
restriction de nombre qualifiée ($\mathcal{Q}$)						×
rôle inverse ($\mathcal{I}$)						×

TAB. 1.1 : Langages de LD

- *TBox simple* : TBox comprend un ensemble de types d'axiomes (1) et (2). Dans une TBox simple, un nom de concept est appelé *concept défini* s'il apparaît du côté gauche d'une définition de concept ; un nom de concept qui apparaît du côté gauche d'une spécification du concept primitif s'appelle *concept primitif* ; et un nom de concept est appelé *concept atomique* s'il n'apparaît dans aucune partie gauche d'axiome. Une TBox simple est classifiée en deux types suivants :
  - *TBox cyclique* : une TBox est appelée *cyclique* si elle contient une inclusion de concept récursive, par exemple, {PERSONNE$\sqsubseteq$ $\exists$ avoirParent.MERE, MERE$\sqsubseteq$ $\exists$ avoirEnfant.PERSONNE }. Toutefois, de telles TBox ne sont pas examinées dans ce mémoire.
  - *TBox acyclique* : une TBox est appelée *acyclique* s'il n'existe pas un nom de concept qui est directement ou indirectement défini via lui-même (c.-à-d., pour chaque $A \equiv C$, A n'apparaît pas dans C). Habituellement, on impose la condition additionnelle que les définitions sont uniques (c.-à-d., un nom de concept apparaît au plus une fois comme le côté gauche d'une définition).
- *TBox générale* : TBox consiste en types d'axiomes (3), (4) (les types d'axiome (1), (2) sont des cas particuliers). Cette TBox permet aux concepts généraux d'apparaître dans le côté gauche d'un axiome. Quand on utilise les TBox générales, il n'y a pas de distinction entre le concept défini, le concept primitif et le concept atomique. Dans ce mémoire, nous considérons les TBox générales, donc nous ne distinguons pas ces trois genres de concept et ils sont souvent appelés par le *nom de concept*.

Pour illustrer, la table 1.2 présente la TBox $\mathcal{T}_{uni}$ décrivant une petite partie de l'université et des activités qui s'y produisent (nous extrayons quelques axiomes de l'ontologie SHOE [ST04a]).

1.6.2 Sémantique

Afin de définir les sémantiques de termes de concept, on considère des *interprétations* $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, qui se composent d'un ensemble non-vide $\Delta^{\mathcal{I}}$ (le *domaine d'interprétation*) et d'une fonction d'interprétation $.^{\mathcal{I}}$ qui associe chaque concept A à un ensemble $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, et chaque rôle R à un ensemble $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Une interprétation $\mathcal{I}$ est un *modèle* d'un concept C si $C^{\mathcal{I}}$ est non vide. Un concept C est *satisfaisable* s'il y a un modèle et *insatisfaisable* dans le cas inverse. Nous disons que C est *subsumé* par D si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, pour toutes $\mathcal{I}$; et C est *équivalent* à D ($C \equiv D$) si $C^{\mathcal{I}} = D^{\mathcal{I}}$, pour toutes $\mathcal{I}$.

Une interprétation $\mathcal{I}$ est un *modèle* d'une TBox si et seulement si elle satisfait tous les axiomes $A \equiv C$ et $P \equiv R$ dans la TBox, c.-a-d., si $A^{\mathcal{I}} = C^{\mathcal{I}}$ et $P^{\mathcal{I}} = R^{\mathcal{I}}$ pour toutes ces définitions.

ETUDIANT	$\sqsubseteq$	PERSONNER
ENSEIGNANT	$\sqsubseteq$	PERSONNER
THESARD	$\sqsubseteq$	STUDENT
THESE	$\sqsubseteq$	PUBLICATION
ARTICLE	$\sqsubseteq$	PUBLICATION
LIVRE	$\sqsubseteq$	PUBLICATION
THESE-MASTER	$\sqsubseteq$	THESE
THESE-DOCTORAL	$\sqsubseteq$	THESE
DOCTEUR	$\sqsubseteq$	THESARD $\sqcap\exists$ avoirPub.THESE-DOCTORAL
PROFESSEUR	$\sqsubseteq$	DOCTEUR$\sqcap\exists$ travailler.UNIVERSITE

Tab. 1.2 : Un exemple de la TBox $\mathcal{T}_{uni}$

THESARD(Mary), THESARD(Frank), ETUDIANT(Mary),
THESE-DOCTORAL(ThD1), THESE-DOCTORAL(ThD2),
ARTICLE(Ar1), ARTICLE(Ar2),
avoirPub(Marc, ThD1), avoirPub(Frank, ThD2), avoirPub(Frank, Ar2),
travailler(Frank, Nice-Université)

Tab. 1.3 : Un exemple de l'ABox $\mathcal{A}_{uni}$

Pour la composante d'*assertion* (ABox) d'une BC, on peut introduire les individus (en donnant leurs noms, et leurs propriétés). Si a, b sont des noms d'individus, C est un terme de concept et R est un terme de rôle, alors $C(a)$ et $R(a, b)$ sont des assertions. Un ensemble fini de telles assertions s'appelle une ABox $\mathcal{A}$. La table 1.3 montre un exemple d'une ABox qui correspond à la TBox $\mathcal{T}_{uni}$. L'interprétation $\mathcal{I}$ *satisfait* l'assertion de concept $C(a)$ si $a^{\mathcal{I}} \in C^{\mathcal{I}}$, et $\mathcal{I}$ *satisfait* l'assertion de rôle $R(a, b)$ si $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. Une interprétation $\mathcal{I}$ *satisfait* l'ABox $\mathcal{A}$ ($\mathcal{I}$ est un modèle de $\mathcal{A}$) si $\mathcal{I}$ satisfait toute assertion dans $\mathcal{A}$. La sémantique conduit à l'autre tâche, on peut affirmer qu'un individu est un membre d'un concept, comme une façon de lui donner une description partielle. Par exemple, ETUDIANT$\sqcap\neg$ HOMME (Elodie) affirme que Elodie est une étudiante qui n'est pas un homme. De plus, on peut affirmer l'inter-relation entre deux individus, e.g., avoirPub(Marc, ThD1) signifie que ThD1 est une publication de Marc.

Constructeurs	Syntaxe	Sémantique
universel (top) $\top$	$\top$	$\Delta^{\mathcal{I}}$
vide (bottom) $\bot$	$\bot$	$\emptyset$
conjonction	$C_1 \sqcap ... \sqcap C_n$	$C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$
disjonction ($\mathcal{U}$)	$C_1 \sqcup ... \sqcup C_n$	$C_1^{\mathcal{I}} \cup ... \cup C_n^{\mathcal{I}}$
négation ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
quantification existentielle inqualifiée	$\exists R.\top$	$\left\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in R^{\mathcal{I}}\right\}$
quantification existentielle ($\mathcal{E}$)	$\exists R.C$	$\left\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\right\}$
quantification universelle	$\forall R.C$	$\left\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\right\}$
rôle transitif (R^+)	R^+	$\bigcup_{1 \leq i \leq n} (R^{\mathcal{I}})^i$
rôle hiérarchie ($\mathcal{H}$)	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
restriction de nombre ($\mathcal{N}$) ($\geq$)	$\geq nR$	$\left\{x \in \Delta^{\mathcal{I}} \mid card(y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}) \geq n\right\}$
restriction de nombre ($\mathcal{N}$) ($\leq$)	$\leq nR$	$\left\{x \in \Delta^{\mathcal{I}} \mid card(y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}) \leq n\right\}$
restriction de nombre qualifiée ($\mathcal{Q}$) ($\geq$)	$\geq nR.C$	$\left\{x \in \Delta^{\mathcal{I}} \mid card\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\} \geq n\right\}$
restriction de nombre qualifiée ($\mathcal{Q}$) ($\leq$)	$\leq nR.C$	$\left\{x \in \Delta^{\mathcal{I}} \mid card\{y \in \Delta^{\mathcal{I}} \mid (x,y) \in R^{\mathcal{I}}, y \in C^{\mathcal{I}}\} \leq n\right\}$
rôle inverse ($\mathcal{I}$)	R^-	$\{(x,y) \mid (y,x) \in R^{\mathcal{I}}\}$

Tab. 1.4 : Syntaxe et Sémantique de termes de concept et de rôle

Par conséquent, étant donné un langage de LD $\mathcal{L}$, une *base de connaissances* $\sum$ dans $\mathcal{L}$ est une paire $\sum = (\mathcal{T}, \mathcal{A})$, où $\mathcal{T}$ est une TBox dans $\mathcal{L}$ et $\mathcal{A}$ est un ABox dans $\mathcal{L}$. Une interprétation $\mathcal{I}$ est un modèle de $\sum$ si elle est un modèle de tous les deux $\mathcal{T}$ et $\mathcal{A}$. Une BC $\sum$ implique logiquement α, où α est un axiome de TBox ou une assertion (contrainte) de l'ABox, écrit $\sum \models \alpha$, si α est vrai dans chaque modèle de $\sum$.

1.7 Conclusion

Dans ce chapitre, nous avons rappelé des notions de base du domaine de représentation de connaissances. Par l'introduction brève de l'histoire, la syntaxe et la sémantique des logiques de description, nous avons montré la capacité de représentation de connaissances d'une manière structurée et bien formelle des LD. Ceci contribue à l'affirmation que les LD sont des bons candidats pour la représentation de connaissances. Dans le prochain chapitre, nous allons aborder des diverses capacités du raisonnement de LD qui résultent des connaissances implicites à partir des connaissances explicites.

Chapitre 2

Raisonnement dans la logique de description

Les systèmes de LD fournissent aux utilisateurs plusieurs capacités d'inférence. Le raisonnement permet d'inférer des connaissances implicites à partir des connaissances explicites stockées dans la base de connaissances. L'inférence élémentaire sur les expressions de concepts dans la LD est la *subsomption* entre deux concepts qui détermine les relations de sous-concept / super-concept. L'inférence élémentaire sur les individus est de déterminer si un individu donné est une instance d'un certain concept. La recherche du raisonnement dans la LD se concentre sur deux questions. La première est de rechercher des algorithmes corrects et complets pour résoudre les problèmes de raisonnement. Ceci garantit la précision des résultats du raisonnement et l'intégralité de l'implémentation du raisonnement dans le système de représentation de connaissances. La seconde est le compromis entre la complexité des algorithmes de raisonnement et l'expressivité du langage utilisé. Ici la difficulté du raisonnement dépend de l'expressivité du langage : plus le langage est expressif, plus le raisonnement est complexe. Dans ce chapitre, nous examinons le premier problème.

Ce chapitre débute par l'introduction des services d'inférence élémentaires sur la TBox et l'ABox. Par la suite, nous décrirons les algorithmes pour résoudre les problèmes de raisonnement dans les LDs. Finalement, les problèmes de raisonnement dans la TBox et l'algorithme de tableaux sont présentés plus en détail.

2.1 Services d'inférence

Un système de LD stocke non seulement des terminologies et des assertions, mais offre également les services d'inférence. La charge principale du raisonnement dans une LD est de découvrir des connaissances implicites à partir des connaissances explicites par l'inférence. Les

services d'inférence sont également réalisés sur toutes la TBox et ainsi que sur l'ABox.

2.1.1 Inférences élémentaires sur la TBox

Étant donnée $\mathcal{T}$ une TBox, C et D deux concepts, alors les tâches typiques du raisonnement sur $\mathcal{T}$ se composent de :

- La vérification de *satisfaisabilité* d'un concept : Un concept C est satisfaisable (ou consistant) par rapport à une TBox $\mathcal{T}$ s'il existe un modèle $\mathcal{I}$ de la TBox $\mathcal{T}$ tel que $C^{\mathcal{I}} \neq \emptyset$ ($\mathcal{I}$ est un modèle de C), on écrit $\mathcal{I} \models C$
- La vérification de relation de *subsomption* entre deux concepts : D *subsume* C (le concept D est considéré plus général que C), écrit $C \sqsubseteq D$, par rapport à TBox $\mathcal{T}$ ssi $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ pour tous modèles $\mathcal{I}$ de la TBox $\mathcal{T}$. Dans ce cas, on écrit $C \sqsubseteq_{\mathcal{T}} D$ ou $\mathcal{T} \models C \sqsubseteq D$. Par exemple, PARENT$\sqsubseteq$ PERSONNE. La relation de subsomption présente le service plus complexe de classification : étant donné un concept C et une TBox $\mathcal{T}$, pour tous les concepts D de $\mathcal{T}$ déterminent si D subsume C ou D est subsumé par C. Intuitivement, cette détermination recherche des relations implicites dans la terminologie. En particulier, la *classification*, une tâche de base dans la construction de terminologie qui place une nouvelle expression de concept dans l'endroit approprié dans la hiérarchie taxonomique des concepts, peut être accomplie en vérifiant la relation de subsomption entre chaque concept défini dans la hiérarchie et la nouvelle expression de concept.
- La vérification d'*équivalence* entre deux concepts : Deux concepts C et D sont équivalent, écrit $C \equiv D$, par rapport à $\mathcal{T}$ ssi $C^{\mathcal{I}} \equiv D^{\mathcal{I}}$ pour tous modèles $\mathcal{I}$ du TBox $\mathcal{T}$. Dans ce cas, on écrit $C \equiv_{\mathcal{T}} D$ ou $\mathcal{T} \models C \equiv D$
- La vérification de *disjonction* entre deux concepts : Deux concepts C et D sont disjoints, écrit $C \neq D$, par rapport à une TBox $\mathcal{T}$ ssi $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ pour tous modèles $\mathcal{I}$ de TBox $\mathcal{T}$.

En fait, la vérification de la satisfaisabilité de concept est une inférence principale. Dans [BN03], Baader F. et Nutt W. ont montré que les autres inférences pour les concepts peuvent être réduites à la (in)satisfaisabilité et inversement.

Proposition 2.1. *(Réduction à insatisfaisabilité) : Pour deux concepts C et D, on a :*

- *C est subsumé par D* $\Longleftrightarrow$ $C \sqcap \neg D$ *est insatisfaisable.*
- *C et D sont équivalents* $\Longleftrightarrow$ *tous les deux* $C \sqcap \neg D$ *et* $(\neg C \sqcap D)$ *sont insatisfaisables.*
- *C et D sont disjoints* $\Longleftrightarrow$ $C \sqcap D$ *est insatisfaisable.*

Proposition 2.2. *(Réduction à subsomption) : Pour deux concepts C et D, on a :*

– *C est insatisfaisable* $\Longleftrightarrow$ *C est subsumé par* $\bot$.

– *C et D sont équivalents* $\Longleftrightarrow$ *C est subsumé par D et D est subsumé par C*.

– *C et D sont disjoints* $\Longleftrightarrow$ $C \sqcap D$ *est subsumé par* $\bot$.

Dans [LB87, Neb88], les auteurs ont montré que, même pour les très petits langages, il ne peut pas exister des algorithmes de subsomption qui sont à la fois complets et de complexité polynomiale. La complexité de $\mathcal{ALC}$ par rapport à une TBox vide est PSPACE [SSS91]. Par rapport à une TBox acyclique, la complexité de $\mathcal{ALC}$ et $\mathcal{SHIQ}$ est ExpTime-complet. Toutefois, l'optimisation des algorithmes de tableaux a montré une performance raisonnable dans plusieurs applications réalisées [Hor97]. Certaines techniques d'optimisation parmi elles seront étudiées en détail dans le Chapitre 3.

2.1.2 Inférences élémentaires sur l'ABox

Le raisonnement sur une ABox se focalise sur le test de la correction d'un modèle du domaine. Il faut effectuer les deux tâches suivantes :

– La vérification d'*instance* : vérifier si un individu a d'une ABox $\mathcal{A}$ est une instance d'une description de concept donnée C ($a \in C^{\mathcal{I}}$), écrit $\mathcal{A} \models C(a)$.

– La vérification de *consistance* : Un ABox $\mathcal{A}$ est *consistante* par rapport à une TBox $\mathcal{T}$, s'il existe une interprétation qui est un modèle des deux $\mathcal{A}$ et $\mathcal{T}$.

Dans ce rapport, nous examinons uniquement l'inférence sur la TBox. Nous allons transformer le test de subsomption des concepts en test d'insatisfaisabilité de description de concept : $C \sqsubseteq D$ par rapport à $\mathcal{T}$ si et seulement si $(C \sqcap \neg D)$ n'est pas satisfaisable par rapport à $\mathcal{T}$.

2.2 Approches du raisonnement

À partir de la technique employée dans le développement des algorithmes de décision pour le problème d'inférence, nous pouvons les catégoriser en deux groupes. Le premier est appelé *algorithmes structurels*, i.e., les algorithmes qui comparent la structure syntaxique des concepts, pour résoudre le problème de subsomption de concept dans quelques LDs primitives. Ces algorithmes, toutefois, ne sont pas applicables pour les LDs avec la négation, la disjonction,..., e.g., les LDs de famille-$\mathcal{S}$. Pour de tels langages, on a donné le deuxième groupe qui est appelé *algorithmes de tableaux*. Les algorithmes de tableaux ont été donnés la première fois par Schmidt-Schauß et Smolka, et ils deviennent réellement un outil principal pour les problèmes de satisfaisabilité et de subsomption de concept dans les LDs.

2.2.1 Raisonnement basé sur la comparaison structurelle

Les premières recherches des propriétés du calcul de langages terminologiques ont été proposées par Brachman et Levesque [BL84]. Les algorithmes du calcul de subsomption sont basés sur la comparaison structurelle entre les expressions de concepts. L'idée de cette approche est que si deux expressions de concept sont faites de sous-expressions, alors elles peuvent être comparées séparément en comparant une sous-expression d'un concept avec toutes celles de l'autre. Nous examinons cet algorithme pour vérifier la subsomption dans le langage $\mathcal{FL}^-$. L'algorithme s'exécute en deux phases : premièrement, les concepts sont ré-écrits dans une forme normale (déplier les concepts et factoriser les rôles), et ensuite leurs structures sont comparées :

- Toutes les conjonctions emboîtées sont égalisées, c.-à-d., $A \sqcap (B \sqcap C) \Longleftrightarrow A \sqcap B \sqcap C$. Toutes les conjonctions de quantifications universelles sont factorisées, c.-à-d., $\forall P.C \sqcap \forall P.D \Longleftrightarrow \forall P.(C \sqcap D)$. Les concepts ré-écrits sont logiquement équivalents avec les précédents, donc la subsomption est préservée par cette transformation.
- Soient $C = C_1 \sqcap C_2 \sqcap ... \sqcap C_m$ et $D = D_1 \sqcap D_2 \sqcap ... \sqcap D_n$, alors D subsume C si et seulement si pour chaque D_i, il existe un C_j avec :
  - Si D_i est un concept atomique, ou est un concept de forme $\exists P$, alors $D_i = C_j$.
  - Si D_i est un concept de forme $\forall P.D'$, $C_j = \forall P.C'$ (le même rôle atomique P), alors $C' \sqsubseteq D'$.

Exemple 2.1 :

On peut facilement vérifier que la subsomption suivante est satisfaite selon cet algorithme :

FEMME⊓∀ avoirEnfant.PROFESSEUR ⊑ PARENT⊓∀ avoirEnfant.∀ employer.UNIVERSITE

parce que FEMME⊑ PARENT et PROFESSEUR ⊑ employer.UNIVERSITE On considère un autre exemple avec un langage plus expressif :

Exemple 2.2 :

$$(\geq 1 \text{ avoirEnfant.FEMME}) \sqcap (\geq 2 \text{ avoirEnfant.HOMME}) \sqsubseteq (\geq 3 \text{ avoirEnfant})$$

Le concept ($\geq$ 3 avoirEnfant) subsume ni l'un ni l'autre composant dans la côte gauche, la comparaison structurelle dans ce cas ne révélerait pas la relation du subsomption.

Cette approche a été principalement utilisée dans des systèmes comme KL-ONE, K-REP, BACK, et LOOM [BS85, MDW91, Pel91, Mac91]. Ces algorithmes de comparaison structurelle sont très efficaces (polynomiaux) pour des langages de LD très peu expressifs (comme $\mathcal{FL}^-$). Un autre avantage est son emploi dans les inférences non-standard (voir [LD04]). Pour

les langages de LD plus expressifs, ils ne peuvent pas détecter toutes les relations de subsomption et d'instance existantes. Autrement dit, ces algorithmes sont incomplets par rapport aux sémantiques de langage de la logique du première-ordre, si le langage permet les combinaisons particulières de constructeurs. Toutefois, les algorithmes structurels peuvent être améliorés en tenant compte de combinaisons particulières de constructeurs [Neb90], mais la difficulté est de prouver la complétude de ces algorithmes (c.-à-d. prouver que des comparaisons ont été considérées pour capturer toutes les interactions possibles).

2.2.2 Raisonnement basé sur l'algorithme de tableaux

Cette technique de raisonnement est basée sur les calculs de tableaux pour la logique des prédicats du premier-ordre [Smu68]. Son rôle principal est l'adaptation de calculs aux analyses de la complexité. Les structures de tableaux obtenues en raisonnant avec un langage de LD donné sont soigneusement analysées, et les vérifications redondantes dans les tableaux sont éliminées afin de donner une limite supérieure stricte sur la complexité de la méthode.

L'idée principale du calcul de tableau est de vérifier si une formule donnée F est une conséquence logique d'une théorie donnée T. On essaye de construire, avec les *règles de propagation* convenables, le modèle le plus générique de T où F est faux. Si le modèle est construit avec succès, alors la réponse est NON (parce que F n'est pas un conséquence logique de T, donc la méthode est complète) ; si le modèle construit n'est pas un succès, alors la réponse est OUI (parce qu'il n'existe pas un modèle de T avec F faux, donc F est réellement une conséquence logique de T et la méthode est correcte). Les règles de propagation viennent directement de la sémantique de constructeurs.

D'une manière générale, l'algorithme de tableaux appliqué dans une LD [HNSS90] essaye de prouver la satisfaisabilité d'une expression de concept D en démontrant l'existence d'un modèle - une interprétation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ dans laquelle $D^{\mathcal{I}} \neq \emptyset$. On peut proposer des algorithmes de décision de satisfaisabilité et de subsomption qui sont corrects et complets pour les langages de LDs très expressifs. Toutefois l'investigation de la différence entre l'expressivité de LDs et la complexité de leurs problèmes d'inférence [Fra03] reste l'une des questions les plus importantes dans la recherche de LD. Cette technique sera abordée avec plus en détail dans la Section 2.4.2.

2.3 Raisonnement sur les TBox

Les services principaux du raisonnement de base sur la TBox se composent de la satisfaisabilité de concept et la subsomption de concept. Nous adressons ici ces deux problèmes dans l'hypothèse de l'absence d'ABox-Assertions, c.-à-d., la base de connaissances est dans

la forme $\langle \mathcal{T}, \emptyset \rangle$. Cette section rappelle brièvement le raisonnement dans trois types de TBox (comme nous avons dit dans la section 1.6.1) : TBox primitive, TBox simple et TBox générale [DLNS96]. Néanmoins, comme ce mémoire se concentrera donc sur le raisonnement dans une TBox générale, les bases de connaissances dans les applications réalisées sont représentées par les TBox générales.

2.3.1 TBox primitives

Rappelons qu'une TBox simple $\mathcal{T}$ est l'ensemble des définitions de concepts de la forme $A \equiv C$, qui obéit à la restriction syntaxique suivante : pour chaque nom de concept A, il y a tout au plus une définition de A dans $\mathcal{T}$. Dans [BMD+94] les systèmes de LD sont introduits comme un ensemble de spécifications de concept primitif dans un langage obtenu de $\mathcal{FL}^-$ en ajoutant les deux constructeurs $(\leq\ 1P)$ et $(\geq\ 1P)$, où P est un rôle atomique. La subsomption entre les concepts atomiques peut être décidée en temps polynomial. Quand le langage est enrichi avec les rôles inverses ou avec la négation de concepts atomiques, la subsomption devient un problème NP-difficile.

2.3.2 TBox simples

Toutes les définitions de concept ($A \equiv C$) dans une TBox simple doivent finalement se fonder sur quelques concepts et rôles primitifs qui sont des éléments atomiques sans définition. Donc, dans une TBox simple, on distingue les concepts primitifs et les concepts définis. Un problème posé est : pour une TBox simple $\mathcal{T}$, y a t-il une manière unique de déterminer l'interprétation des concepts définis à partir de l'interprétation des concepts primitifs. Appelons une interprétation initiale $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ dont la fonction d'interprétation envoie chaque concept primitif dans $\mathcal{T}$ à un sous-ensemble de $\Delta^{\mathcal{I}}$, et chaque rôle à un sous-ensemble de $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Étant donné une première interprétation $\mathcal{I}$, nous obtenons une interprétation pour $\mathcal{T}$ en étendant la fonction d'interprétation $.^{\mathcal{I}}$ à tous les concepts et rôles dans $\mathcal{T}$.

La réponse à ce problème dépend de la forme de TBox, et plus particulièrement, la TBox acyclique ou la TBox cyclique.

– TBox simples cycliques Les TBox simples cycliques apparaissent quand un concept est défini en se référant directement ou indirectement à lui-même comme dans la définition de concept HUMAIN : un HUMAIN est défini comme un MAMMIFERE avec exactement deux PARENTS et tous les PARENTS sont HUMAINS.

$$\text{HUMAIN} \equiv \text{MAMMIFERE} \sqcap \exists \geq 2\text{parent} \sqcap \exists \leq 2\text{parent} \sqcap \forall \text{parent.HUMAIN}$$

Dans les TBox simples cycliques, il n'y a aucune manière unique d'étendre l'interpré-

tation initiale de TBox $\mathcal{T}$ à un modèle pour $\mathcal{T}$. Par exemple, une TBox $\mathcal{T}$ constituée par une définition cyclique $C \equiv A \sqcap \forall R.C$, et par une interprétation initiale $\mathcal{I}'$ tels que $\Delta^{\mathcal{I}'} = \{a, b, c, d, e\}, A^{\mathcal{I}'} = \{a, b, c, d, e\}$, et $R^{\mathcal{I}'} = \{(a, a), (a, b), (c, d), (c, e), (a, d)\}$. On peut facilement indiquer deux manières pour étendre $\mathcal{I}'$ à un modèle $\mathcal{I}$ de $\mathcal{T}$:

1. $\mathcal{I}'$ est étendu à $\mathcal{I}$ tel que $C^{\mathcal{I}} = \{a, c, d\}$
2. $\mathcal{I}'$ est étendu à $\mathcal{I}$ tel que $C^{\mathcal{I}} = \{a, b, c, d, e\}$

Dans [Baa90, Neb91], trois manières d'interprétation de telles définitions cycliques sont discutées selon une des trois sémantiques : sémantique descriptive, sémantique du plus petit point-fixe (least fixpoint semantics) et sémantique du plus grand point-fixe (greatest fixpoint semantics). Les propriétés de calcul sont étudiées et il s'avère que, pour le langage $\mathcal{FLN}^-$, la subsomption dans les TBox cycliques est PSPACE-complet par rapport aux deux sémantiques du plus petit point-fixe et du plus grand point-fixe. En ce qui concerne la sémantique descriptive, la subsomption est dans coNP-difficile et dans PSPACE, mais on ne sait pas s'il est complet pour PSPACE. Toutefois, Nebel [Neb91] a également montré que l'utilisation sans restriction des TBox cycliques peut mener à l'indécidabilité. Concrètement, la subsomption par rapport aux sémantiques descriptives et du plus grand point-fixe dans les $\mathcal{TLC}$-terminologies générales est indécidable. Donc ces résultats d'indécidabilité signifient que les cycles terminologiques ne sont pas toujours tolérables. Pour ces raisons, dans ce mémoire, nous considérons seulement les terminologies acycliques.

– TBox simples acycliques : Les TBox simples acycliques (appelées brièvement les TBox acycliques) sont omniprésentes dans des systèmes de LD. En effet, la plupart des systèmes existants ne permettent pas la spécification de TBox cycliques. Dans [Neb90], Nebel a montré que : étant donnée une TBox acyclique $\mathcal{T}$, n'importe quelle interprétation initiale peut être étendue à tout au plus un modèle de $\mathcal{T}$. Ceci signifie qu'il n'y a aucune ambiguïté en interprétant un concept défini, sur la base de l'interprétation pour des concepts primitifs. D'ailleurs, il est facile de voir que la subsomption dans les TBox simples peut être réduite à la subsomption dans une terminologie vide. En effet, si nous voulons vérifier si C est subsumé par D dans une TBox acyclique $\mathcal{T}$, nous pouvons itérativement substituer chaque occurrence de n'importe quel concept défini A dans C et D par l'expression correspondante $f_{\mathcal{T}}(A)$ [1], obtenant les deux expressions C', D'. Puisque $\mathcal{T}$ est acyclique, ce processus se termine en un nombre fini d'itérations, et $\langle \mathcal{T}, \emptyset \rangle \models C \sqsubseteq D$ si et seulement si $\langle \emptyset, \emptyset \rangle \models C' \sqsubseteq D'$.

Ce résultat montre que le raisonnement avec les TBox acycliques peut être réduit au

[1] une fonction totale $f_{\mathcal{T}}$ de concepts atomiques à des expressions de concept est définie : $f_{\mathcal{T}}(A) = C$, si $A \equiv C \in \mathcal{T}$, et $f_{\mathcal{T}}(A) = A$, si A est primitif dans $\mathcal{T}$

raisonnement avec les expressions de concept. Nebel ([Neb90]) prouve également que la méthode ci-dessus peut mener à un coût exponentiel pour l'algorithme de subsumption par rapport à la taille de la TBox. Cependant, dans le même papier, cet auteur montre que l'indocilité ne provient pas de la méthode, mais de la complexité inhérente du problème : la détermination de la subsomption entre C et D dans une TBox acyclique est Co-NP-complète, même si le langage exprimé C, D et $\mathcal{T}$ est moins expressif que $\mathcal{FL}^-$. Mais selon Nebel, le pire des cas exponentiel est improbable, et donc les algorithmes de subsomption dans des TBox simples acycliques se sont bien comportés dans la pratique.

2.3.3 TBox générales

La TBox générale se compose d'un ensemble d'inclusions de concepts ($C \sqsubseteq D$) et d'équations de concept ($C \equiv D$). Le raisonnement dans les TBox générales exige des techniques qui sont différentes que celles des TBox primitives et simples. Ceci est aussi confirmé par les résultats de complexité : pour le langage $\mathcal{FL}^-$, tandis que le raisonnement dans les TBox simples adoptant la sémantique descriptive est PSPACE, le raisonnement dans les TBox générales est EXPTIME-difficile. Cependant, cette distinction disparaît pour les langages qui contiennent l'union et la négation. Afin d'évaluer la subsomption par rapport à une terminologie générale, l'algorithme de raisonnement peut être étendu avec l'addition d'une *méta-contrainte* et d'une stratégie plus sophistiquée appelée *blocage*.

2.4 Algorithmes de tableaux

2.4.1 Formes normales de concept

Un concept est dit être dans la *forme normale de négation* ou dans la forme *simple* s'il contient seulement les compléments de la forme $\neg A$, où A est un concept primitif. Les concepts quelconques peuvent être transformés en formes simples par les règles suivantes :

$$
\begin{array}{lcl}
\neg\top & \longrightarrow & \bot \\
\neg\bot & \longrightarrow & \top \\
\neg(C \sqcap D) & \longrightarrow & \neg C \sqcup \neg D \\
\neg(C \sqcup D) & \longrightarrow & \neg C \sqcap \neg D \\
\neg\neg C & \longrightarrow & C \\
\neg(\forall R.C) & \longrightarrow & \exists R.\neg C \\
\neg(\exists R.C) & \longrightarrow & \forall R.\neg C \\
\neg(\leq nR) & \longrightarrow & \geq (n+1)R \\
\neg(\geq nR) & \longrightarrow & \begin{cases} (\forall R.\bot) & si\ n = 1 \\ \leq (n-1)R & si\ n > 1 \end{cases} \\
\neg(\leq nR.C) & \longrightarrow & \geq (n+1)R.C \\
\neg(\geq nR.C) & \longrightarrow & \begin{cases} (\forall R.\bot) & si\ n = 1 \\ \leq (n-1)R.C & si\ n > 1 \end{cases}
\end{array}
$$

Si C' est un concept simple qui est obtenu de C en utilisant les règles au-dessus, alors nous disons que C' est la forme normale de négation de C. L'idée de considérer des concepts dans cette forme a été donnée dans [SSS91].

Proposition 2.3. *[SSS91] Pour chaque concept, on peut calculer en temps linéaire sa forme normale de négation qui est équivalente au concept d'origine.*

2.4.2 Algorithme de tableaux pour $\mathcal{ALC}$

Nous rappelons l'algorithme de tableaux pour le langage $\mathcal{ALC}$ avec une terminologie générale. La méthode des tableaux peut être mieux illustrée en décrivant un algorithme pour décider la satisfaisabilité des expressions de concept d'$\mathcal{ALC}$.

La vérification de satisfaisabilité par rapport à une TBox générale $\mathcal{T} = \{C_1 \sqsubseteq D_1, ..., C_n \sqsubseteq D_n\}$ est nécessaire pour restreindre les modèles $\mathcal{I}$ produits par l'algorithme à ceux qui satisfont $\mathcal{T}$:

$$C_i \sqsubseteq D_i \Longleftrightarrow (C_i \sqcap \neg D_i)^{\mathcal{I}} = \emptyset \text{ pour tous les modèles } \mathcal{I} \text{ de } \mathcal{T} \text{ et } i \in \{1, ..., n\}$$

où un modèle $\mathcal{I}$ satisfait une TBox $\mathcal{T}$ s'il satisfait tous les axiomes dans $\mathcal{T}$. Tous les axiomes peuvent être transformés en les ICGs équivalents :

$$C \equiv D \Leftrightarrow \{C \sqsubseteq D, D \sqsubseteq C\}$$
$$C \sqsubseteq D \Leftrightarrow C \sqcap \neg D \equiv \bot$$

Un ICG $C \sqsubseteq D \in \mathcal{T}$ est satisfait par un modèle $\mathcal{I}$ ssi $C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \Longleftrightarrow (C \sqcap \neg D)^{\mathcal{I}} = \emptyset \Longleftrightarrow (\neg C \cup D)^{\mathcal{I}} = \Delta^{\mathcal{I}}$ [Hor97].

L'algorithme emploie un arbre **T** pour représenter le modèle construit. Chaque noeud x dans l'arbre représente un individu et est étiqueté par un ensemble $\mathcal{L}(x)$ d'expressions de concept d'$\mathcal{ALC}$ qu'il doit satisfaire :

$$C \in \mathcal{L}(x) \Rightarrow x \in C^{\mathcal{I}} \text{ (ou écrit } x : C)$$

Chaque arc $\langle x, y\rangle$ dans l'arbre représente une paire d'individus dans l'interprétation d'un rôle et est étiqueté avec le nom du rôle :

$$R = \mathcal{L}(\langle x, y\rangle) \Rightarrow \langle x, y\rangle \in R^{\mathcal{I}} \text{ (ou écrit } xRy)$$

$x : C$, xRy sont également les contraintes, et l'arbre **T** correspond à un système de contraintes. Intuitivement, $x \in C^{\mathcal{I}}$ ($x : C$) dit que x est dans l'interprétation de C et xRy dit que la paire (x, y) est dans l'interprétation de R ($(x, y) \in R^{\mathcal{I}}$).

Supposons qu'un concept C est représenté dans un langage $\mathcal{ALC}$. Un tableau $\mathcal{ALC}$ doit satisfaire les conditions suivantes :

- T1 : Si $C \in \mathcal{L}(x)$, alors $\neg C \notin \mathcal{L}(x)$
- T2 : Si $C_1 \sqcap C_2 \in \mathcal{L}(x)$, alors $C_1 \in \mathcal{L}(x)$ et $C_2 \in \mathcal{L}(x)$,
- T3 : Si $C_1 \sqcup C_2 \in \mathcal{L}(x)$, alors $C_1 \in \mathcal{L}(x)$ ou $C_2 \in \mathcal{L}(x)$,
- T4 : Si $\forall r.C \in \mathcal{L}(x)$ et $r \in \mathcal{L}(\langle x, y\rangle)$, alors $C \in \mathcal{L}(y)$,
- T5 : Si $\exists r.C \in \mathcal{L}(x)$, alors il existe y tels que $r \in \mathcal{L}(\langle x, y\rangle)$ et $C \in \mathcal{L}(y)$.

Une stratégie des algorithmes de tableaux est l'*"internalisation"* avant de mettre en application le raisonnement. Étant donné un axiome $E \sqsubseteq F$, son *concept internalisé* est $C_{E\sqsubseteq F} = \neg E \sqcup F$; pour une TBox $\mathcal{T}_i$, son concept internalisé est $\mathcal{M} = C_{\mathcal{T}} = \bigsqcap_{E\sqsubseteq F\in\mathcal{T}} \neg E \sqcup F$ ($\mathcal{M}$ est appelé *méta-contrainte* [Hor97]). Aussi bien que la hiérarchie de rôle $\mathcal{R}_{\mathcal{T}}$ contient les axiomes de rôle de $\mathcal{T}$, en complétant des axiomes $P \sqsubseteq U$, pour chaque rôle P de $\mathcal{T}_i$, avec un rôle universel U.

Pour déterminer la satisfaisabilité d'une expression de concept D par rapport à une terminologie générale $\mathcal{T}$ d'$\mathcal{ALC}$, un arbre **T** est initialisé à un seul noeud x_0, avec $\mathcal{L}(x_0) = \{D, \mathcal{M}\}$, et est étendu en appliquant à maintes reprises les règles de la Table 2.1, **T** est complètement étendu quand aucune des règles ne peut être appliquée. **T** contient une contradiction évidente quand, pour un noeud x et un concept C, $\bot \in \mathcal{L}(x)$ ou $\{C, \neg C\} \subseteq \mathcal{L}(x)$.

Par conséquent, les arbres qui sont construits par l'algorithme de tableaux peuvent être restreints à ceux qui représentent des modèles qui satisfont les ICGs en imposant une méta-contrainte : pour tout noeud x dans **T**, $(\neg C \sqcup D) \in \mathcal{L}(x)$. L'algorithme se termine quand **T** est complet (aucune règle d'expansion ne peut être appliquée) ou quand une contradiction évidente a été décelée. Le non-déterminisme est négocié en cherchant des expansions possibles différentes : le concept est insatisfaisable si chaque expansion mène à une contradiction et est satisfaisable si toute expansion possible mène à la découverte d'un arbre de contradiction-libre (clash-free) complet.

$\sqcap$-règle	si	1.	$(C_1 \sqcap C_2) \in \mathcal{L}(x)$
		2.	$\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
	alors		$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-règle	si	1.	$(C_1 \sqcup C_2) \in \mathcal{L}(x)$
		2.	$\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
		3.	$D = C_1$ ou $D = C_2$
	alors		$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup D$
$\exists$-règle	si	1.	$\exists R.C \in \mathcal{L}(x)$
		2.	il n'y a aucun y tels que $\mathcal{L}(\langle x, y\rangle) = R$ et $C \in \mathcal{L}(y)$
			alors créer un noeud nouveau y et arc $\langle x, y\rangle$
		avec	$\mathcal{L}(y) = \{C\}$ et $\mathcal{L}(\langle x, y\rangle) = R$
$\forall$-règle	si	1.	$\forall r.C \in \mathcal{L}(x)$
		2.	il n'y a aucun certain y tels que $\mathcal{L}(\langle x, y\rangle) = R$ et $C \notin \mathcal{L}(y)$
		alors	$\mathcal{L}(y) \longrightarrow \mathcal{L}(y) \cup \{C\}$

Tab. 2.1 : Règles d'expansion de tableaux pour $\mathcal{ALC}$

$\exists$-règle	si	1.	$\exists R.C \in \mathcal{L}(x)$
		2.	il n'y a aucun y tels que $\mathcal{L}(\langle x, y\rangle) = R$ et $C \in \mathcal{L}(y)$
		3.	il n'y a aucun y tels que y est un ancêtre de x et $\mathcal{L}(x) \subseteq \mathcal{L}(y)$
			alors créer un noeud nouveau y et arc $\langle x, y\rangle$
		avec	$\mathcal{L}(y) = \{C, \mathcal{M}\}$ et $\mathcal{L}(\langle x, y\rangle) = R$

Tab. 2.2 : $\exists$-Règle modifiée avec la méta-contrainte

Pour les LDs expressives, e.g., la famille-$\mathcal{S}$ de LDs ($\mathcal{ALC}_{R^+}$), on a proposé une technique appelée "*blocage*" (blocking) [BDS93] qui est employée pour assurer la terminaison de l'algorithme de tableaux. Pour expliquer le blocage, nous introduirons quelques notations :

- Le *père* d'un noeud y est un noeud x si $\langle x, y\rangle$ est un arc dans **T** ;
- L'*ancêtre* est la fermeture transitive du père.
- Une *règle génératrice* est une règle d'expansion de tableau qui étend l'arbre **T** en ajoutant un noeud ou plusieurs noeuds nouveaux. Dans le cas de $\mathcal{ALC}$, $\exists$-règle est la règle génératrice.

Un noeud y est dit bloqué s'il existe un noeud d'ancêtre x tel que $\mathcal{L}(y) \subseteq \mathcal{L}(x)$; dans ce cas, x est appelé le *noeud de blocage*.

Le blocage impose une nouvelle condition sur les règles génératrices : la règle peut seulement être appliquée avec un noeud y s'il n'a pas un noeud d'ancêtre x tel que $\mathcal{L}(y) \subseteq \mathcal{L}(x)$. Intuitivement, il peut être vu que la terminaison est garantie, parce que une terminologie finie peut seulement produire un nombre fini d'expressions de concepts différentes et donc un nombre fini d'ensembles d'étiquettes différentes, tous les noeuds doivent donc être bloqués. La table 2.2 illustre une $\exists$-règle modifiée qui ajoute $\mathcal{M}$ aux nouveaux noeuds et intégre le blocage. Notez que les LDs différentes peuvent utiliser des genres différents de techniques de blocage [Hor97].

2.5 Conclusion

Nous avons examiné les thèmes les plus importants concernant les problèmes du raisonnement dans la construction des systèmes de représentation de connaissances basés sur la logique de description. Le raisonnement se concentre sur les TBox générales avec l'algorithme de tableaux. En raison de l'indocilité du raisonnement par rapport aux TBox générales, qui peut entraîner un coût exponentiel, des améliorations de la performance du raisonnement sont nécessaires. Quelques techniques d'optimisation seront introduites dans le chapitre suivant.

Chapitre 3

Techniques d'optimisation du raisonnement dans la logique de description

En vue de garantir un comportement raisonnable et prévisible d'un système de LD, ces problèmes de raisonnement devraient au moins être décidables pour la LD employée par le système, et ils préfèrent une complexité faible. Néanmoins, la décidabilité et la complexité des problèmes d'inférence dépendent de la puissance de l'expressivité de LD. Les investigations dans [LB87, Neb88, BDS93, SSS91] ont montré que la détermination de subsomption est au moins NP-difficile voire indécidable avec des LDs très expressives. Par conséquent, les techniques d'optimisation qui améliorent la performance de l'algorithme de subsomption sont nécessaires pour les recherches dans les LDs.

Dans ce chapitre, nous présentons quelques techniques d'optimisation du raisonnement qui ont été développées et qui sont largement connues et employées dans différents systèmes de LD. Dans les logiques de description, les logiques temporelles, les logiques modales, il est souvent utile pour le raisonnement de considérer des axiomes représentant des vérités universelles d'un domaine d'application. Dans les expériences avec un ensemble arbitraire d'axiomes, des systèmes sans optimisation s'exécutent très mal, bien plus lentement que les systèmes optimisés, même pour des logiques relativement inexpressives. Dans plusieurs cas, le temps du traitement coûte des heures (dans certains cas même des centaines d'heures) pour des systèmes sans optimisation tandis que les systèmes optimisés prennent seulement quelques milli-secondes pour traiter le même problème. Par conséquent, il est essentiel d'étudier les techniques d'optimisation pour traiter efficacement le raisonnement dans des applications réelles. C'est particulièrement approprié dans le cas des logiques de description, où un moteur d'inférence traite les requêtes spécifiques en utilisant son propre algorithme d'essai de subsomption (ou, équivalence, ou satisfaisabilité) par rapport à une terminologie comme un problème fondamental. Les techniques

pour améliorer la performance du raisonnement dans une LD peuvent donc être réalisées naturellement en trois niveaux :

- L'optimisation de l'algorithme (niveau algorithmique) : techniques pour réduire les exigences de stockage de l'algorithme, et réduire le nombre d'essais de subsomption [HPS99a] pour optimiser des algorithmes de tableaux.
- L'optimisation de terminologie (niveau conceptuel) : techniques pour optimiser les structures d'axiomes dans la terminologie.
- L'optimisation de requête (niveau utilisateur) : techniques pour optimiser les requêtes d'utilisateur.

Dans ce chapitre, nous résumons quelques techniques d'optimisation aux niveaux algorithmique et conceptuel qui ont été appliquées dans des systèmes très connus comme FACT [1] [Hor98, HPS98]. Une nouvelle technique sera proposée dans les chapitres suivants de cette thèse.

3.1 Techniques d'optimisation du raisonnement

Le raisonnement dans les LDs repose essentiellement sur l'algorithme de tableaux. L'algorithme de tableaux classique est trop lent pour former la base d'un système utile de logique de description. Nous avons donc étudié et utilisé un rayon connu d'adaptation et d'optimisations du procédé qui améliorent l'exécution de l'algorithme du test de satisfaisabilité. Dans cette section, nous allons rappeler quelques techniques d'optimisation [HPS99a, Hor97] utilisées dans le système FACT [HPS98].

3.1.1 Optimisation algorithmique

3.1.1.1 Déploiement paresseux

Dans la plupart des terminologies réalistes, les grands concepts et les concepts complexes sont classés dans une hiérarchie de concepts nommés (concepts primitifs, concepts atomiques et concepts définis) dont les descriptions sont moins complexes. L'algorithme de tableaux classique traite souvent les expressions de concept qui sont entièrement déployées, c.-à-d., une contradiction est seulement détectée quand un concept primitif et sa négation sont présentés dans le label d'un même noeud. Un concept A est dit être déployé s'il satisfait les trois conditions suivantes [THPS07] :

- Le concept A est dans une des deux formes $A \sqsubseteq C$ ou $A \equiv C$.

[1] Fast Classification of Terminologies

– La définition de A est unique, c.-à-d., pour chaque nom de concept A, il y a tout au plus une expression de la forme $A \equiv C$, et s'il y a une expression de la forme $A \equiv C$ alors il n'existe pas une expression de la forme $A \sqsubseteq C$.
– Il n'y a aucune expression de la forme $A \equiv C$ telle qu'A se produit dans C (une expression cyclique). Un nom de concept A *se produit (apparaît)* dans une expression de concept C si A se produit syntactiquement dans C, ou il y a un nom de concept A' tels qu'A' se produit syntactiquement dans C, et $A' \equiv C'$ tels qu'A se produit dans C'.

L'idée du déploiement paresseux est apparue parce que dans la réalité, les expressions de concept sont déployées selon les exigences du progrès de l'algorithme. Par exemple, on examine la satisfaisabilité de l'expression de concept :

$$\exists R.C \sqcap \forall R.\neg C$$

Si C est un concept primitif alors une contradiction sera détectée dès que l'algorithme crée un R-successeur y parce que $\{C, \neg C\} \in \mathcal{L}(y)$.

Si C est un nom de concept, le déploiement de C peut être retardé jusqu'à ce que le $\exists$-règle ait créé un R-successeur y avec $\{C, \neg C\} \in \mathcal{L}(y)$. Dans ce cas une contradiction peut être immédiatement détectée sans déploiement de C. Cette méthode, appelée *déploiement paresseux* (lazy unfolding), est plus efficace car elle épargne beaucoup de travail gaspillé du déploiement de C. De plus, si C est une expression de concept, alors la contradiction ne sera pas détectée immédiatement en transformant C en forme normale de négation. Pour une expression large et complexe C, on peut appliquer la méthode du déploiement paresseux en la transformant en une *forme normalisée lexicale* qui sera introduite dans la section suivante.

3.1.1.2 Retour sur la direction de dépendance

La plupart des systèmes du raisonnement dans les LDs emploient souvent une approche de haut en bas (top-down) pour ranger en ordre d'application des règles d'expansion. Dans cette approche, des règles génératrices (comme $\geq$ - règle et $\exists$ -règle) sont appliquées après toutes les autres. Nous considérons, par exemple, un concept C dans la forme :

$$(C_1 \sqcup D_1) \sqcap ... \sqcap (C_n \sqcup D_n) \sqcap \exists R.(A \sqcap B) \sqcap \forall R.\neg A$$

Un noeud x serait étendu et étiqueté par

$$\mathcal{L}(x) = \{(C_1 \sqcup D_1), ..., (C_n \sqcup D_n), \exists R.(A \sqcap B), \forall R.\neg A\}$$

L'algorithme de tableaux étendra premièrement n disjonctions et 2^n choix d'expansion de disjonction différents seraient essayés avant que le concept soit déterminé être insatisfaisable par la

∃-règle. Afin d'éviter ce comportement exponentiel pour vérifier la satisfaisabilité de C, une solution sophistiquée est proposée, appelée *le retour sur la direction de dépendance* (dependency directed backtracking or backjumping) [Hor98]. Pour une raison succincte, nous utilisons le "*backjumping*" pour cette méthode.

Le backjumping opère en étiquetant chaque expression de concept C dans l'étiquette d'un noeud x avec un ensemble de dépendances $D_C(x)$ qui indique des points d'embranchement (appelés des ⊔-noeuds, i.e., les noeuds avec ⊔-successeurs) desquels ils dépendent. Une expression de concept dépend d'un point d'embranchement quand elle a été ajoutée à l'étiquette de point d'embranchement x par l'algorithme de recherche, ou quand elle dépend d'une autre expression de concept qui dépend de x. Une expression de concept C dépend d'expression de concept E quand C a été ajouté à une étiquette de noeud par une expansion déterministe (une application de ⊓-règle ou ∃-règle) qui a utilisé E. Par exemple, si $A \in \mathcal{L}(x)$ est présenté par l'expansion de $(A \sqcap B) \in \mathcal{L}(x)$, alors $A \in \mathcal{L}(x)$ dépend de $(A \sqcap B) \in \mathcal{L}(x)$
Dans le cas où l'arbre complet contient un certain noeud x avec $\{C, \neg C\} \in \mathcal{L}(x)$ (une contradiction est détectée), nous utilisons $D_C(x)$ et $D_{\neg C}(x)$ pour identifier le point d'embranchement le plus récent où l'exploration d'une autre branche peut alléger la cause de contradiction. L'algorithme peut sauter alors en arrière sur des points d'embranchement sans exploration de branches alternatives. Une explication plus détaillée de la réalisation de cette technique peut être trouvée dans [Hor97].

Par exemple, on étend le noeud x de l'exemple précédent, l'algorithme de recherche effectuera une séquence de n branches, il conduit éventuellement au noeud x_n avec $\{\exists R.(A \sqcap B), \forall R.\neg A\} \subset \mathcal{L}(x_n)$. Quand x_n est étendu, l'algorithme produira un R-successeur y_1 avec $\mathcal{L}(y_1) = \{(A \sqcap B), \neg A\}$. L'expression de concept $A \sqcap B$ sera alors étendue et une contradiction sera détectée parce que $\{A, \neg A\} \subset \mathcal{L}(y_1)$. L'algorithme peut retourner immédiatement l'*insatisfaisabilité* si les deux ensembles de dépendances de A et de $\neg A$ sont vides, ou faire un retour arrière au point le plus récent dont l'un des A ou $\neg A$ dépend, sans l'exploration de branches alternatives à l'un des points de branchement d'intervention. Dans cet exemple, $(n+2)$ noeuds sont explorés au lieu de $(2^n + 2^{n+1} - 1)$ noeuds.

3.1.1.3 Recherche de l'embranchement sémantique

Les algorithmes standards de tableau sont souvent inefficaces parce qu'ils emploient une technique de recherche basée sur l'embranchement syntaxique (Syntactic Branching Search). En étendant le label d'un noeud x, l'embranchement syntaxique choisit une disjonction non-expansée $(C_1 \sqcup ... \sqcup C_n)$ dans $\mathcal{L}(x)$ et cherche des modèles différents obtenus en ajoutant chaque

disjonct[2] $C_i (i \in \{1, ..n\})$ à $\mathcal{L}(x)$. Parce que les branches expansées de l'arbre de recherche ne sont pas disjointes, il n'y a aucun empêchement pour la répétition d'un disjonct insatisfaisable dans des branches différentes. Ce problème peut mener à un grand gaspillage. Par exemple, l'expansion de tableaux d'un noeud x, où $\{(C_1 \sqcup D), (C_2 \sqcup D)\} \subseteq \mathcal{L}(x)$ et D est un concept insatisfaisable. Cette expansion est présentée dans la figure 3.1, dans laquelle l'insatisfaisabilité de D doit être démontrée deux fois. Ce problème peut être traité en employant une technique d'embranche-

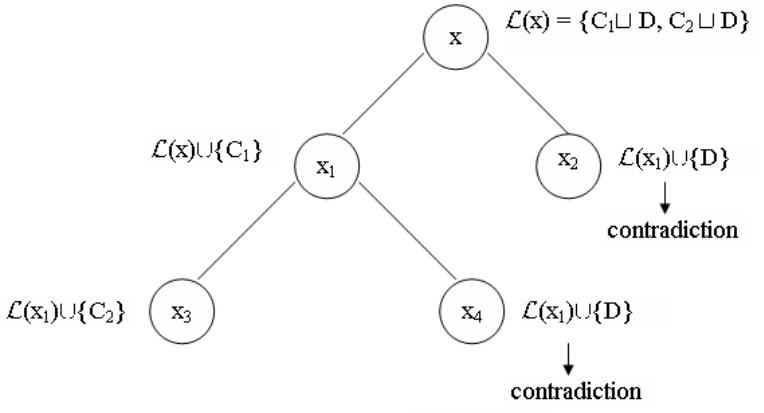


Fig. 3.1 : Embranchement syntactique

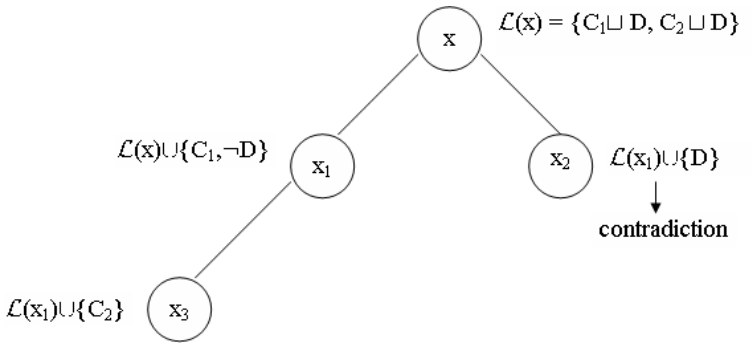


Fig. 3.2 : Embranchement sémantique

ment sémantique adaptée de la procédure de Davis-Putnam-Logemann-Loveland (DPL) qui est utilisée pour résoudre des problèmes de satisfaisabilité propositionnelle (SAT) [Fre96]. On va choisir un disjonct simple D d'une des disjoncts non-expansées dans $\mathcal{L}(x)$. Les deux sous-arbres obtenus en ajoutant D ou $\neg D$ à $\mathcal{L}(x)$ sont alors recherchés. Puisque les deux sous-arbres sont

[2]Étant donnée une expression disjonctive $(C_1 \sqcup ... \sqcup C_n)$, alors chaque membre C_i $(i \in \{1, .., n\})$ est appelé un *disjonct*

strictement disjoints, il n'y a aucune recherche gaspillée comme dans l'embranchement syntaxique. Néanmoins, avec un concept complexe D, l'addition de $\neg D$ peut amener à un espace de recherche plus grand. Mais comme dans [HPS99b], les auteurs ont montré que ceci ne semble pas être un problème significatif dans la pratique, et l'embranchement sémantique est aussi une méthode efficace.

3.1.1.4 Propagation de contrainte booléenne

La propagation de contrainte booléenne est une technique employée pour maximiser l'expansion déterministe avant d'exécuter l'expansion non déterministe (embranchement). Donc on peut élaguer l'arbre de recherche via la détection de contradiction. L'idée principale est qu'avant d'appliquer la $\sqcup$-règle à l'étiquette d'un noeud x, la PCB étend déterminément les disjonctions dans $\mathcal{L}(x)$ qui présentent seulement une possibilité d'expansion et détecte une contradiction quand une disjonction dans $\mathcal{L}(x)$ n'a aucune possibilité d'expansion. Le nombre de possibilités d'expansion présenté par une disjonction $(C_1 \sqcup ... \sqcup C_n) \in \mathcal{L}(x)$ est égale au nombre de disjoncts C_i tel que $\neg C_i \notin \mathcal{L}(x)$. PCB applique la règle d'inférence suivante :

$$\frac{\neg C_1, ..., \neg C_n, C_1 \sqcup ... \sqcup C_n \sqcup D}{D}$$

Exemple 3.1 :

Étant donné un noeud x et une étiquette $\mathcal{L}(x)$:

$$\{((C_1 \sqcap C_2) \sqcup D), (\neg C_1 \sqcup \neg C_2), \neg D\} \subseteq \mathcal{L}(x)$$

PCB déterminément étend la disjonction $((C_1 \sqcap C_2) \sqcup D)$, parce que $\neg D \in \mathcal{L}(x)$:

$$\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{(C_1 \sqcap C_2)\}$$

Après $(C_1 \sqcap C_2)$ est étendu à donner $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$, PCB identifie $(\neg C_1 \sqcup \neg C_2)$ comme une contradiction parce que $C_1 \in \mathcal{L}(x)$ et $C_2 \in \mathcal{L}(x)$.

3.1.1.5 Heuristique

En exécutant des algorithmes de raisonnement, l'heuristique peut être employée pour essayer de trouver un "bon" ordre d'application des règles d'inférence. Pour les règles non-déterministes, un ordre heuristique peut explorer les différents choix d'expansion obtenus par les applications de ces règles. Le but est de choisir un ordre qui mène rapidement à la découverte d'un modèle de l'entrée (au cas où l'entrée serait satisfaite) ou à une preuve qu'aucun modèle

n'existe (au cas où l'entrée serait insatisfaite). Une des heuristiques les plus répandues dans le problème SAT est le MOMS (Maximum Occurrences in clausses of Minimum Size) [Fre95]. Une méthode possible dans les LDs est de s'embrancher sur le disjonct qui a le nombre d'apparitions maximum dans les disjonctions de taille minimum. Cette technique peut facilement être adaptée aux algorithmes de tableaux : la valeur de MOMS pour un concept de candidat C peut être calculée en comptant simplement le nombre de fois où C ou sa négation se produit dans la disjonction de taille minimum. Par exemple, si l'étiquette d'un noeud x contient les disjonctions :

$$\{(C \sqcup D_1), ..., (C \sqcup D_n)\} \subseteq \mathcal{L}(x)$$

L'heuristique MOMS choisirait C comme le disjonct qui s'est branché parce qu'il se produit dans n disjonctions de la taille 2. Le $\sqcup$-successeur dans lequel C est ajouté à $\mathcal{L}(x)$ serait exploré d'abord parce que PCB cause également $D_1, ..., D_n$ sont ajoutés à $\mathcal{L}(x)$.

3.1.1.6 Mise en cache

La combinaison de normalisation, encodage et déploiement paresseux facilite la découverte rapide d'insatisfaisabilité évidente (subsomption) mais la détection de la satisfaisabilité évidente est plus difficile pour les algorithmes de tableaux. De plus, dans les terminologies réalistes, la plupart des tests est satisfaisable et les tests satisfaisables sont généralement beaucoup plus chers.

L'idée principale de cette méthode est d'essayer d'utiliser des résultats mis en cache de tests du tableau antérieur pour démontrer la satisfaisabilité d'une expression de concept. C'est possible parce que les problèmes de satisfaisabilité dans des sous-arbres sont complètement indépendants et n'ont pas d'effet sur des noeuds d'ancêtres. De cette manière, une quantité appréciable de travail peut être épargnée lorsque les modèles larges ou complexes sont ré-utilisés.

Exemple 3.2 :

Étant donné deux concepts :

$$A \equiv C \sqcap \exists R.C_1 \sqcap \exists R_1.C_2$$

$$B \equiv \neg D \sqcup \forall R.C_3$$

La satisfaisabilité de $A \sqcap \neg B$ (c.-à-d., $A \sqsubseteq B$) peut être démontrée par un modèle combiné de deux modèles de A et $\neg B \equiv D \sqcap \exists R.\neg C_3$ en les joignant à leurs racines, comme dans la Figure 3.3. Afin de démontrer que deux modèles joints à leurs racines résultent en un modèle valide, il est

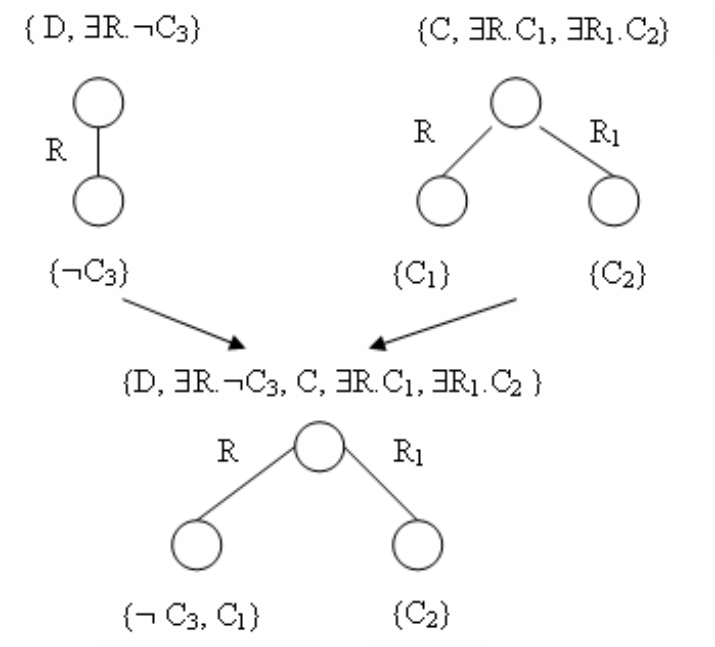


FIG. 3.3 : Fusionnement de deux modèles de A et de $\neg B$

seulement nécessaire d'examiner leurs contraintes à la racine. Deux modèles présentés par les systèmes de contrainte $\mathbf{S}^A$ et $\mathbf{S}^B$ peuvent être joints comme suit :

1. L'union de leurs noeuds de racine contient une contradiction immédiate si :

$$x_0 : C \in \mathbf{S}^A \wedge x_0 : \neg C \in \mathbf{S}^B$$

2. Une règle d'expansion de tableau supplémentaire peut être applicable quand les contraintes de la racine sont combinées :

$$x_0 R x_n \in \mathbf{S}^A \wedge x_0 : \forall R.C \in \mathbf{S}^B \wedge x_0 : \forall R.C \notin \mathbf{S}^A$$

Dans des termes plus généraux, quand on vérifie que si A est subsumé par B, la méthode de mise en cache se réalise comme suit :

- Si les modèles de $A, \neg A, B$ ou $\neg B$ n'ont pas été mis en cache, on exécute le(s) test(s) de satisfaisabilité et on met $\mathbf{S}_A, \mathbf{S}_{\neg A}, \mathbf{S}_\exists, \mathbf{S}_\forall$ et S_R en cache à partir des systèmes de contrainte complètement étendus. Si un test de satisfaisabilité est un échec, le concept testé est égal à $\bot$ et sa négation à $\top$.
- Retourne *vrai* ($A \sqsubseteq B$) ou *faux* ($A \not\sqsubseteq B$) si la subsomption évidente ou la non-subsomption

est détectée :

$$\begin{aligned} \neg B = \bot \quad &\Rightarrow vrai(A \sqsubseteq B) \\ A = \bot \quad &\Rightarrow vrai(A \sqsubseteq B) \\ B = \bot \text{ et C} \neq \bot \quad &\Rightarrow faux(A \not\sqsubseteq B) \\ \neg A = \bot \text{ et } \neg\text{B} \neq \bot \quad &\Rightarrow faux(A \not\sqsubseteq B) \end{aligned}$$

– Retourne *faux* ($A \not\sqsubseteq B$) si les modèles mis en cache de A et $\neg B$ peuvent être fusionnés. $\mathbf{S}^A$ et $\mathbf{S}^{\neg B}$ peuvent être fusionnés comme :

1. $\mathbf{S}^A_C \cap \mathbf{S}^{\neg B}_{\neg C} \neq \emptyset$
2. $\mathbf{S}^A_{\neg C} \cap \mathbf{S}^{\neg B}_C \neq \emptyset$
3. il y a certains R, S tels que $r \in \mathbf{S}^A_\exists$ et $S \in \mathbf{S}^{\neg B}_\forall$ et $R \sqsubseteq S$
4. il y a certains R, S tels que $R \in \mathbf{S}^A_\forall$ et $S \in \mathbf{S}^{\neg B}_\exists$ et $S \sqsubseteq R$
5. $\mathbf{S}^A_R \cap \mathbf{S}^{\neg B}_R \neq \emptyset$

– Réalise un test de satisfaisabilité du tableau sur $A \sqcap \neg B$ en retournant *faux* ($A \not\sqsubseteq B$) s'il est satisfaisable et *vrai* ($A \sqsubseteq B$) s'il n'est pas satisfaisable.

3.1.2 Optimisation conceptuelle

La complexité des algorithmes de raisonnement dépend de l'expressivité du langage utilisé et de la présence des ICGs. À ce niveau, l'optimisation traite directement sur la syntaxe de l'entrée. Ces optimisations servent à pré-traiter et à simplifier l'entrée dans une forme plus favorable pour le traitement suivant. Les optimisations de pré-traitement peuvent également mener une accélération significative pour le processus de raisonnement suivant. En effet, la vérification de satisfaisabilité dans les LDs expressives exige, dans le pire des cas, un temps qui est au moins exponentiel par rapport à la taille de l'entrée, tandis que la plupart des optimisations de pré-traitement ont une complexité polynômiale ou linéaire dans le pire des cas.

3.1.2.1 Simplification

La simplification est une technique employée pour réduire le nombre d'expansions non-déterministes (l'embranchement). Avant d'effectuer une expansion non-déterministe d'un label de noeud x, des disjonctions (conjonctions) de $\mathcal{L}(x)$ sont examinées, et simplifiées si possible. La simplification permet d'étendre déterminément les disjonctions dans $\mathcal{L}(x)$ qui présentent seulement une possibilité d'expansion et de détecter une contradiction quand une disjonction dans $\mathcal{L}(x)$ n'a aucune possibilité d'expansion. Cette simplification est appelée la propagation

de contrainte booléenne (PCB). En effet, la règle d'inférence $\frac{\neg C_1, ..., \neg C_n, C_1 \sqcup ... \sqcup C_n \sqcup D}{D}$ est employée pour simplifier l'expression représentée par $\mathcal{L}(x)$.

Par exemple, étant donné un noeud x avec $\{(C \sqcup (D_1 \sqcap D_2)), (\neg D_1 \sqcup \neg D_2\ ), \neg C\} \subseteq \mathcal{L}(x)$, PCB étend déterminément la disjonction (C$\sqcup(D_1 \sqcap D_2)$) parce que $\neg C \in \mathcal{L}(x)$. L'expansion déterministe de $(D_1 \sqcap D_2)$ ajoute D_1 et D_2 à $\mathcal{L}(x)$, permettant PCB d'identifier $(\neg D_1 \sqcup \neg D_2)$ comme une contradiction sans embranchement.

3.1.2.2 Normalisation lexicale et encodage

L'algorithme de tableaux suppose toujours que l'entrée est dans la forme normale de négation (FNN) ; ceci est simple pour la réalisation, mais cela veut dire que l'on doit déployer l'entrée dans la phase initiale de l'algorithme. Néanmoins, comme nous l'avons dit dans la technique d'optimisation de déploiement paresseux, la détection des contradictions peut être adressée en transformant des expressions de concepts (et leurs sous-expressions) en forme lexicale normalisée, et en identifiant des expressions lexicales équivalentes. Alors une contradiction est détectée lorsqu'une expression de concept et sa négation apparaissent dans le même label de noeud. Dans cette forme lexicale normalisée, les expressions de concepts consistent seulement en concepts atomiques, concepts de conjonction, concepts universels de rôle, et leurs négations. Afin de faciliter la découverte de telles contradictions, l'entrée doit être normalisée et simplifiée en concepts logiquement équivalents en appliquant un ensemble de règles pour ré-écrire des expressions de concept, et en rangeant les *conjoncts*[3] selon un certain ordre total. C'est à dire que les expressions de concept et les sous-expressions de concept sont normalisées et encodées récursivement de sorte que :

1. Toutes les sous-expressions sont des concepts nommés, par exemple, $\forall R.(C_1 \sqcap C_2)$ serait encodé comme $\forall R.D$, où $D \equiv C_1 \sqcap C_2$.
2. Toutes les expressions de concept sont dans une forme standard, par exemple, toutes les restrictions existentielles (les expressions en forme $\exists R.C$) sont converties en des restrictions de valeur (les expressions en forme $\neg\forall R.\neg C$).

Les transformations équivalentes se composent :

$$(C \sqcap \top) \equiv C$$
$$(C \sqcap \bot) \equiv \bot$$
$$(\forall R.\top) \equiv \top$$

[3]Étant donnée une expression conjonctive $C_1 \sqcap ... \sqcap C_n$, alors chaque membre C_i ($i \in \{1, .., n\}$) est appelé un conjonct

$$(C \sqcap C) \equiv C$$
$$(\neg\neg C) \equiv C$$
$$(C \sqcap \neg C) \equiv \bot$$
$$(\geq 1R.C) \equiv \exists R.C$$

Exemple 3.3 :

$\exists R.(C \sqcap D) \sqcap \forall R.C$ serait transformé en
$\sqcap\{\neg(\forall R.\neg \sqcap \{C, D\}), \forall R.\neg C\}$
une contradiction sera immédiatement détectée, indépendamment de la structure de C.

Dans [Hor97], une définition fonctionnelle du processus de normalisation et de codage a été donné comme dans la Table 3.1

Exemple 3.4 :

Normalisation et encodage d'expression $\exists R.(C \sqcap \neg D \sqcap \neg E) \sqcap \forall R.(\neg C \sqcup D \sqcup E)$ est réalisé comme suit :

1. Normaliser la première sous-expression :

   $Normalise(\exists R.(C \sqcap \neg D \sqcap \neg E)) \longrightarrow \neg Normalise(\forall R.\neg(C \sqcap \neg D \sqcap \neg E))$

   $Normalise(\forall R.\neg(C \sqcap \neg D \sqcap \neg E)) \longrightarrow Encode(\forall R.Normalise(\neg(C \sqcap \neg D \sqcap \neg E)))$

   $Normalise(\neg(C \sqcap \neg D \sqcap \neg E)) \longrightarrow \neg Encode(C \sqcap \neg D \sqcap \neg E)$

   $Encode(C \sqcap \neg D \sqcap \neg E) \longrightarrow CN_1$, où CN_1 est un nouveau nom de concept et $T \longrightarrow T \cup \{CN_1 \equiv (C \sqcap \neg D \sqcap \neg E)\}$

   $Encode(\forall R.Normalise\neg(C \sqcap \neg D \sqcap \neg E)) \longrightarrow CN_2$, où $T \longrightarrow T \cup \{CN_2 \equiv \forall R.\neg CN_1\}$

   $Normalise(\exists R.(C \sqcap \neg D \sqcap \neg E)) \longrightarrow \neg CN_2$

2. Normaliser la deuxième sous-expression :

   $Normalise(\forall R.(\neg C \sqcup D \sqcup E)) \longrightarrow Encode(\forall R.Normalise((\neg C \sqcup D \sqcup E))$

   $Normalise(\neg C \sqcup D \sqcup E) \longrightarrow Normalise(\neg(C \sqcap \neg D \sqcap \neg E)$

   $Normalise(\neg(C \sqcap \neg D \sqcap \neg E) \longrightarrow \neg Encode(C \sqcap \neg D \sqcap \neg E)$

   $Encode(C \sqcap \neg D \sqcap \neg E) \longrightarrow CN_1$

   $Normalise(\forall R.(\neg C \sqcup D \sqcup E)) \longrightarrow Encode(\forall R.\neg CN_1) \longrightarrow CN_2$

3. Recombiner les deux sous-expressions :

   $Normalise(\exists R.(C \sqcap \neg D \sqcap \neg E) \sqcap \forall R.(\neg C \sqcup D \sqcup E)) \longrightarrow Normalise(\neg CN_2 \sqcap CN_2) \longrightarrow \bot$

Normaliser(C) :

C = nom de concept

$\longrightarrow CN$ si Normaliser(D) = $\neg CN$

$\bot$ si Normaliser(D) = $\top$

$\top$ si Normaliser(D) = $\bot$

autrement $\neg$Normaliser(D)

$C = \forall R.D$ $\longrightarrow \top$ si Normaliser(D) = $\top$

autrement Encoder($\forall R$.Normaliser(D))

$C = D_1 \sqcap ... \sqcap D_n$

$\longrightarrow \bot$ si $\bot \in$ {Normaliser(D_1),...,Normaliser(D_n)}

si $\exists R.(\{D, \neg D\} \subseteq$ {Normaliser(D_1),...,Normaliser(D_n)})

autrement Encode(Normaliser(D_1) $\sqcap$ ...$\sqcap$Normaliser(D_n))

$C = \exists R.D$ $\longrightarrow$ Normaliser($\neg \forall R.\neg D$)

$C = D_1 \sqcup ... \sqcup D_n$

$\longrightarrow$ Normaliser($\neg(\neg D_1 \sqcup ... \sqcup \neg D_n)$)

Encoder(C) :

$C = \forall R.D$ $\longrightarrow CN$ si $CN \equiv \forall R.D \in \mathcal{T}$

autrement CN' où CN' est un nom de concept nouveau et

$$\mathcal{T} \longrightarrow \mathcal{T} \cup \{CN' \equiv \forall R.D\}$$

$C = D_1 \sqcap ... \sqcap D_n$

$\longrightarrow CN$ si $CN \equiv (D'_1 \sqcap ... \sqcap D'_n) \in \mathcal{T}$ et

$$\forall D.(D \in \{D_1, ..., D_n\} \Leftrightarrow D \in \{D'_1, ..., D'_n\})$$

autrement CN' où CN' est un nom de concept nouveau et

$$\mathcal{T} \longrightarrow \mathcal{T} \cup \{CN' \equiv D_1 \sqcap ... \sqcap D_n\}$$

Tab. 3.1 : Normalisation et Encodage

3.1.2.3 Absorption de ICG

Nous rappelons qu'un axiome dans la forme $A \equiv D$ est appelé introduction de concept primitif, et $C \sqsubseteq D$ est appelé inclusion de concept général (ICG), où A est un nom de concept, C et D sont des expressions de concept.

D'abord, nous présentons quelques notations, pour une ICG ($C \sqsubseteq D$), le côté gauche (C) s'appellera son *antécédent* et le côté droit (D) est son *conséquent*.

L'absorption vise à réduire le nombre d'ICGs en les absorbant autant que possible dans des axiomes d'introduction de concept primitif. Cela est fait habituellement en ré-écrivant les axiomes dans une forme équivalente convenable. Pour l'*absorption de concept*, l'axiome devrait être dans la forme $CN \sqsubseteq D$, où CN est un nom de concept primitif et D est une expression de concept arbitraire ; pour l'*absorption de rôle*, l'axiome devrait être dans la forme $\exists R.\top \sqsubseteq D$, où D est une expression de concept arbitraire.

Cette technique est suggérée par la structure d'ICGs dans la terminologie GALEN [Hor97] et par des restrictions sur la structure d'ICGs imposée par la syntaxe du langage de description de concept GRAIL [Hor97]. Une ICG est *absorbable* si sa structure appartient à une des formes suivantes :

- ICG dont l'antécédent est seulement un nom de concept primitif.
  $CN \sqsubseteq C$ et $CN \sqsubseteq D$ sont absorbés en : $CN \sqsubseteq C \sqcap D$
  La correspondance de la sémantique :
  $CN^{\mathcal{I}} \subseteq C^{\mathcal{I}} \wedge CN^{\mathcal{I}} \subseteq D^{\mathcal{I}} \Longleftrightarrow CN^{\mathcal{I}} \subseteq C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- ICG dont l'antécédent est soit une expression conjonctive de concept soit un nom de concept non-primitif dont la définition est une expression conjonctive de concept. La première proposition conjointe d'une expression conjonctive de concept est toujours soit un nom de concept primitif soit un nom de concept non-primitif dont la définition est une expression conjonctive de concept.
  $CN \sqcap C \sqsubseteq D$ sont absorbés en : $CN \sqsubseteq D \sqcup \neg C$
  La correspondance de la sémantique :
  $CN^{\mathcal{I}} \cap C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \Longleftrightarrow CN^{\mathcal{I}} \subseteq D^{\mathcal{I}} \cup (\neg C)^{\mathcal{I}}$

En général, un ICG $C \sqsubseteq D$ est absorbé en un axiome d'introduction de concept primitif, toutes les fois que possible, en utilisant les étapes suivantes :

1. $\mathbf{G} := \{D, \neg C\}$
2. S'il y a un nom de concept primitif négatif $CN \in \mathbf{G}$ avec un axiome de la forme $CN \sqsubseteq E$, on absorbe l'ICG en l'introduction de CN :
$$CN \sqsubseteq E \sqcap \left(\bigcup_{K \in \mathbf{G} - \{CN\}}\right)$$
3. S'il y a une expression de concept conjonctive négative $E \equiv \neg(C_1 \sqcap ... \sqcap C_n) \in \mathbf{G}$ ou un

nom de concept non primitif dont la définition est une expression de concept conjonctive $C_1 \sqcap ... \sqcap C_n$, alors :

$$\mathbf{G} \longrightarrow (\mathbf{G} - \{E\}) \cup \{\neg C_1, ..., \neg C_n\}$$

et retourne à l'étape 2.

4. S'il y a une expression de concept disjonctive $E \equiv (C_1 \sqcup ... \sqcup C_n) \in \mathbf{G}$ ou un nom de concept non primitif dont définition est une expression de concept disjonctive $C_1 \sqcup ... \sqcup C_n$, alors :

$$\mathbf{G} \longrightarrow (\mathbf{G} - \{E\}) \cup \{C_1, ..., C_n\}$$

et retourne à l'étape 2.

5. Pour que l'ICG ne soit pas absorbé, il doit être ajouté à la meta-contrainte $\mathcal{M}$:

$$\mathcal{M} \longrightarrow \mathcal{M} \sqcap (D \sqcup \neg C)$$

et traité en utilisant l'internalisation.

Exemple 3.5 :

Étant donné un ICG :
ETUDIANT $\sqcap \exists$ avoirPub.THESE $\sqsubseteq \exists$ travailler.ENTREPRISE
peut être absorbé en :
ETUDIANT $\sqsubseteq \neg \exists$avoirPub.THESE $\sqcup \exists$ travailler.ENTREPRISE

L'absorption permet d'éviter l'addition d'un grand nombre de disjonctions non pertinentes à l'étiquette de chaque noeud dans un arbre **T** construit par l'algorithme d'expansion de tableaux. Par exemple, l'internalisation d'ICG :

$$\text{ETUDIANT} \sqcap \exists \text{ avoirPub.THESE} \sqsubseteq \exists \text{ travailler.ENTREPRISE}$$

résulterait en la disjonction :

$$\neg(\text{ETUDIANT} \sqcap \exists \text{ avoirPub.THESE}) \sqcup \exists \text{ travailler.ENTREPRISE}$$
$$\Longleftrightarrow \neg \text{ ETUDIANT} \sqcup \forall \text{ avoirPub.} \neg \text{THESE} \sqcup \exists \text{travailler.ENTREPRISE}$$

cette disjonction est ajoutée à l'étiquette de chaque noeud dans **T**. Cependant que l'absorption d'ICG dans l'axiome d'introduction d'ETUDIANT résulterait en la disjonction :

$$\neg \exists \text{ avoirPub.THESE} \sqcup \exists \text{ travailler.ENTREPRISE}$$

qui est ajoutée seulement à un noeud x quand ETUDIANT $\in \mathcal{L}(x)$ - cela se passera automatiquement quand ETUDIANT est déployé.

3.2 Discussion et conclusion

Dans ce chapitre, nous avons présenté quelques techniques d'optimisation qui se sont montrées réellement efficaces [Hor97] dans des ontologies réalistes. Le déploiement paresseux s'est distingué parce qu'il n'est pas limité par les concepts nommés, les contradictions peuvent également être détectées rapidement entre des expressions de concept encodées. Le déploiement et l'absorption se sont avérés très efficaces en réduisant la taille de l'espace de recherche. L'absorption peut être employée avec des LDs qui soutiennent la négation dans des descriptions de concept, et est indépendante de l'algorithme de vérification de subsomption. L'absorption, la normalisation et l'encodage peuvent être effectués comme un pas de pré-traitement. La normalisation et l'encodage effectuent efficacement toutes les comparaisons structurales avant de classifier et de mettre en cache les résultats dans la forme des introductions de nouveaux concepts non primitifs. Ces deux techniques peuvent être appliquées avec des LDs incluses dans des implantations existantes.

L'heuristique est utilisée pour minimiser la taille de l'arbre de recherche en maximisant l'effet de l'élagage (the pruning effect) ajouté à chaque point d'embranchement. L'embranchement sémantique, la PCB et la recherche heuristique peuvent être exécutés plus efficacement parce qu'un grand nombre de comparaisons des disjoncts exigés peuvent être effectuées à un niveau de nom de concept au lieu d'un niveau structurel. La recherche d'embranchement sémantique et la PCB peuvent être employées avec n'importe quel algorithme de tableaux, bien qu'ils demandent un changement considérable dans l'implémentation des algorithmes. Pour la technique de mise en cache, quand une terminologie est classifiée, les modèles de chaque expression et sous-expression de concept dans la terminologie peuvent être mis en cache par la suite.

Toutefois, le comportement des techniques d'optimisation n'est pas orthogonal, il peut parfois interférer avec d'autres. Concrètement, l'heuristique MOMS minimise l'effet de PCB. L'emploi de l'heuristique MOMS pour choisir les disjoncts branchés dégrade la performance de backjumping [Hor97]. Plusieurs optimisations augmentent des conditions de stockage. Le backjumping, en particulier, exige un ensemble de dépendances pour être stocké avec chaque expression de concept dans une étiquette de noeud. Cependant, comme les mêmes ensembles de dépendance seront employés entièrement ou partiellement par des grands nombres d'expressions de concept, la condition de stockage peut être réduite en employant un pointeur basé sur l'exécution telle que la liste de LISP [Hor97].

Les techniques d'optimisation vues dans ce chapitre visent à réduire le nombre d'ICGs et l'espace de recherche. Néanmoins, elles sont seulement efficaces dans certaines terminologies concrètes. Dans le chapitre suivant, nous allons proposer une nouvelle technique pour optimiser la performance du raisonnement en décomposant une terminologie en plusieurs sous terminologies. Cette méthode permet aussi de diminuer le nombre d'ICGs, et elle peut obtenir un certain

effet dans toutes les terminologies réalistes.

Deuxième partie

Théorie de décomposition de l’ontologie

Cette partie présente les contributions à la construction d'une approche de décomposition de l'ontologie et aux développements algorithmiques pour le raisonnement sur l'ontologie décomposée. Dans notre approche qui s'appelle "décomposition overlay", l'exploitation de ses propriétés nous mènera à une première conclusion importante : ***la préservation de la sémantique de l'ontologie originelle****. Une seconde conclusion importante concernant cette décomposition :* ***la préservation d'inférence de l'ontologie originelle*** *est également démontrée via les deux algorithmes de raisonnement parallèle et distribué proposés sur l'ontologie décomposée.*

Chapitre 4

Théorie de décomposition

Ce chapitre commence par une vue générale du problème de décomposition. L'application de décomposition dans la plupart des domaines vise à l'objectif d'*optimisation*. Par la suite, nous introduisons successivement la notion de décomposition dans la théorie logique et dans les bases de données qui est la prémisse pour l'application dans les bases de connaissances. Nous proposons deux approches de décomposition pour la terminologie (TBox) s'appelant *décomposition intra-TBox* et *décomposition inter-TBox*. Nous nous intéressons à la décomposition inter-TBox. Comme il est nécessaire d'avoir un formalisme pour représenter le résultat de cette décomposition, dans la dernière section, quelques *formalismes hybrides* de représentation d'un système d'intégration de plusieurs ontologies sont décrits.

4.1 Problème de décomposition

Une démarche usuelle pour résoudre un problème consiste à s'intéresser à des cas particuliers plus simples, pour ensuite ramener le problème général à ces cas particuliers. Pour cette démarche, on pense souvent à la décomposition du problème général en des problèmes plus simples. Par conséquent, la décomposition est largement utilisée dans différents domaines comme la biologie, la chimie, la physique, les mathématiques, le traitement du signal, voire la grammaire et la linguistique,... En informatique, la décomposition se rapporte au processus par lequel un problème ou un système complexe est décomposé en différentes parties qui sont plus faciles à concevoir, comprendre, programmer, et maintenir. Par exemple, dans la programmation structurée, la décomposition algorithmique casse un processus en des étapes bien définies. La décomposition orientée-objet, d'autre part, casse un grand système en des classes plus petites où les objets sont responsables d'une certaine partie du domaine du problème. La plupart des décompositions de paradigmes suggèrent de décomposer un programme en différentes parties afin de réduire au minimum les dépendances statiques parmi ces parties, et maximiser leur co-

hésion. Habituellement l'utilisation d'un paradigme de décomposition permet d'optimiser une certaine métrique concernant la complexité du programme.

À partir de cette idée, la décomposition a été appliquée dans les théories de la logique du premier-ordre et de la logique propositionnelle en divisant une théorie en des partitions et en raisonnant sur ces partitions. Dans la théorie des bases de données relationnelles, on a étudié plusieurs algorithmes pour décomposer un schéma relationnel quelconque en des schémas qui sont dans une certaine forme normale. Notre but dans ce mémoire ne dépasse pas cette idée. Nous proposons une décomposition de l'ontologie visant deux objectifs. Premièrement, l'*objectif d'optimisation* : l'étude de l'impact de la décomposition sur le raisonnement, en particulier, les algorithmes du raisonnement seront réalisés sur des ontologies décomposées au lieu d'être exécutés sur une ontologie originelle. L'étude détaillée sera présentée dans les deux chapitres suivants. Le deuxième est l'*objectif conceptuel* qui propose une méthodologie de conception des ontologies. Pour cette étape de conception, les ontologies sont représentées par des graphes dont l'idée part des avantages inhérents à la décomposition dans la théorie des graphes. Cela sera étudié plus en détail dans les chapitres 7 et 8.

4.2 Décomposition de théorie logique

Dans la théorie logique, on a réalisé la décomposition d'une théorie logique en des *sous-théories* pour réduire le temps et l'espace de calcul dans le raisonnement. En particulier, Amir et. al. [AM05] ont fourni une méthode de décomposition d'une théorie logique du premier-ordre ou propositionelle en des petites partitions, et des algorithmes de transmission de message en avant et en arrière (forward and backward message-passing algorithms) pour le raisonnement sur des partitions.

Une théorie A est décomposée en $\{A_i\}_{i\leq n}$, avec $A = \bigcup_i A_i$. Chaque A_i est appelé une *partition*, $L(A_i)$ est sa signature (l'ensemble de symboles non-logiques) et $\mathcal{L}(A_i)$ est son langage (l'ensemble de formules établies avec $L(A_i)$). Chaque décomposition définit un graphe étiqueté $G = (V, E, l)$ qui est appelé le *graphe d'intersection*. Dans le graphe d'intersection, chaque noeud i représente une partition individuelle A_i, ($V = \{1, ..., n\}$), deux noeuds i, j sont liés par un arc si $L(A_i)$ et $L(A_j)$ ont un symbole commun ($E = \{(i, j)|L(A_i) \cap L(A_j) \neq \emptyset\}$), et les arcs sont étiquetés par l'ensemble de symboles partagés entre les partitions associées ($l(i, j) = L(A_i) \cap L(A_j)$)). On se réfère à $l(i, j)$ comme le langage de communication entre les partitions A_i et A_j. On suppose que le graphe d'intersection est connecté en ajoutant un nombre minimal d'arcs à E avec des étiquettes vides, $l(i, j) = \emptyset$.

Pour illustrer cette décomposition, nous reprenons un exemple dans [AM05]. Une théorie propositionnelle simple A est décrite dans la table 4.1 (ceci est une forme propositionnelle de

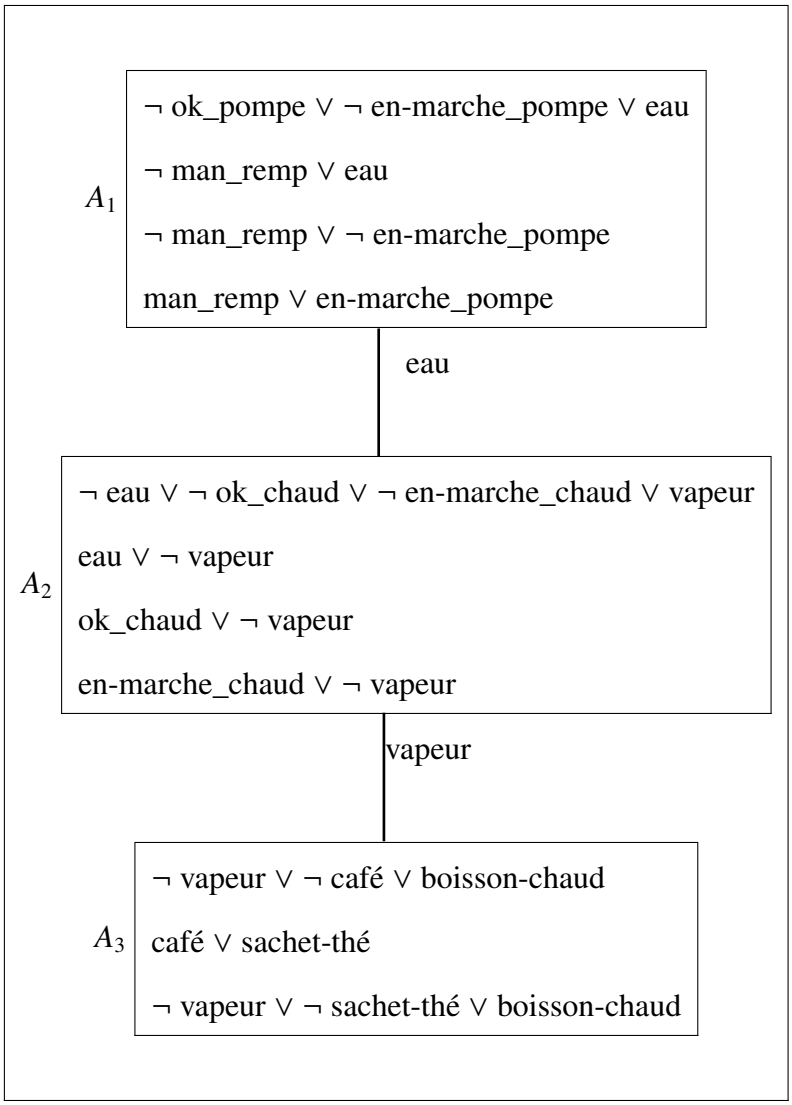
ok_pompe ∧ en-marche_pompe ⇒ eau	man_remp ⇒ eau
man_remp ⇒ ¬ en-marche_pompe	¬ man_remp ⇒ en-marche_pompe
eau ∧ ok_chaud ∧ en-marche_chaud ⇒ vapeur	¬ eau ⇒ ¬ vapeur
¬ ok_chaud ⇒ ¬ vapeur	¬ en-marche_chaud ⇒ ¬ vapeur
vapeur ∧ café ⇒ boisson-chaud	café ∨ sachet-thé
vapeur ∧ sachet-thé ⇒ boisson-chaud	

Tab. 4.1 : Axiomatisation d'une machine d'espresso

la théorie présentée par une implication matérielle). Les axiomes capturent le fonctionnement des aspects d'une machine espresso. Les quatre premiers axiomes dénotent que si la pompe de machine est OK et la pompe est en marche, alors la machine a une réserve d'eau. La machine peut être remplie manuellement, mais la machine ne peut jamais être remplie manuellement tandis que la pompe est en marche. Les quatre suivants axiomes dénotent qu'il y a de la vapeur si et seulement si la chaudière est OK et est en marche, et qu'il y a une réserve d'eau. Les trois derniers axiomes dénotent qu'il y a toujours soit du café soit du thé, et que la vapeur associée au café (ou au thé) donnent lieu à une boisson chaude.

La table 4.2 illustre une décomposition de A en trois partitions A_1, A_2 et A_3, et son graphe d'intersection (notons que les axiomes dans cette table ont été transformés en des phrases (des propositions logiques). Le raisonnement sur des partitions d'axiomes logiques est effectué par les algorithmes de transmission de message en avant et en arrière. Pour le premier algorithme, quand une requête Q est posée sur A_k, on doit déterminer la direction dans laquelle ce message devrait être envoyé dans le graphe d'intersection en calculant un ordre partiel strict sur des noeuds dans le graphe qui utilise la décomposition ainsi que la requête Q. Pour le deuxième algorithme, à partir de Q, afin d'affirmer que $\neg Q$ est dans A_k, on choisit une partition A_j qui est la plus éloignée de A_k dans G (où la distance entre deux noeuds dans le graphe G est le nombre des noeuds comportant le chemin le plus court (the shortest path) entre les deux noeuds), et essaie de prouver {} dans A_j.

Ces algorithmes exigent que la requête Q soit dans le langage d'une seule partition $\mathcal{L}(A_k)$, $k \leq n$. Pour une requête Q qui se compose des symboles dans de multiples partitions, on doit ajouter une nouvelle partition A_Q, avec le langage de requête $\mathcal{L}(A_Q) = \mathcal{L}(Q)$. A_Q peut contenir $\neg Q$ ou aucun axiome. On peut également décomposer la requête dans les partitions appropriées, toutefois ce problème n'est pas simple, on ne peut que réaliser le cas simple de requête propositionnelle comme introduit dans [AM05].



TAB. 4.2 : Une décomposition de A et son graphe d'intersection

4.3 Décomposition du schéma relationnel des bases de données

Dans cette section, nous ne rappelons pas les notions dans la théorie des bases de données, et nous supposons que ces notions sont familières aux lecteurs. Nous voulons simplement aborder le problème de la décomposition qui a été développé depuis longtemps dans le domaine des bases de données. La structure d'une base de données relationnelle peut être définie par un schéma relationnel universel et par un ensemble de dépendances fonctionnelles. Les relations qui se conforment au schéma universel peuvent contenir beaucoup d'informations superflues et avoir les propriétés de mise à jour indésirables. Par exemple, on considère un schéma relationnel :

$$R = \{\text{Employé, Responsable, Département, Projet}\}$$

avec les dépendances fonctionnelles :

$$\{\text{Employé} \rightarrow \text{Département, Département} \rightarrow \text{Responsable }\}$$

Dans ce schéma, quelques anomalies apparaîtront lors d'opérations de mise à jour :

- Si un département d change son responsable, on doit changer le responsable dans tous les tuples où d apparaît.
- On ne peut que présenter un employé quand le département dans lequel un responsable travaille. Cette anomalie résulte de la dépendance transitive.
- Si on supprime un employé qui travaille dans un projet simple, on perd la relation projet-département,...

De plus, la théorie de conception du schéma des bases de données vise à caractériser des "bons" schémas, et à inventer des algorithmes pour tester et améliorer la qualité d'un schéma donné. Les dépendances fonctionnelles jouent un rôle essentiel pour ces deux charges. En tant qu'heuristique générale, un schéma est considéré comme "bon" s'il se compose de composants presque atomiques et presque indépendants où l'atomicité et l'indépendance sont exprimées relativement aux caractéristiques du modèle de données.

Pour ces raisons, on a proposé la décomposition comme une solution. L'idée principale de décomposition d'un schéma relationnel est de diviser un schéma d'origine disparate en des schémas dans quelques formes normales selon des dépendances fonctionnelles, et pour réduire la redondance de l'information, et pour réduire les anomalies par rapport à des opérations de mise à jour.

Soit un *schéma universel* R, R est décomposé en des sous-schémas $R_1, ..., R_n$ avec $R = \bigcup_{i=\overline{1,n}} R_i$. La collection de $\{R_1, ..., R_n\}$ est appelée un *schéma de base de données*. Donc les

relations universelles peuvent être présentées en tant que leurs projections sur les schémas relationnels dans le schéma de base de données. [GR93] a proposé quelques propriétés pour une "bonne" décomposition :

- *Redondance minimale* : devrait représenter les objets significatifs de sorte que, par exemple, ils puissent être mise à jour sans connaître les valeurs pour tous les attributs dans les schémas universels. Ceci est habituellement utilisé pour signifier que les schémas relationnels devraient être dans une des différentes formes normales.
- *Représentation* : il devrait être possible de récupérer la relation universelle de relations composantes, i.e., elle devrait avoir un joint "lossless".
- *Séparation* : Quand une relation est mise à jour, il devrait être possible de s'assurer que les dépendances dans la relation universelle tiennent toujours après la mise à jour sans construire réellement la relation universelle. Cette condition est satisfaite si la décomposition préserve la dépendance, i.e., si les dépendances projetées impliquent l'ensemble original de dépendances.

On a déjà proposé plusieurs algorithmes de décomposition. Certains fournissent une décomposition "lossless" selon des dépendances, tandis que d'autres fournissent une décomposition "lossless" selon le contenu des bases de données. Concrètement, dans [CY92], les auteurs ont étudié une méthode de décomposition d'une relation multi-niveaux dans une collection de relations 3NF et 4NF d'après des dépendances fonctionnelles diverses.

Nous reprenons l'exemple ci-dessus. Cette relation n'est pas 3NF ni 2NF, parce que (Employé, Projet) est une clef primaire de R, mais Responsable n'est pas entièrement indépendant sur (Employé, Projet). Les anomalies dans R disparaissent quand on décompose la relation ci-dessus en des relations 3NF suivantes :

$$R_1 = \{\text{Employé, Projet}\}$$
$$R_2 = \{\text{Employé, Département}\}$$
$$R_1 = \{\text{Département, Responsable}\}$$

4.4 Décomposition de la base de connaissances

Nous considérons une base de connaissances qui est représentée dans la logique de description. L'idée de décomposition de la base de connaissances est proposée à partir de l'exigence d'optimisation du raisonnement par rapport aux grandes bases de connaissances. Notre problème de décomposition se concentre sur la composante terminologie (TBox) qui se compose d'un ensemble de concepts et de rôles, et un ensemble d'axiomes. Comme le traitement d'une BC joue un rôle important dans la détection et la déduction des connaissances implicites à partir

de connaissances explicites disponibles, notre but principal dans ce mémoire vise à préserver les services d'inférence en plus de la préservation de tous les objets de la BC originelle. L'impact de l'ensemble d'inclusions de concept général est une raison cruciale qui amène à l'explosion du temps et de l'espace dans le processus du raisonnement. Donc, plusieurs techniques d'optimisation ont pour but de minimiser ces axiomes (comme ceci a été présentés dans le chapitre 2). Par conséquent, nous souhaitons chercher une approche de décomposition de l'ensemble d'axiomes qui permet d'*améliorer la performance de l'algorithme de raisonnement et de préserver la sémantique et les résultats du raisonnement de l'ontologie originelle*. L'amélioration de la performance de l'algorithme vise à l'optimisation, tandis que la préservation des résultats du raisonnement garantit la correction de la méthode de décomposition.

Cependant, à partir d'une vue générale de décomposition, nous examinons deux approches de décomposition des axiomes (voir la figure 4.1) comme suit :

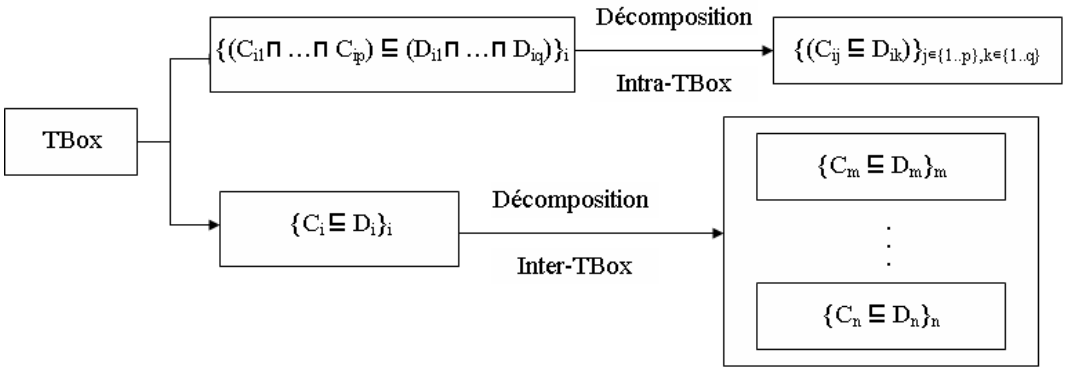


Fig. 4.1 : Décompositions de TBox

4.4.1 Décomposition intra-TBox

Avant de définir la décomposition intra-TBox, nous donnons une notion de *sous-axiome* comme suit :

Définition 4.1. *(sous-axiome) Soit un axiome A dans la forme* $(C_1 \sqcap ... \sqcap C_p) \sqsubseteq (D_1 \sqcap ... \sqcap D_q)$*, alors* $C_i \sqsubseteq D_j$ *est appelé un sous-axiome de A, avec* $i = \overline{1,p}$, $j = \overline{1,q}$.

Définition 4.2. *(Décomposition intra-TBox) Soit une TBox* $\mathcal{T}$ *avec l'ensemble d'axiomes dans la forme* $\{(C_{i1} \sqcap ... \sqcap C_{ip}) \sqsubseteq (D_{i1} \sqcap ... \sqcap D_{iq})\}$*, une décomposition intra-TBox (intra-ontologie) de*

$\mathcal{T}$ est une TBox avec l'ensemble d'axiomes $\{(C_{ij} \sqsubseteq D_{ik})\}_{j=\overline{1,p},k=\overline{1,q}}$, pour $i \in I$ et I est un ensemble d'index.

Avec cette définition, la décomposition *intra-ontologie* examine la structure d'axiomes, i.e., décomposer les formules qui représentent les axiomes en formules moins complexes. À partir du point de vue des réseaux sémantiques dans lequel la description de concept est considérée comme un graphe orienté comportant des noeuds et des arcs, le calcul des relations de subsomption est basé sur la structure (dans la section 2.2.1). Pour cette décomposition, les résultats du raisonnement dans la TBox originelle ne sont pas garantis dans la nouvelle TBox. Cette approche peut mener la correction des résultats du raisonnement si l'on peut appliquer l'algorithme de comparaison structurelle. Autrement dit, un axiome $(C_1 \sqcap ... \sqcap C_m) \sqsubseteq (D_1 \sqcap ... \sqcap D_n)$ sera décomposé en sous-axiomes $C_i \sqsubseteq D_j$, où $i = 1, ..., m$ et $j = 1, ..., n$, si seulement si C_i, D_j satisfont les conditions de subsomption structurelle comme dans la section 2.2.1. Donc le raisonnement est exécuté sur ces sous-axiomes en comparant la structure [PVG96], et il a été indiqué très efficace (polynômial).

Néanmoins, malheureusement l'algorithme de comparaison structurelle est incomplet et les langages présentés d'ontologie dans ce cas sont très limités. En particulier, pour seulement des langages très inexpressifs comme $\mathcal{FL}^-$ et avec les expressions comparables.

4.4.2 Décomposition inter-TBox

Définition 4.3. *(Décomposition inter-TBox) Soit une TBox $\mathcal{T}$ avec l'ensemble d'axiomes* **A** *(chaque axiome A_i est dans la forme $C_i \sqsubseteq D_i$), une décomposition inter-TBox (inter-ontologie) du $\mathcal{T}$ est un l'ensemble des TBox$\{\mathcal{T}_k\}$ avec des sous-ensembles d'axiomes $\mathbf{A}_k$ respectivement, où $\mathbf{A}_k \in \mathbf{A}$.*

La décomposition inter-TBox décompose une TBox en *sous-TBox* (voir la définition dans la section 5.2.1) en divisant uniquement l'ensemble d'axiomes en sous-ensembles d'axiomes. Les sous-TBox contiennent respectivement ces sous-ensembles d'axiomes et tous les concepts et rôles de la TBox donnée. Cette approche est signifiante dans la réalité pour des grandes ontologies concernant plusieurs domaines locaux. En effet, on peut diviser cette TBox en TBox locales (sous-TBox) qui correspondent à des domaines locaux, les exigences (requêtes) locales seront indépendamment traitées dans ces TBox locales. Ceci évite des traitements superflus dans la TBox originelle. Par exemple, la TBox $\mathcal{T}_{uni}$ dans la table 1.2 peut être décomposée en deux composants qui sont stockés dans deux TBox $\mathcal{T}_1$ et $\mathcal{T}_2$ comme dans la table 4.3. Dans

$\mathcal{T}_1$

ETUDIANT	$\sqsubseteq$	PERSONNER
ENSEIGNANT	$\sqsubseteq$	PERSONNER
THESARD	$\sqsubseteq$	ETUDIANT
DOCTEUR	$\sqsubseteq$	THESARD $\sqcap\exists$ avoirPub.THESE-DOCTORAL
PROFESSEUR	$\sqsubseteq$	DOCTEUR $\sqcap\exists$ travailler.UNIVERSITE

$\mathcal{T}_2$

THESE	$\sqsubseteq$	PUBLICATION
ARTICLE	$\sqsubseteq$	PUBLICATION
LIVRE	$\sqsubseteq$	PUBLICATION
THESE-MASTER	$\sqsubseteq$	THESE
THESE-DOCTORAL	$\sqsubseteq$	THESE

TAB. 4.3 : Une décomposition inter-TBox de $\mathcal{T}_{uni}$ en deux TBox $\mathcal{T}_1$ et $\mathcal{T}_2$

cette décomposition, la TBox $\mathcal{T}_1$ contient les informations de personnes dans l'université, tandis que la TBox $\mathcal{T}_2$ contient les publications d'une bibliothèque. Les demandes séparées de personnel sont réalisées sur la TBox $\mathcal{T}_1$, et les demandes séparées de publication sont réalisées sur la $\mathcal{T}_2$. Néanmoins, la signification de décomposition sera beaucoup réduite si elle ne peut que traiter avec les requêtes locales, parce que les domaines locaux ont un rapport très intime entre eux. Par exemple, dans l'exemple ci-dessus, les personnes sont dans la relation "avoirPub" avec les publications. En particulier, le concept THESARD est dans la relation "avoirPub" avec le concept THESE-DOCTORALE, et le concept DOCTEUR est défini à partir de cette relation :"DOCTEUR $\sqsubseteq$ THESARD $\sqcap\exists$ avoirPub.THESE-DOCTORALE". Donc, les requêtes qui ont besoin d'informations dans toutes les TBox locales ne peuvent pas être réalisées sur une ontologie locale. Autrement dit, nous perdrons la plupart des inférences de la TBox originelle.

Comme nous avons dit dans la première partie du chapitre, notre but vise non seulement à la préservation de tous les axiomes avec leur sémantique mais encore avec les résultats du raisonnement de la base de connaissances originelle. Donc, nous avons besoin d'avoir une manière de présentation de ce système qui peut également bien exprimer les relations entre les sous-ontologies. Au lieu de raisonner sur une ontologie donnée, les algorithmes de tableaux sont mis en application en parallèle sur ces sous-ontologies produites de l'ontologie originelle. Plusieurs

auteurs ont proposé des méthodes différentes pour combiner les formules de la représentation et du raisonnement de connaissances comme Oliver Kutz et al. avec "$\mathcal{E}$-connections" [GPS04b, GPS04a], Logique de description basée sur package [BCH06a, BCH06b], logique de description distribuée [ST04a, ST04b, SBT05].

D'autre part, l'objectif d'optimisation du raisonnement nous conduit à l'idée de recherche d'une *bonne décomposition* d'une BC. L'analyse des paramètres calculés des algorithmes de raisonnement suggère des critères de décomposition. Par exemple, dans [AM05], les auteurs ont donné les critères : le nombre de symboles non-logiques inclus dans la communication entre les sous-ensembles, la taille de chaque sous-ensemble et la topologie du graphe de décomposition. Une décomposition est réellement efficace quand ces paramètres sont optimisés. Ces critères sont les mêmes dans notre décomposition. Le cas le plus idéal est que les sous-ensembles d'axiomes sont incohérents, i.e., il n'y a pas de partie de communication entre eux. Pourtant, dans la réalité, les objets de l'ontologie sont cohérents, donc on doit chercher une méthode pour minimiser ce paramètre.

Les travaux dans ce mémoire ne considèrent que la méthode inter-ontologie, donc nous utiliserons seulement le terme "décomposition" dans la suite.

4.5 Formalismes hybrides de représentation de connaissances

La combinaison récente des logiques de description avec d'autres formalismes logiques inclut [KLWZ03] :

- La combinaison des LDs avec des logiques temporelles pour former des *LDs temporelles multidimensionnelles* [AFWZ02].
- La *fusion* de multiple LDs dans un seul formalisme qui hérite de la décidabilité de ses composantes [BLSW00]. Le but de la fusion est l'extension de l'expressivité de LDs qui garantit la décidabilité du raisonnement.
- La combinaison des LDs différentes dans le contexte des systèmes d'information fédérée relâchée a résulté en *logiques de description distribuées* [BS03].

Néanmoins, dans le contexte de décomposition, les sous-ontologies sont représentées dans le même langage logique que l'ontologie originelle. De plus, nous n'examinons que les ontologies représentées dans des logiques de description. Donc, nous introduirons les techniques hybrides de représentation des connaissances suivantes :

- *$\mathcal{E}$-connections* : $\mathcal{E}$-connections est une méthode de combinaison pour la représentation de connaissances et les formalismes du raisonnement. Cette méthode est employée dans le cas où les domaines sont complètement disjoints. Elle a défini et étudié des *systèmes de description abstraits* (SDA) qui se composent des logiques de description standards, des

logiques modales, et des logiques épistémiques etc.. Le résultat théorique principal est que chaque $\mathcal{E}$-connection d'un nombre fini de SDAs décidables est aussi décidable [KWZ02]. $\mathcal{E}$-connection divise des rôles en ensembles disjoints de rôles locaux (reliant des concepts dans une sous-ontologie) et en liens (reliant les concepts dans des sous-ontologies différentes). Par exemple, deux ontologies "Personne" (O_1) et "Pays" (O_2) peuvent être reliées par quelques liens "aime", "habite-en", "citoyen-de" et O_1 peut utiliser un tel lien pour construire des concepts locaux comme (Figure 4.2) :

$$Francophile \sqsubseteq \langle aime \rangle^1 France$$

où l'opérateur $\langle r \rangle^1 C$ est un des opérateurs de connexion pour parler de relations de lien. Intuitivement, $\langle r \rangle^1 C$ peut être considéré comme une restriction de valeur existentielle. le symbole $.^1$ indique que cet opérateur est appliqué à un concept de l'ontologie O_2. On peut utiliser des relations de lien dans l'autre direction :

$$France \sqsubseteq \langle habite - en \rangle^2 (Humain \sqcap \neg \langle citoyen - de \rangle^1 France)$$

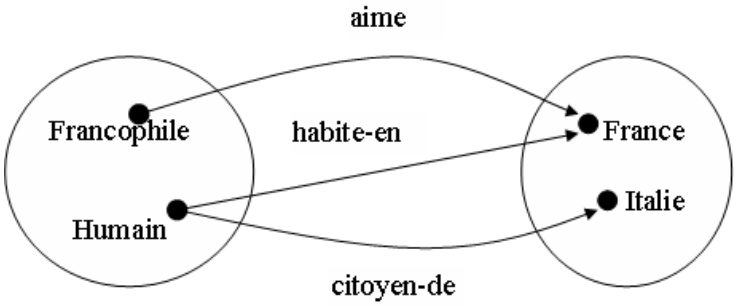


Fig. 4.2 : Connexion de deux ontologies

Le langage de $\mathcal{E}$-connections est l'union de langages originaux qui est enrichi par des opérateurs capables d'exprimer des relations de lien. Pourtant il manque la relation de subsomption inter-ontologies. La procédure du raisonnement basée sur le tableau, réalisée dans le raisonneur Pellet, génère un ensemble de tableaux (arbres) reliés par des instances de $\mathcal{E}$-connections (des instances du rôle qui traversent des sous-ontologies).

- *Logique de description distribuée* (LDD) : Aussi bien que $\mathcal{E}$-connections, LDD [BS03, ST04a], est proposée par Serafini et. al., est également utilisée avec les domaines disjoints, mais ces domaines peuvent avoir une partie de chevauchement. On peut examiner quelques exemples dans [BS03] :

  Exemple 4.1 :

  Supposons que l'ontologie O_1 contient l'information de "couples", l'ontologie O_2 contient l'information de "personnes". Il y a des relations claires entre l'information dans les deux ontologies, par exemple, si un couple a une adresse associée, alors le mari et l'épouse qui composent le couple ont la même adresse, et cette relation doit être *dirigée*.

Dans ce cas, O_1 contient l'information d'individus qui sont des *abstractions* sur des individus dans O_2. On doit exprimer une *relation entre les individus* dans deux domaines, par exemple, entre chaque **couple12** dans O_1 et **Frank** et **Marie** dans O_2.

Exemple 4.2 :

Supposons que C_1, D et C_2 sont trois cours de plus en plus difficiles sur des matières relatives. L'université U_1 offre C_1, C_2 tandis que l'université U_2 offre D. Les universités permettent de substituer un cours x par un autre y, même comme un transfert, si x est plus dur que y, et x recouvre la majeure partie du matériel de y (disons 80%). U_1 peut décider de traiter D en tant qu'équivalent à C_1 ;d'une part, selon U_2, D est seulement équivalent au C_2, puisque C_1 est plus faible que D. Si des cours sont vus comme des collections d'étudiants qui les suivent, ceci peut être reformulé en termes de subsomption de classe : selon U_1, la classe C_1 subsume D, tandis que selon U_2, la classe D subsume C_2 ; mais U_1 ne pourrait pas voir le C_2 comme une sous-classe de C_1, puisque ceux-ci pourraient être en désaccord sur plus de 80% du matériel.

Cet exemple montre que des relations plus générales entre des ontologies, comme la subsomption, devraient être orientées. LDD est inspirée par la Logique du premier-ordre distribuée [GS98]. Les relations entre les ontologies sont présentées par les *règles de pont* qui visent à résoudre la correspondance d'individus entre deux ontologies, et les règles de pont seront concernées par cet aspect. Une introduction de LDD en détail sera montrée dans la Section 5.1 du Chapitre 5. L'algorithme du raisonnement est réalisé sur plusieurs ontologies locales. L'idée principale de cet algorithme est d'impliquer la subsomption de concept dans une ontologie à partir des subsomptions dans les autres ontologies et des règles de pont inter-ontologies qui relient des concepts d'une ontologie aux concepts d'une autre ontologie.

- *Logique de description basée sur le package* : Bao et. al. [BCH06a, BCH06b] ont défini une logique de description basée sur le package, dans laquelle une ontologie se compose d'une collection de modules appelés des packages. Chaque terme (nom de concept, propriété ou individu ou axiome) est associé avec un *home package*. Un package peut utiliser des termes définis dans les autres packages. Les packages sont reliés par les relations *importées*. Si un terme t a un "home package" Q, et s'il apparaît également dans un package P qui est différent de Q, alors t est appelé un terme étranger dans P. On dit que P importe $Q : t$, dénoté par $Q \xrightarrow{t} P$. Si un terme défini dans Q est importé dans P, nous disons que P importe Q, dénoté par $Q \rightarrow P$.

  Par exemple, on reprend deux packages "Personne" et "Pays", $1 : Francophile \sqsubseteq \exists 2 : aime.(2 : France)$. Dans ce cas, on peut avoir des relations de subsomption de concept inter-ontologies et des relations de rôle. Un algorithme de raisonnement basé sur un tableau distribué a été présenté pour l'extension de langages de LDs (il s'appelle P-LD) basé sur des packages. Cet algorithme adopte une approche fédérée au raisonnement en utili-

sant le stockage distribué d'un tableau global. Certains tableaux locaux peuvent partager quelques noeuds et communiquer en envoyant des messages entre eux. Il évite strictement de combiner les ontologies locales dans un espace de mémoire centralisé en utilisant le raisonnement distribué avec la sémantique localisée de P-LD, donc il permet de raisonner indépendamment sur des ontologies locales. Il supporte également le raisonnement avec des relations de subsomption inter-ontologies et de rôle inter-ontologies.

D'après nous, un choix convenable est la *logique de description distribuée* qui sera introduite dans le chapitre suivant. La discussion de ce choix sera faite dans la Section 5.3 du Chapitre 5 et dans la Section 6.5.1 du chapitre 6. Alors, une ontologie présentée dans la LD sera décomposée en plusieurs ontologies présentées dans la LDD, voir la Figure 4.3.

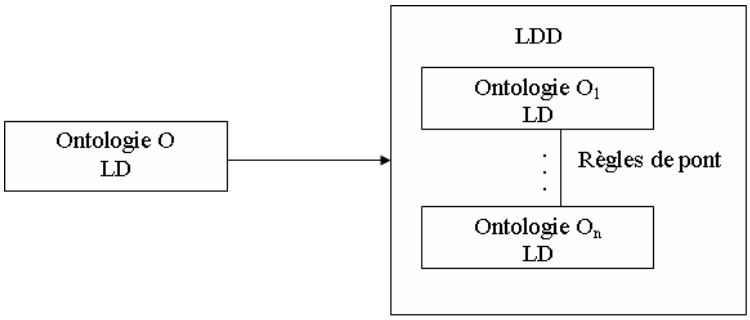


FIG. 4.3 : Représentation d'une décomposition de l'ontologie par la LDD

4.6 Conclusion

Ce chapitre a débuté par l'énonciation du problème de décomposition général. Ensuite, nous avons introduit l'application de décomposition dans la théorie logique, dans le schéma relationnel des bases de données qui est la prémisse pour notre décomposition avec la base de connaissances dans la logique de description. Pour l'ontologie des LDs, nous proposons deux visions de décomposition qui sont la décomposition intra-TBox et la décomposition inter-TBox. La décomposition intra-TBox est très difficile à réaliser parce qu'elle requiert l'intervention profonde de la structure d'axiomes. Et la préservation de sémantique des axiomes ne peut pas être réalisée. Nous choisissons la décomposition inter-TBox pour le but principal de préservation de sémantique et des services d'inférence de l'ontologie originelle.

Pour cela, nous présentons également des formalismes hybrides pour représenter le systèmes des ontologies locales avec leurs avantages et insuffisances, comme "$\mathcal{E}$-connections", la logique de description basée sur package et la logique de description distribuée. Enfin, nous proposons

une sélection de la logique de description distribuée pour notre décomposition, et le résultat de cette sélection sera introduit dans les deux chapitres suivants.

Chapitre 5

Logique de description distribuée et décomposition de l'ontologie

Les techniques d'optimisation présentées au chapitre précédent sont réellement efficaces dans certains cas avec des structures particulières des ICGs. Dans ce chapitre, notre travail consiste à représenter une ontologie générale (une TBox) par un ensemble des *sous-ontologies (sous-TBox)* dont chacune contient un nombre d'ICGs plus réduit que celui de l'ontologie initiale. Le raisonnement est mis en application sur le système de ces sous-TBox qui est représenté par une TBox distribuée. L'effet de raisonnement dépend de la méthode de décomposition de la TBox originelle, et nous proposons une technique appelée "la décomposition overlay". Le résultat de cette décomposition est une *TBox-décomposante*, est représenté dans la logique de description distribuée sous une forme spécifique par rapport à la *TBox distribuée* ordinaire. Pour des raisons de commodité, dans la première section de ce chapitre, nous rappelons quelques notions de la logique de description distribuée qui sont utilisées dans notre approche. Ensuite, nous présenterons les définitions et les propriétés de TBox-décomposante.

5.1 Logiques de description distribuées

Une logique de Description Distribuée (LDD) est proposée par Borgida et Serafini [BS03] pour représenter et raisonner au sujet des bases de connaissance (ontologies) dans les environnements distribués. Nous rappelons qu'une base de connaissances (BC) construite par un langage de concepts se compose généralement de deux niveaux de composantes : terminologique (TBox) et assertionnel (ABox). Dans une LDD, chaque ontologie correspond à une théorie de logique de description (TBox) et des ontologies sont liées par les mappings sémantiques. Quelques définitions de LDD seront brièvement résumées comme dans [BS03, ST04a].

5.1.1 Syntaxe

Soit $\{\mathcal{LD}_i\}_{i\in I}$, une collection de logiques de description, où I est un ensemble non vide des index. Pour chaque $i \in I$, une TBox $\mathcal{T}_i$ est présentée dans une $\mathcal{LD}_i$ concrète. Afin de distinguer des descriptions dans chaque TBox $\mathcal{T}_i$, nous mettons en tête des descriptions avec l'index de leurs TBox. Par exemple, $i : C$ dénote un concept C de $\{\mathcal{LD}_i\}_{i\in I}$. Les mappings sémantiques entre les différentes TBox sont présentés en utilisant des *règles de pont*.

Définition 5.1. *(Règle de pont) Une règle de pont de $\mathcal{T}_i$ à $\mathcal{T}_j$ est une expression d'une des trois formes suivantes :*

$$i : x \underset{\rightarrow}{\sqsubseteq} j : y, \qquad \textit{règle de pont "into"}$$

$$i : x \underset{\rightarrow}{\sqsupseteq} j : y, \qquad \textit{règle de pont "onto"}$$

$$i : x \underset{\rightarrow}{\equiv} j : x, \qquad \textit{règle de pont identique}$$

Où x et y sont soit deux concepts soit deux rôles, soit deux individus de $\mathcal{LD}_i$ et $\mathcal{LD}_j$ respectivement.

Les règles du pont de $\mathcal{T}_i$ à $\mathcal{T}_j$ représentent des relations entre $\mathcal{T}_i$ et $\mathcal{T}_j$ du point de vue de TBox j-ième. En particulier, la règle de pont into $i : A \underset{\rightarrow}{\sqsubseteq} j : G$ exprime que, du point de vue de $\mathcal{T}_j$, le concept A dans $\mathcal{T}_i$ est moins général que son concept local G. Pareillement avec la règle onto et la règle identique.

Définition 5.2. *(TBox distribuée) une TBox distribuée (TBD) $\mathfrak{T} = (\{\mathcal{T}_i\}_{i\in I}, \mathcal{B})$ se compose d'une collection des TBox $\{\mathcal{T}_i\}_{i\in I}$ et d'une collection des règles de pont $\mathcal{B} = \{\mathcal{B}_{ij}\}_{i\neq j\in I}$ entre elles.*

5.1.2 Sémantique

Chaque TBox $\mathcal{T}_i$ est interprétée par une interprétation locale $\mathcal{I}_i$ dans son domaine local. Les règles de pont sont interprétées en utilisant la *relation de domaine* r_{ij} entre les domaines.

Définition 5.3. *(Relation de domaine) Une relation de domaine r_{ij} de $\Delta^{\mathcal{I}_i}$ à $\Delta^{\mathcal{I}_j}$ est un sous-*

ensemble de $\Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$. *On note :*

$$\begin{aligned} r_{ij}(d) &= \{d' \in \Delta^{\mathcal{I}_j} | (d, d') \in r_{ij}\} \\ r_{ij}(D) &= \cup_{d \in D} r_{ij}(d) \\ r_{ij}(R) &= \cup_{(d,d') \in R} r_{ij}(d) * r_{ij}(d') \end{aligned}$$

avec $D \subseteq \Delta^{\mathcal{I}_i}$ *et* $r \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$.

La relation de domaine représente la possibilité de l'envoi des individus de $\Delta^{\mathcal{I}_i}$ à $\Delta^{\mathcal{I}_j}$ du point de vue de $\mathcal{LD}_j$.

Définition 5.4. *(Interprétation distribuée) Une interprétation distribuée* $\Im = (\{\mathcal{I}_i\}_{i \in I}, \{r_{ij}\}_{i \neq j \in I})$ *d'une TBD combine des interprétations locales* $\mathcal{I}_i$ *de chaque* $\mathcal{T}_i$ *sur le domaine local* $\Delta^{\mathcal{I}_i}$ *et une famille des relations* $r_{ij} \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_j}$ *entre des domaines locaux.*

Définition 5.5. *Une interprétation* $\Im = (\{\mathcal{I}_i\}_{i \in I}, \{r_{ij}\}_{i \neq j \in I})$ *satisfait les éléments d'une TBD* $\mathfrak{T} = (\{\mathcal{T}_i\}_{i \in I}, \mathcal{B})$ *si :*

$$\begin{array}{lll} \Im \models_d i : A \underset{\rightarrow}{\sqsubseteq} j : G & si & r_{ij}(A^{\mathcal{I}_i}) \subseteq G^{\mathcal{I}_j} \\ \Im \models_d i : B \underset{\rightarrow}{\sqsupseteq} j : H & si & r_{ij}(B^{\mathcal{I}_i}) \supseteq H^{\mathcal{I}_j} \\ \Im \models_d i : A \sqsubseteq B & si & \mathcal{I}_i \models A \sqsubseteq B \\ \Im \models_d \mathcal{T}_i & si & \mathcal{I}_i \models \mathcal{T}_i \\ \Im \models_d \mathfrak{T} & si & \Im \models_d \mathcal{T}_i \text{ et } \Im \models_d \mathcal{B}_{ij}, \forall i, j \in I \\ \mathfrak{T} \models_d i : A \sqsubseteq B & si & \forall \Im, \Im \models_d \mathfrak{T} \Longrightarrow \Im \models_d i : A \sqsubseteq B \end{array}$$

Définition 5.6. *(ABox Distribuée) Une ABox distribuée (ABD)* $\mathfrak{A} = \left\langle \{A_i\}_{i \neq j \in I}, \mathfrak{C} \right\rangle$ *se compose d'un ensemble d'ABoxes* $\{A_i\}_{i \in I}$, *et d'un ensemble* $\mathfrak{C} = C_{ij}{}_{i \neq j \in I}$ *de correspondances d'individus complétés, identiques et partiels de i à j. Pour chaque* $k \in I$, *toutes les descriptions dans* A_k *doivent être dans le langage correspondant* $\mathcal{DL}_k$, *et pour chaque rôle correspondant* $i : x \mapsto j : y$ *ou* $i : x \overset{\equiv}{\mapsto} j : x$ *ou* $i : x \overset{=}{\mapsto} j : \{y_1, y_2, ...\}$ *dans* C_{ij}, *le nom d'individu x doit être dans* $\mathcal{DL}_i, \mathcal{DL}_j$, *et* $y_1, y_2, ...$ *doivent être dans* $\mathcal{DL}_j$.

Une interprétation distribuée $\mathfrak{J}$ d-satisfait l'éléments d'un ABD $\mathfrak{A} = \langle \{A_i\}_{i\in I}, \mathfrak{C}\rangle$ selon les clauses suivantes : Pour chaque $i, j \in I$

$$
\begin{array}{rll}
\mathfrak{J} \models_d i : x \mapsto j : y, & \text{si} & y^{\mathcal{I}_j} \in r_{ij}(x^{\mathcal{I}_i}) \\
\mathfrak{J} \models_d i : x \overset{\equiv}{\mapsto} j : x, & \text{if} & r_{ij}(x^{\mathcal{I}_i}) = x^{\mathcal{I}_j} \\
\mathfrak{J} \models_d i : x \overset{=}{\mapsto} j : \{y_1, y_2, ...\}, & \text{si} & r_{ij}(x^{\mathcal{I}_i}) = \{y_1^{\mathcal{I}_j}, y_2^{\mathcal{I}_j}, ...\} \\
\mathfrak{J} \models_d i : C(a), & \text{si} & \mathcal{I}_i \models_d C(a) \\
\mathfrak{J} \models_d i : p(a,b), & \text{si} & \mathcal{I}_i \models_d p(a,b) \\
\mathfrak{J} \models_d A_i, & \text{ssi} & \mathfrak{J} \models_d \pi \text{ pour chaque assertion } \pi = C(a), p(a,b) \text{ dans } A_i \\
\mathfrak{J} \models_d \mathfrak{A}, & \text{si} & \text{pour } \forall i \in I,\ \mathfrak{J} \models_d A_i \text{ et} \\
 & & \mathfrak{J} \text{ d-satisfait chaque correspondance d'individu dans } \mathfrak{C}. \\
\mathfrak{A} \models_d i : C(a), & \text{si} & \text{pour chaque interprétation distribuée } \mathfrak{J}, \\
 & & \mathfrak{J} \models_d \mathfrak{A} \text{ implique } \mathfrak{J} \models_d i : C(a). \\
\mathfrak{A} \models_d i : p(a,b), & \text{si} & \text{pour chaque interprétation distribuée } \mathfrak{J}, \\
 & & \mathfrak{J} \models_d \mathfrak{A} \text{ implique } \mathfrak{J} \models_d i : p(a,b).
\end{array}
$$

5.2 Décomposition de l'ontologie

Nous rappelons qu'une base de connaissance (ontologie) établie par un langage de concept se compose généralement de deux niveaux composants : terminologique (appelé TBox) et assertionnel (appelé ABox). Nous nous intéressons au niveau terminologique et au problème de raisonnement dans une grande TBox. Une TBox est un ensemble fini des inclusions générales de concept (axiomes). Par conséquent la décomposition d'une ontologie est réduite à examiner l'ensemble des axiomes - nous coupons l'ensemble des axiomes en petits groupes selon les critères proposés dans [PLT06].

5.2.1 Définitions

Soient une TBox $\mathcal{T}$ avec $\mathbf{C}, \mathbf{R}, \mathbf{A}$ les ensembles de noms de concepts, de noms de rôles, et d'axiomes respectivement de $\mathcal{T}$. On note :

$$
\begin{aligned}
\mathcal{T} &= (\mathbf{C}, \mathbf{R}, \mathbf{A}), \\
\mathbf{A} &= \{(C \sqsubseteq D)\},
\end{aligned}
$$

où C et D sont des expressions de concepts.

Avant de donner une définition pour cette décomposition, nous définissons la notion de *sous-TBox* :

Définition 5.7. *(sous-TBox (ou sous-ontologie)) Une TBox* $\mathcal{T}' = (\mathbf{C}', \mathbf{R}', \mathbf{A}')$ *est dite une sous-Tbox de TBox* $\mathcal{T} = (\mathbf{C}, \mathbf{R}, \mathbf{A})$ *si et seulement si* $\mathbf{C}' \subseteq \mathbf{C}, \mathbf{R}' \subseteq \mathbf{R}$*, et* $\mathbf{A}' \subseteq \mathbf{A}$

Donc, les TBox $\mathcal{T} = (\mathbf{C}, \mathbf{R}, \mathbf{A})$ et $\mathcal{T}_\emptyset = (\emptyset, \emptyset, \emptyset)$ sont également des sous-TBox de $\mathcal{T}$.

Une *décomposition overlay* de TBox $\mathcal{T}$ est définie comme suit :

Définition 5.8. *(Décomposition overlay) Une décomposition overlay d'une TBox* $\mathcal{T}$ *est une TBox distribuée* $\langle\{\mathcal{T}_i\}, \{\mathcal{B}_{ij}\}_{i\neq j\in I}\rangle$ *(dénotée par* $\mathfrak{T}$ *or* $\mathcal{T}_{ij}$*) telle que :*

- *Chaque TBox* $\mathcal{T}_i$ *est une sous-Tbox de* $\mathcal{T}$ *et* $\mathcal{T}_i = (\mathbf{C}, \mathbf{R}, \mathbf{A}_i)$ *(c.-à-d.,*$\mathcal{T}_i$ *se compose de tous les noms de concept et tous les noms de rôle de* $\mathcal{T}$*), où* $\mathbf{A}_i = \{(i : C \sqsubseteq i : D)\}$*,* $\mathbf{A}_i \subseteq \mathbf{A}$; $\cup_i \mathbf{A}_i = \mathbf{A}$ *et* $\cap_i \mathbf{A}_i = \emptyset$*,* $\forall i \in I$*.*
- $\mathcal{B}_{ij}$ *est un ensemble de mappings sémantiques (règles de pont identiques) entre* $\mathcal{T}_i$ *et* $\mathcal{T}_j$*. Un mapping sémantique est une relation identique entre deux concepts (deux rôles) qui sont présentés dans deux axiomes de différentes TBox (*$\mathcal{B}_{ij}$ *doit être vide si* $\mathbf{A}_i$ *et* $\mathbf{A}_j$ *sont "disjoints"* [1]*).*

Pour une décomposition overlay $\mathfrak{T} = \langle\{\mathcal{T}_i\}, \{\mathcal{B}_{ij}\}_{i\neq j\in I}\rangle$ de TBox $\mathcal{T}$, $\mathcal{T}$ est appelée la *TBox originelle*, $\mathfrak{T}$ est appelé la *TBox-décomposante* (écrite brièvement TBD), et chaque $\mathcal{T}_i$ est appelé une *TBox composante* (*TBox locale* ou sous-TBox). Notons que, quand nous disons "la décomposition de l'ontologie", "TBox" est alors remplacée par "ontologie".

Nous trouvons facilement qu'une décomposition overlay d'une TBox $\mathcal{T}$ est une TBox distribuée spéciale avec le $\mathcal{B}_{ij}$ qui inclut seulement des règles de pont identiques. La TBox-décomposante capture toutes les propriétés de TBox distribuée générale (comme donné dans la section ci-dessus) et quelques propriétés particulières. Concrètement, nous avons une définition pour la satisfaisabilité de l'interprétation de TBox-décomposante comme suit :

Définition 5.9. *Une interprétation distribuée* $\mathfrak{I} = (\{\mathcal{I}_i\}_{i\in I}, \{r_{ij}\}_{i\neq j\in I})$ *satisfait les éléments d'une*

[1] $\mathbf{A}_i$ et $\mathbf{A}_j$ sont disjoints s'il n'y a pas un concept (un rôle) commun qui soit présent dans deux expressions de $\mathbf{A}_i$ et de $\mathbf{A}_j$

TBox-décomposante $\mathfrak{T} = (\{\mathcal{T}_i\}_{i\in I}, \mathcal{B})$ *si :*

$$\mathfrak{J} \models_d i : A \underset{\rightarrow}{\equiv} j : A \quad si \quad r_{ij}(A^{\mathcal{I}_i}) \equiv A^{\mathcal{I}_j}$$

$$\mathfrak{J} \models_d i : A \sqsubseteq B \quad si \quad \mathcal{I}_i \models A \sqsubseteq B$$

$$\mathfrak{J} \models_d i : A \sqsubseteq j : B \quad si \quad r_{ij}(A^{\mathcal{I}_i}) \subseteq B^{\mathcal{I}_j}$$

$$\mathfrak{J} \models_d \mathcal{T}_i \quad si \quad \mathcal{I}_i \models \mathcal{T}_i$$

$$\mathfrak{J} \models_d \mathfrak{T} \quad si \quad \mathfrak{J} \models_d \mathcal{T}_i \; et \; \mathfrak{J} \models_d \mathcal{B}_{ij}, \forall i, j \in I$$

Et donc,

$$\mathfrak{T} \models_d i : A \sqsubseteq B \quad si \quad \forall \mathfrak{J}, \mathfrak{J} \models_d \mathfrak{T} \Longrightarrow \mathfrak{J} \models_d i : A \sqsubseteq B$$

$$\mathfrak{T} \models_d i : A \sqsubseteq j : B \quad si \quad pour \; \forall \mathfrak{J}, \; \mathfrak{J} \models_d \mathfrak{T} \Rightarrow \; \mathfrak{J} \models_d i : A \underset{\rightarrow}{\sqsubseteq} j : B$$

Notons que, tous les $\mathcal{T}_i, (i \in I)$ ont le même ensemble de concepts et de rôles que $\mathcal{T}$. Ainsi nous écrivons $i : X$ pour prouver que nous parlons du concept (rôle) X dans $\mathcal{T}_i$. En outre, nous notons en particulier que LDD exprime l'*"orientation/aucun flux de retour"* ("directionality" / "no backflow"), c.-à-d., l'ensemble de règles de pont $\mathcal{B}_{ij}$ implique que le flux (courant) de l'information est seulement propagé de $\mathcal{T}_i$ à $\mathcal{T}_j$, et pas dans la direction inverse. Nous utilisons cette propriété pour éviter la boucle infinie de la propagation des algorithmes de tableaux sur des ontologies locales dans le processus du raisonnement. Par conséquent nous devons indiquer la direction de règles de pont pour chaque paire de TBox. Pour la convenance, nous prescrivons que s'il y a un ensemble de règles de pont $\mathcal{B}_{ij}$ de $\mathcal{T}_i$ à $\mathcal{T}_j$ alors $\mathcal{T}_i$ s'appelle la TBox *source* et $\mathcal{T}_j$ s'appelle la TBox *cible*. Pour simplifier, les travaux dans ce mémoire considèrent seulement le cas le plus simple avec deux TBox composantes ($I = \{1, 2\}$).

Exemple 5.1 :

Pour illustrer une décomposition de TBox $\mathcal{T}_{uni}$ dans la table 1.2, nous voyons la figure 5.1.

La figure 5.1 dépeint deux TBox décomposées $\mathcal{T}_1, \mathcal{T}_2$ de TBox $\mathcal{T}_{uni}$. $\mathcal{T}_1, \mathcal{T}_2$ ont un mapping sémantique 1 :THESE-DOCTORALE $\underset{\rightarrow}{\equiv}$ 2 :THESE-DOCTORALE
où THESE-DOCTORALE est un nom de concept.
$\mathcal{T}_1$ et $\mathcal{T}_2$ sont distingués par leurs ensembles d'axiomes, et $\mathcal{T}_{uni}, \mathcal{T}_1$ et $\mathcal{T}_2$ partagent un ensemble commun de concepts et de rôles :
{ETUDIANT, PERSONNE, ENSEIGNANT, THESARD, THESE, PUBLICATION, ARTICLE, LIVRE, THESE-MASTER, THESE-DOCTORALE, DOCTEUR, PROFESSEUR, avoirPub, travailler}.

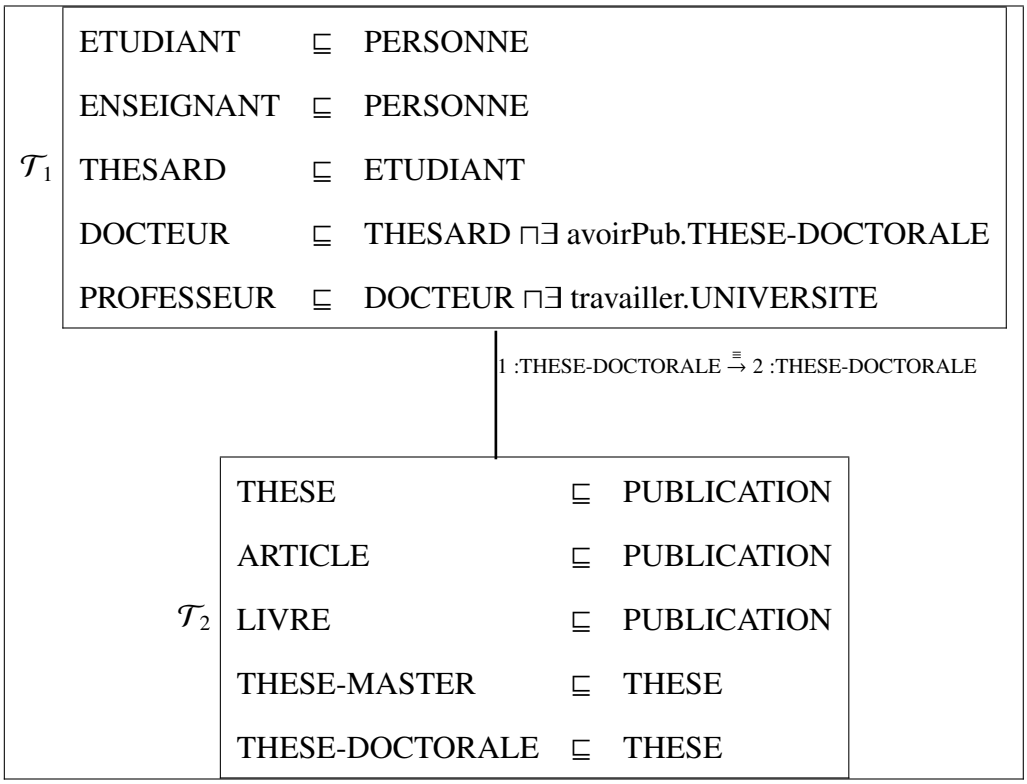


Fig. 5.1 : TBox-décomposante de TBox $\mathcal{T}_{uni}$

5.2.2 Relation entre TBox-décomposante et TBox ordinaire

La TBox-décomposante est également une TBox, ses composants peuvent être déployés dans des éléments primitifs par la définition suivante :

Définition 5.10. *(Expansion) Une expansion $Ex(.)$ d'une description de concept (ou d'une règle de pont) est l'ensemble des concepts primitifs et des rôles primitifs qui apparaissent dans leurs expressions de définition :*

- *Si A est un concept (rôle) atomique ou sa négation, alors $Ex(A) = \{A\}$*
- *Si A est un concept défini sous la forme $\exists R.C$ ou $\forall R.C$, alors $Ex(A) = \{A\} \cup Ex(R) \cup Ex(C)$*
- *Si A est un concept défini sous la forme $C_1 \sqcap C_2$ ou $C_1 \sqcup C_2$, alors $Ex(A) = \{A\} \cup Ex(C_1) \cup Ex(C_2)$*
- *Si A est une description de concept composé sous la forme $\rho(M_1, ..., M_k)$, où ρ est un constructeur de concept, $Ex(\rho(M_1, ..., M_k)) = \cup_i \{Ex(M_i)\}$*
- *Pour une règle de pont into (onto) $B_k = 1 : C \stackrel{\sqsubseteq}{\rightarrow} 2 : D$ ($B_k = 1 : C \stackrel{\sqsubseteq}{\rightarrow} 2 : D$),*

$Ex(B_k) = Ex(C) \cup Ex(D)$

– *Pour une règle de pont identique* $B_k = 1 : C \xrightarrow{\equiv} 2 : C$, $Ex(B_k) = Ex(C)$

L'expansion d'une description de concept C peut être vue comme l'ensemble des concepts et des rôles construisant le concept C. Nous pouvons étendre cette définition pour les axiomes, l'ensemble d'axiomes et l'ensemble de règles de pont :

– pour un axiome $C \sqsubseteq D$:

$$Ex(C \sqsubseteq D) = Ex(C) \cup Ex(D)$$

– pour l'ensemble d'axiomes $\{C_k \sqsubseteq D_k\}$:

$$Ex(\{C_k \sqsubseteq D_k\}) = \bigcup_k (Ex(C_k) \cup Ex(D_k))$$

– pour un ensemble des règles de pont $\mathcal{B}_{ij}$:

$$Ex(\{B_k\}) = \bigcup_k Ex(B_k), \text{pour } B_k \in \mathcal{B}_{ij}$$

D'après la définition de décomposition de TBox, les TBox composantes ont une relation stricte avec la TBox originelle. Nous nous intéressons à la recherche de cette relation, parce qu'elle nous permet de transférer des techniques de raisonnement d'une TBox ordinaire à une TBox-décomposante.

Pour avoir une vision logique de la décomposition de TBox, à partir d'une TBox ordinaire représentée dans un langage de LD $\mathcal{L}$, nous pouvons construire une TBox-décomposante avec les TBox composantes qui sont représentées dans le même langage $\mathcal{L}$. Pour distinguer les descriptions dans les TBox composantes différentes, nous ajoutons un index devant comme un préfixe. Avec un concept primitif A (resp. rôle R) de la TBox originelle $\mathcal{T}$, soit (i, A) (resp. (i, R)) un concept (resp. rôle) primitif de la TBox $\mathcal{T}_i$. Par ailleurs, les TBox composantes utilisent tous les constructeurs de concept (de rôle) de TBox originelle $\mathcal{T}$, donc les TBox composantes $\mathcal{T}_i$ permettent les équivalences de toutes les descriptions composées de $\mathcal{T}$. En effet, nous commençons par définir qu'une *fonction de décomposition* (comme donnée dans [BS03]) est une application de concepts et de rôles de la TBox originelle aux concepts et rôles des TBox composantes.

Définition 5.11. *(Fonction de Décomposition) Une fonction de décomposition* #() *envoie des concepts et des rôles de la TBox originelle* $\mathcal{T}$ *vers des concepts et des rôles dans* $\mathcal{T}_i$ *(*$i \in I$*) comme suit :*

1. *Si M est un concept (rôle) primitif ou* Top *ou Bot, alors* $\#(M) = (i, M)$.

2. *Si M est une expression de concept avec le constructeur de concept ρ prenant k arguments, $M = \rho(M_1, ..., M_k)$, alors*

   $\#(\rho(M_1, \ldots, M_k)) = Top_i \sqcap \rho(\#(M_1), \ldots, \#(M_k))$

3. *Si M est une expression de rôle avec le constructeur de rôle ρ prenant k arguments, $M = \rho(M_1, ..., M_k)$, alors*

   $\#(\rho(M_1, \ldots, M_k) = \rho(\#(M_1), \ldots, \#(M_k))$

où, Top, Bot dénotent les concepts de "top" et de "bottom" de $\mathcal{T}$ respectivement. Ils sont distingués avec des concepts de top (Top_i) et de bottom (Bot_i) de $\mathcal{T}_i$.

Exemple 5.2 :

#(ETUDIANT⊓∃ avoirPub.ARTICLE) fournit

TOP⊓*i* :ETUDIANT ⊓∃(*i* :avoirPub).(TOP⊓*i* :ARTICLE

Nous fournissons maintenant une TBD dans la LDD en appliquant # pour une TBox originelle $\mathcal{T}$, alors un $TBD\ \#(\mathcal{T}) = (\{\mathcal{T}_i\}_{i\in I}, \mathcal{B}_{ij})$ sera créé dans un langage de LDD, où $\mathcal{B}_{ij}$ est l'ensemble des règles de pont identiques, qui compose les axiomes suivants :

1. copier tous les axiomes de TBox originelle à chaque TBox locale $\mathcal{T}_i, i \in I$:
   $i : \#(X) \sqsubseteq \#(Y)$, pour tous $X \sqsubseteq Y \in \mathcal{T}$ et $\#(X) \in \mathcal{T}_i$; $\#(Y) \in \mathcal{T}_i$
2. $i : \#(X) \underset{\rightarrow}{\equiv} j\ \#(X)$, pour tous $X \in \mathcal{T}$, $\#(X) \in \mathcal{T}_i$; $\#(X) \in \mathcal{T}_j$ et $X \in Ex(\mathbf{A}_i), X \in Ex(\mathbf{A}_2)$
3. restriction de relation de domaine r_{ij} telle qu'elle connecte seulement des objets dans $\mathcal{T}_i$ et $\mathcal{T}_j$:
   $Top \sqsubseteq \forall r_{ij}.Top_j$ (le domaine de r_{ij} est dans $\Delta^{\mathcal{I}_j}$)
   $\neg(Top_i) \sqsubseteq \forall r_{ij}.Bot$ (r_{ij} n'est pas défini en dehors $\Delta^{\mathcal{I}_i}$)
4. Top$\sqsubseteq Top_i$, pour s'assurer que Top_i n'est pas vide.
5. $Bot_i \sqsubseteq Bot$, pour limiter Bot_i tel qu'il est toujours le concept incohérent.
6. $(i, A) \sqsubseteq Top_i$ (A est un concept atomique de $\mathcal{T}_i$) pour s'assurer que Top_i est le top local approprié de sa hiérarchie d'IS-A.
7. pour s'assurer que chaque rôle i, r est dans le même domaine et l'espace de Top_i :
   $Top_i \sqsubseteq \forall(i, R).(Top_i)$, pour chaque rôle R de $\mathcal{T}$, l'espace de (i, R) est dans $\Delta^{\mathcal{I}_i}$
   $\neg(Top_i) \sqsubseteq \forall(i, R).Bot$, ($R$ est seulement défini dans $\Delta^{\mathcal{I}_i}$)

5.2.3 Propriétés de décomposition

Dans la décomposition overlay, la relation de domaine r_{ij} entre $\mathcal{I}_i$ et $\mathcal{I}_j$ peut être considérée comme un mapping identique qui applique un objet de $\Delta^{\mathcal{I}_i}$ à lui même dans $\Delta^{\mathcal{I}_j}$, c.-à-d., $r_{ij}(x) = x$.

Selon la manière de construction de la décomposition overlay, nous obtenons facilement que les TBox composantes sont consistantes d'après la TBox originelle. Par conséquent, nous soulignons que si la TBox originelle est satisfaisable alors toutes les TBox composantes sont également satisfaisables. Ceci est très important, parce qu'il évite le problème d'inconsistance dans la LDD, comme indiqué dans [BS03], et assure la *préservation d'inférences*.

Nous définissons une nouvelle relation entre les interprétations de deux TBox comme suit :

Définition 5.12. *Soient deux Tbox $\mathcal{T}_i$ et $\mathcal{T}_j, i \neq j$, avec leurs interprétations $\mathcal{I}_i = (\Delta^{\mathcal{I}_i}, \cdot^{\mathcal{I}_i})$ et $\mathcal{I}_j = (\Delta^{\mathcal{I}_j}, \cdot^{\mathcal{I}_j})$ respectivement. Nous disons que $\mathcal{T}_i$ et $\mathcal{T}_j$ sont interprétées dans le même domaine (ou $\mathcal{I}_i$ et $\mathcal{I}_j$ ont le même domaine) ssi $\Delta^{\mathcal{I}_i} = \Delta^{\mathcal{I}_j}$ et $\cdot^{\mathcal{I}_i} = \cdot^{\mathcal{I}_i}$ pour tous concepts (rôles) primitifs communs.*

À partir cette définition, nous obtenons des résultats dans les propositions suivantes :

Proposition 5.13. *Soit une TBox $\mathcal{T}$ et sa TBox-décomposante $\#(\mathcal{T}) = (\{\mathcal{T}_i\}_{i\in I}, \mathcal{B}_{ij})$, alors $\mathcal{T}$ et T_i $(i \in I)$ sont interprétées dans le même domaine.*

La preuve de cette proposition est facilement impliquée par les définitions 5.8, 5.9 et 5.12

Proposition 5.14. *Si deux TBox T_i et T_j sont interprétées dans le même domaine, alors $C^{\mathcal{I}_i} = C^{\mathcal{I}_j}$, pour tout C est une description de concept commun de T_i et T_j.*

La preuve de cette proposition [PLT06, PTS08] reposera sur la relation sémantique des descriptions de concepts entre les TBox composantes et la TBox originelle (i.e., les TBox sont interprétées dans le même domaine). On sait que les descriptions de concepts sont établies à partir de l'ensemble des noms de concepts, des noms de rôles et des constructeurs d'un langage de LD particulier.

En particulier, nous examinons quelques cas élémentaires d'une description de concept C (de rôle R) qui est présentée dans toute paire de TBox $\mathcal{T}_i$ et $\mathcal{T}_j$, où $\mathcal{T}_i$ et $\mathcal{T}_j$ sont interprétées dans le même domaine, c.-à-d., $\Delta^{\mathcal{I}_i} = \Delta^{\mathcal{I}_i}$ et $\cdot^{\mathcal{I}_i} = \cdot^{\mathcal{I}_i}$ pour tous concepts (rôles) primitifs communs.

Soient $C_1, ..., C_n$ les concepts primitifs communs, $r_1, ..., r_n$ les rôles primitifs communs de $\mathcal{T}_i$ et $\mathcal{T}_j$, alors $(C_k)^{\mathcal{I}_i} = (C_k)^{\mathcal{I}_j}$ et $(R_k)^{\mathcal{I}_i} = (R_k)^{\mathcal{I}_j}$ (de la définition 5.12), pour $k, p, q = \overline{1, n}$, nous avons :

- Si C est de la forme $C_1 \sqcap ... \sqcap C_n$ (ou $C_1 \sqcup ... \sqcup C_n$), alors
  $(C_1 \sqcap ... \sqcap C_n)^{\mathcal{I}_i} = (C_1)^{\mathcal{I}_i} \cap ... \cap (C_n)^{\mathcal{I}_i} = (C_1)^{\mathcal{I}_j} \cap ... \cap (C_n)^{\mathcal{I}_j} = (C_1 \sqcap ... \sqcap C_n)^{\mathcal{I}_j}$
- Si C est de la forme $\neg C_k$, alors $(\neg C_k)^{\mathcal{I}_i} = \Delta^{\mathcal{I}_i} \setminus C_k^{\mathcal{I}_i} = \Delta^{\mathcal{I}_j} \setminus C_k^{\mathcal{I}_j} = (\neg C_k)^{\mathcal{I}_j}$
- Si C est de la forme $\forall r_k.C_k$, alors
  $(\forall R_k.C_k)^{\mathcal{I}_i} = \{x \in \Delta^{\mathcal{I}_i} | \forall y : (x,y) \in R_k^{\mathcal{I}_i} \Rightarrow y \in C_k^{\mathcal{I}_i}\} = \{x \in \Delta^{\mathcal{I}_j} | \forall y : (x,y) \in R_k^{\mathcal{I}_j} \Rightarrow y \in C_k^{\mathcal{I}_j}\} = (\forall R_k.C_k)^{\mathcal{I}_j}$
- Si C est de la forme $\exists R_k.C_k$, alors
  $(\exists R_k.C_k)^{\mathcal{I}_i} = \{x \in \Delta^{\mathcal{I}_i} | \exists y : (x,y) \in R_k^{\mathcal{I}_i} \wedge y \in C_k^{\mathcal{I}_i}\} = \{x \in \Delta^{\mathcal{I}_j} | \exists y : (x,y) \in R_k^{\mathcal{I}_j} \Rightarrow y \in C_k^{\mathcal{I}_j}\} = (\exists R_k.C_k)^{\mathcal{I}_j}$
- Si C est de la forme $\geq nR_k.C_k$ ou $(\leq nR_k.C_k)$, alors
  $(\geq nR_k.C_k)^{\mathcal{I}_i} = \{x \in \Delta^{\mathcal{I}_i} | card\{y \in \Delta^{\mathcal{I}_i} | (x,y) \in R_k^{\mathcal{I}_i}, y \in C_k^{\mathcal{I}_i}\} \geq n\} = \{x \in \Delta^{\mathcal{I}_j} | card\{y \in \Delta^{\mathcal{I}_j} | (x,y) \in R_k^{\mathcal{I}_j}, y \in C_k^{\mathcal{I}_k}\} \geq n\} = (\geq nR_k.C_k)^{\mathcal{I}_j}$
- Si R est de la forme R_k^-, alors
  $(R_k^-)^{\mathcal{I}_i} = \{(x,y) | (y,x) \in r^{\mathcal{I}_i}\} = \{(x,y) | (y,x) \in R^{\mathcal{I}_j}\} = (R_k^-)^{\mathcal{I}_j}$
- Si R est de la forme $R_p \sqcap R_q$ (ou $R_p \sqcup R_q$), $p, q \in \overline{1,n}$, alors :
  $(r_p \sqcap R_q)^{\mathcal{I}_i} = (R_p)^{\mathcal{I}_i} \cap (R_q)^{\mathcal{I}_i} = (R_p)^{\mathcal{I}_j} \cap (R_q)^{\mathcal{I}_j} = (R_p \sqcap R_q)^{\mathcal{I}_j}$

En généralisation (d'une manière générale), comme Borgida et Serafini ont déjà mentionné dans [BS03], les sémantiques de descriptions de concepts qui sont dénotés par $\rho(\arg_1, .., \arg_n)$ correspondent à une fonction f_ρ de leurs arguments. Nous examinons quelques exemples :

Exemple 5.3 :

$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$ est égal à $f_{and}(C_1^{\mathcal{I}}, C_2^{\mathcal{I}})$ pour une *fonction sémantique* associée $f_{and}(XC_1, XC_2) = \{d \in XC_1 \cap XC_2\}$

Exemple 5.4 :

$(\forall R.C)^{\mathcal{I}} = all(R,C)^{\mathcal{I}}$ est égal à $f_{all}(R^{\mathcal{I}}, C^{\mathcal{I}})$ pour une fonction sémantique associée $f_{all}(XR, XC, \Delta^{\mathcal{I}}) = \{d \in \Delta^{\mathcal{I}} | XR(d) \subseteq XC\}$

Cependant, une fonction sémantique peut dépendre de l'interprétation de ses arguments (est $\Delta^{\mathcal{I}}$ dans les cas au-dessus). Donc, $(C_1 \sqcap C_2)^{\mathcal{I}}$ sera exprimé comme $f_{and}(C_1^{\mathcal{I}}, C_2^{\mathcal{I}}, \Delta^{\mathcal{I}})$, et $(\forall R.C)^{\mathcal{I}}$ est comme $f_{all}(R^{\mathcal{I}}, C^{\mathcal{I}}, \Delta^{\mathcal{I}})$. En particulier, pour insister sur cette dépendance, nous examinons seulement des constructeurs qui dépendent explicitement de $\Delta^{\mathcal{I}}$. Par exemple, f_{all} peut prendre les arguments additionnels XR, XC, DY substitués les $r^{\mathcal{I}}, C^{\mathcal{I}}, \Delta^{\mathcal{I}}$, alors $f_{all}(XR, XC, DY) = \{d \in DY | XR(d) \subseteq XC\}$. Pareillement avec f_{and}. Nous fournissons une définition de fonction sémantique pour quelques constructeurs dans la table 5.1 [BS03] :

Définition 5.15. *Soit ρ un constructeur de concept dont la sémantique peut être exprimée par $\rho(\arg_1, .., \arg_n)^{\mathcal{I}} = f_\rho(\arg_1^{\mathcal{I}}, ..., \arg_n^{\mathcal{I}}, \Delta^{\mathcal{I}})$, pour une fonction $f_\rho(X_1, ..., X_n, DY)$ dont la définition*

Constructeurs de concept	Fonction sémantique
$C_1 \sqcap C_2$	$f_{and}(XC_1, XC_2, DY) = XC_1 \cap XC_2$
$C_1 \sqcup C_2$	$f_{or}(XC_1, XC_2, DY) = XC_1 \cup C_2$
$\neg C$	$f_{not}(XC, DY) = \{x \in DY \mid x \notin XC\}$
$\exists R.C$	$f_{exists}(XR, XC, DY) = \{x \in DY \mid XR(x) \cap XC \neq \emptyset\}$
$\forall R.C$	$f_{for\_all}(XR, XC, DY) = \{x \in DY \mid XR(x) \subseteq XC\}$
$\geq nR.C$	$f_{q\_at\_least}(n, XR, XC, DY) = \{x \in DY \mid card\{XR(x) \cap XC \geq n\}$
$\leq nR.C$	$f_{q\_at\_most}(n, XR, XC, DY) = \{x \in DY \mid card\{XR(x) \cap XC \leq n\}$
Constructeurs de rôle	**Fonction sémantique**
R^-	$f_{inverse}(XR, XD) = \{(x, y) \mid (y, x) \in XR\}$
$R_1 \sqcap R_2$	$f_{role\_and}(XR_1, XR_2, DY) = XR_1 \cap XR_2$
$R_1 \sqcup R_2$	$f_{role\_or}(XR_1, XR_2, DY) = XR_1 \cup XR_2$

Tab. 5.1 : Fonctions sémantiques de constructeurs de concept et de rôle

ne contient aucune référence à $\mathcal{I}$. Soient $B_1, ..., B_n, W, \Delta$ les ensembles tels que $W \subseteq \Delta$, et chaque B_j est un sous-ensemble de W ou de $W \times W$, avec $1 \leq j \leq n$. Alors ρ s'appelle un constructeur local si f_ρ satisfait : $f_\rho(B_1, ..., B_n, \Delta) = f_\rho(B_1, ..., B_n, W)$, quand il est un constructeur de concept ou de rôle.

Preuve : (Proposition 5.14) Parce que $\mathcal{T}_i$ et $\mathcal{T}_j$ sont interprétées dans le même domaine, nous avons $C^{\mathcal{I}_j} \equiv C^{\mathcal{I}_i}$, $R^{\mathcal{I}_j} \equiv R^{\mathcal{I}_i}$, où C, r sont des concepts et des rôles primitifs communs respectivement. Par induction sur la structure d'un concept (rôle) arbitraire B, nous montrons facilement que $B^{\mathcal{I}_j} \equiv B^{\mathcal{I}_i}$.

Nous examinons le cas où M est une description de concept commun arbitraire construite de concepts (rôles) primitifs communs. Supposons que M est de la forme $\rho(M_1, ..., M_k)$, où les concepts $M_i, ..., M_k$ sont moins complexes que M, et se produisent également dans $\mathcal{T}_j$, ρ est un constructeur de concept prenant k arguments.

Par induction structurée sur M, nous obtenons $M_p^{\mathcal{I}_j} \equiv M_p^{\mathcal{I}_i}\ \forall p = 1, .., k$. Nous devons maintenant montrer que $M^{\mathcal{I}_i} = M^{\mathcal{I}_j}$. En effet, $M^{\mathcal{I}_j} \equiv (\rho(M_1, ..., M_k))^{\mathcal{I}_j} \equiv f_\rho(M_1^{\mathcal{I}_j}, ..., M_k^{\mathcal{I}_j}, \Delta^{\mathcal{I}_j}) \equiv f_\rho(M_1^{\mathcal{I}_i}, ..., M_k^{\mathcal{I}_i}, \Delta^{\mathcal{I}_i}) \equiv (\rho(M_1, ..., M_k))^{\mathcal{I}_i} \equiv M^{\mathcal{I}_i}$.

□

En outre, dans la TBox-décomposante, nous obtenons les propriétés importantes suivantes :

Proposition 5.16. *Soit une TBox $\mathcal{T}$, alors la TBox-décomposante $\langle\{\mathcal{T}_i\}, \mathcal{B}_{ij|i\neq j\in\{1,2\}}\rangle$ de $\mathcal{T}$ a les propriétés suivantes :*

1. *La préservation des concepts (des rôles) : chaque concept (rôle) de $\mathcal{T}$ est également un concept (rôle) dans tout $\mathcal{T}_i$*
2. *La préservation des axiomes : ($\mathbf{A} = \cup_i \mathbf{A}_i$) : chaque axiome $C \sqsubseteq D$ de l'ontologie $\mathcal{T}$ est également un axiome dans une des $\mathcal{T}_i$. Notons qu'il n'existe pas d'axiome qui apparaît dans deux ontologies différentes (c.-à-d., les ensembles d'axiomes de deux ontologies différentes sont disjoints).*
3. *Les mappings sémantiques sont proposés afin de conserver les relations des entités entre sous-ontologies. Ils présentent seulement des relations identiques entre deux concepts (deux rôles) qui se produisent dans deux axiomes de deux ontologies différentes. Ils sont représentés par des règles de pont dans $\mathcal{B}$.*

4. *La préservation des sémantiques :* $\Delta = \Delta_i$, *et* $\cdot^{\mathcal{I}} = \cdot^{\mathcal{I}_i}$ *pour tous les concepts (rôles) communs entre* $\mathcal{T}$ *et* $\mathcal{T}_i$, *avec* $\mathcal{I}, \mathcal{I}_i$ *qui sont les interprétations de* $\mathcal{T}$ *et* $\mathcal{T}_i$ *respectivement* $(i = 1, 2)$.

5. *La préservation des services d'inférence : si une description de concept est satisfaisable par rapport à* $\mathcal{T}$, *alors elle est également satisfaisable par rapport à* $\langle \mathcal{T}_i, \mathcal{B} \rangle$, *et réciproquement.*

Preuve : Les trois premières propriétés sont faciles à déduire de la définition de la décomposition, et elles prouvent la conservation de la syntaxe. Ces propriétés assurent que ($\{\mathcal{T}_i\}$,$\mathcal{B}$) est bien représenté par la LDD.

La quatrième propriété est déduite de la proposition 5.14. La dernière propriété sera montrée dans le chapitre suivant.

□

Dans les sections suivantes, nous nous concentrons sur les deux dernières propriétés - les plus importantes propriétés de décomposition - les préservations de sémantique et d'inférence. Les définitions et preuves sont seulement examinées dans une décomposition $\langle \{\mathcal{T}_i\}, \mathcal{B} \rangle$ de $\mathcal{T}$, c.-à-d., $\mathcal{T}$ et $\{\mathcal{T}_i\}$ ont le même ensemble de noms de concepts et de rôles. Par conséquent, au lieu de dire "le nom de concept (rôle) primitif commun" nous utilisons la notation plus courte "le nom de concept (rôle)".

En plus des ces propriétés, nous proposons deux approches du raisonnement pour des TBox composantes. En caractérisant la forme de décomposition, nous prouverons qu'il existe un algorithme déterministe qui permet de décomposer une ontologie représentée par une LD en plusieurs ontologies représentées par la LDD qui est conforme à cette forme.

5.3 Discussion et conclusion

Comme nous l'avons introduit dans la Section 4.5, il y a un intérêt significatif pour les langages modulaires de l'ontologie, comme les $\mathcal{E}$-connections, les logiques de description distribuées, les fusions, les logiques de description basées sur le paquet (package-based description logics). Toutefois, si nous avons utilisé une LDD pour présenter une décomposition de l'ontologie, c'est parce que les relations entre les TBox composantes sont proches des règles de pont dans une LDD, et les domaines des TBox composantes ont le même domaine que la TBox originelle. En effet, pour les $\mathcal{E}$-connections, l'idée fondamentale est que les domaines d'interprétation du système de n composants combinés sont disjoints, et que ces domaines sont

connectés par le sens des relations liées n-aires. La LDD est également utilisée dans le cas où les ontologies composantes sont disjointes, mais leurs domaines d'interprétation peuvent contenir une partie couvrante et les relations entre eux sont représentées par des règles de pont. L'approche de la logique de description basée sur le paquet permet à un module d'ontologie de faire directement référence aux termes définis dans d'autres modules d'ontologie en important des termes étrangers. Les règles de pont dans la LDD et les liens de $\mathcal{E}$-connections peuvent être réduites à des axiomes de P-DL [BCH06b]. Cette approche peut fournir une expressivité plus forte, des connexions sémantiques précises dans des applications plus générales. Néanmoins, le raisonnement dans un P-LD ne considère que les requêtes locales, c.-à-d., la vérification de satisfaisabilité est seulement réalisée avec le concept local $i : C$.

En outre, notre ontologie de décomposition est représentée comme un cas particulier de LDD dans lequel les relations entre les ontologies locales ne sont que les règles de pont identiques. L'essence de cette décomposition est d'examiner seulement les axiomes, c.-à-d., l'ensemble d'axiomes de TBox originelle sont divisés en plusieurs sous-ensembles d'axiomes pour les sous-TBox. Nous sommes concernés par les axiomes parce que leur présence peut mener à une limite inférieure d'ExpTime pour la complexité du problème de raisonnement [Fra03]. Cependant, le grand nombre des concepts et des rôles présente également des difficultés pour maintenir, réutiliser et se développer indépendamment des ontologies composantes. Mais le but de ce mémoire est justement de considérer le problème d'optimisation du raisonnement qui dépend d'une méthode de décomposition. Intuitivement, une décomposition $\langle\{\mathcal{T}_i\}, \mathcal{B}_{ij}\rangle$ de $\mathcal{T}$ est " bonne " si la quantité de liens entre les ontologies composantes est aussi petite que possible et s'il y un équilibre cardinal des axiomes dans deux sous-ontologies. Nous devons donner une stratégie pour décomposer $\mathcal{T}$ en des ensembles de $\{\mathcal{T}_{i,i\in I}\}$ avec chaque $\mathcal{T}_i$ est un formalisme de LD d'une certaine ontologie.

Comme un résultat naturel, l'algorithme de raisonnement dans le système des ontologies décomposées sera réalisé en se basant sur l'idée de l'algorithme de tableaux distribué. Nos algorithmes assurent la préservation du raisonnement de TBox originelle. En effet, l'efficacité du raisonnement dans notre décomposition de l'ontologie avec LDD sera montrée dans le Chapitre 6.

Chapitre 6

Raisonnement sur la TBox-décomposante

Le raisonnement dans le système des ontologies composantes (TBox composantes) est mis en application avec deux méthodes. Dans la première, qui s'appelle *méthode distribuée*, les TBox composantes sont représentées dans un système distribué par la logique de description distribuée et un algorithme de tableaux distribué est appliqué. Dans la seconde, appelée *méthode parallèle*, on applique l'algorithme de tableaux normal. Tandis que plusieurs autres techniques d'optimisation sont valides seulement pour certaines LDs (elles ont été créées en développant les systèmes de LD et sont applicables à un cadre étendu des systèmes de raisonnement), nos techniques sont complètement indépendantes de la LD soutenue par les raisonneurs. Les algorithmes sont conçus de telle sorte que les résultats de raisonnement obtenus dans les ontologies composantes soient les mêmes que pour le raisonnement dans l'ontologie originelle.

Dans ce chapitre, nous allons présenter en détail les éléments d'une TBox-décomposante. Ensuite, deux méthodes du raisonnement sur la TBox-décomposante seront décrites avec les algorithmes correspondants. La préservation des résultats du raisonnement sera également démontrée avec les propriétés de la terminaison, la correction et la complétude des algorithmes du raisonnement. Les discussions de l'impact de décomposition sur la complexité, des relations entre la décomposition et la logique de description distribuée, du raisonnement parallèle et distribué, et quelques conclusions termineront le chapitre.

6.1 TBox-décomposante

Le raisonnement dans une large ontologie peut être réduit à quelques procédures du raisonnement dans des sous-ontologies basées sur des approches parallèles ou distribuées. Les services du raisonnement dans les LDs modernes sont essentiellement les problèmes de satisfaisabilité de concepts et de subsomption qui sont résolus par l'algorithme de tableaux. Pour simplifier la description, nous examinons l'ontologie dans le langage $\mathcal{ALC}$.

Une décomposition de TBox $\mathcal{ALC}$ (une terminologie distribuée $\mathcal{ALC}$) est présentée avec les règles de formation suivantes :

- Un axiome est dans une des formes :

$$i : C \sqsubseteq D | i : C \equiv D | i : r \sqsubseteq S$$

  où C et D sont des expressions de concept dans $\mathcal{T}_i$, R et S sont des noms de rôle dans $\mathcal{T}_i$, et $i \in I$

- Une expression de concept est dans une des formes :

$$i : CN | \top | \bot | \neg i : C | i : C \sqcap D | i : C \sqcup D | \exists R.i : C | \forall R.i : C$$

  où CN est un nom de concept dans $\mathcal{T}_i$, C et D sont des expressions concept dans $\mathcal{T}_i$, R et S sont des noms de rôle.

- Une règle de pont est dans la forme :

$$i : X \underset{\rightarrow}{\equiv} j : X$$

  pour tous $X \in \mathcal{T}$, $X \in \mathcal{T}_i$, $X \in \mathcal{T}_j$ et $X \in Ex(\mathbf{A}_i)$, $X \in Ex(\mathbf{A}_j)$

Les sémantiques des expressions de concepts distribués $\mathcal{ALC}$ sont décrites dans des termes d'une interprétation globale $\Im = \langle \{\mathcal{I}_i\}_{i \in I}, \{r_{ij}\}_{i \neq j \in I} \rangle$ qui doivent satisfaire les propriétés dans la définition 5.4 et des propriétés additive de TBox-décomposante.

Dans les sections suivantes, deux approches du raisonnement pour les ontologies décomposantes seront présentées avec plus de détail, les techniques sont parallèles et distribuées. Les algorithmes de tableaux seront appliqués sur des ontologies locales puis ils sont fusionnés dans le cas parallèle ou sont propagés dans le cas distribué.

6.2 Raisonnement parallèle

L'idée principale du raisonnement parallèle pour une TBox-décomposante $\mathfrak{T} = \langle \{\mathcal{T}_i\}, \{\mathcal{B}_{ij}\} \rangle$ consiste à construire de multiples tableaux locaux $\mathbf{T}_i$ sur $\mathcal{T}_i$ au lieu d'un tableau global simple. Le rôle des TBox locales est le même dans cette approche, c.-à-d., nous n'avons pas besoin de distinguer la TBox source et la TBox cible, parce que les algorithmes de tableaux sont complètement indépendamment mis en application sur des TBox locales, puis sont combinés. Dans [PLT06], nous avons proposé de fusionner tous les noeuds de feuille de tableaux locaux. Néanmoins, dans le pire des cas, c.-à-d., toutes les branches dans des tableaux locaux de $\mathcal{T}_i$ et $\mathcal{T}_j$ sont ouvertes, en fusionnant tous les noeuds dans $\mathbf{T}_i$ et $\mathbf{T}_j$ peut être conduit en temps exponentiel. Pour éliminer le fusionnement superflu, dans cette section nous adressons une condition de fusionnement pour des noeuds dans les tableaux différents.

D'abord, nous présentons une nouvelle notion, *la règle de pont complète*, qui est une relation d'identité $i : X \overset{\equiv}{\rightarrow} j : X$ avec $X \in \{Ex(Q) \cup Ex(\mathcal{B}_{ij})\}$, où Q est une expression de concept C. En d'autres termes, l'ensemble de règles de pont complètes, dénoté par $\mathbb{M}_{ij}$, est étendu de $\mathcal{B}_{ij}$. Il se compose d'un composant fixe ($\mathcal{B}_{ij}$) et d'un composant variable selon les requêtes. Il aide à fusionner (dans le cas parallèle) et à propager (dans le cas distribué) des tableaux locaux. Deux noeuds dans deux tableaux sont seulement fusionnés s'il y a au moins une *règle de pont complète* entre eux. Un tableau combiné des tableaux locaux est défini comme suit :

Définition 6.1. *(Tableau combiné) Un tableau combiné pour une TBox distribuée* $\mathfrak{T} = \langle \{\mathcal{T}_i\}, \{\mathcal{B}_{ij}\} \rangle$ *est une paire* $\langle \{\boldsymbol{T}_i\}, \{m_{ij}\}_{i \neq j} \rangle$*, où* $\boldsymbol{T}_i$ *est un tableau local de* $\mathcal{T}_i$*,* m_{ij} *est la relation d'image entre* $\boldsymbol{T}_i$ *et* $\boldsymbol{T}_j$*, tel qu'il crée les mappings de noeuds de* $\boldsymbol{T}_i$ *vers les noeuds de* $\boldsymbol{T}_j$*. Pour un noeud* $x \in \boldsymbol{T}_i$ *et un noeud* $y \in \boldsymbol{T}_j$*,* $(x, y) \in m_{ij}$ *s'ils contiennent au moins une règle de pont complète.*

Nous examinons concrètement l'essai de satisfaisabilité d'un concept C par rapport à $\mathcal{T}$ et par rapport à $(\{\mathcal{T}_1, \mathcal{T}_2\})$. Afin d'examiner la satisfaisabilité du concept C par rapport à $(\{\mathcal{T}_1, \mathcal{T}_2\})$, nous mettons en application simultanément et indépendamment les algorithmes de tableaux normaux $\mathbf{T}_1(C)$ et $\mathbf{T}_2(C)$ sur les TBox $\mathcal{T}_1$ et $\mathcal{T}_2$ respectivement. Ces résultats sont :

1. Si $\mathbf{T}_1(C)$ ou $\mathbf{T}_2(C)$ est insatisfaisable alors nous concluons que C est également insatisfaisable par rapport à $\mathcal{T}$.
2. Autrement, c.-à-d., tous les $\mathbf{T}_1(C)$ et $\mathbf{T}_2(C)$ sont ainsi satisfaisables alors nous construisons un fusionnement de $\mathbf{T}_1$ et $\mathbf{T}_2$, dénoté $\mathbf{T}_1 \oplus \mathbf{T}_2$, comme suit :

   Après avoir calculé des tableaux sur $\mathcal{T}_1$ et $\mathcal{T}_2$, deux arbres de modèle AM_1 et AM_2 sont respectivement générés. Les noeuds non contradictoires dans AM_1 peuvent être joints avec les noeuds non contradictoires dans AM_2 par paires (s'ils contiennent une règle de pont complète), produisant de nouveaux noeuds dans un arbre modèle de fusionnement.

   Supposons que m noeuds de feuilles non contradictoires de AM_1 soient fusionnés avec n noeuds de feuilles non contradictoires de AM_2 (m, n sont des nombres entiers non négatifs), alors $m \times n$ nouveaux noeuds seront produits dans l'arbre de fusionnement. Si tous ces $m \times n$ noeuds dans l'arbre de fusionnement sont également contradictoires, alors nous concluons que C est insatisfaisable par rapport à $\mathcal{T}_1, \mathcal{T}_2$. Autrement, nous concluons que C est satisfaisable, c.-à-d., si il existe au moins un noeud qui est non-contradictoire, alors nous concluons que C est satisfaisable.

Enfin, fusionnant $\mathcal{L}(x^C_{T_1})$ et x_{T_2}-label $\mathcal{L}(x^C_{T_2})$ en un noeud simple $x_{T_{12}}$-label $\mathcal{L}(x^C_{T_{12}}) = \mathcal{L}(x^C_{T_1}) \cup \mathcal{L}(x^C_{T_2})$, selon les principes suivants :

- $\exists R.C_1 \in \mathcal{L}(x^C_{T_1})$ et $\exists R.C_1 \in \mathcal{L}(x^C_{T_2})$, alors l'union de ces labels se compose d'un noeud $x_{T_{12}}$-label $\mathcal{L}(x^C_{T_{12}}) = \mathcal{L}(x^C_{T_1}) \cup \mathcal{L}(x^C_{T_2})$, et un arc-label R entre $x_{T_{12}}$ et un noeud $y : \{C_1\}$

- $\exists R.C_1 \in \mathcal{L}(x^C_{T_1})$ et $\exists R.C_2 \in \mathcal{L}(x^C_{T_2})$, alors l'union de ces labels consiste en $x_{T_{12}}$-label $\mathcal{L}(x^C_{T_{12}}) = \mathcal{L}(x^C_{T_1}) \cup \mathcal{L}(x^C_{T_2})$, et un label d'arc R entre $x_{T_{12}}$ et un noeud $y : \{C_1, C_2\}$
- $\exists R_1.C_1 \in \mathcal{L}(x^C_{T_1})$ et $\exists R_2.C_2 \in \mathcal{L}(x^C_{T_2})$, alors l'union de ces labels se compose d'un noeud $x_{T_{12}}$-label $\mathcal{L}(x^C_{T_{12}}) = \mathcal{L}(x^C_{T_1}) \cup \mathcal{L}(x^C_{T_2})$, et deux arcs qui sont l'un label R_1 entre $x_{T_{12}}$ et le noeud $y_1 : \{C_1\}$, et l'autre label R_2 entre $x_{T_{12}}$ et le noeud $y_2 : \{C_2\}$
- $\exists R.C_1 \in \mathcal{L}(x^C_{T_1})$, $\forall S.C_2 \in \mathcal{L}(x^C_{T_2})$ et $R \sqsubseteq S$, alors l'union de ces noeuds comprend un noeud $x_{T_{12}}$-label $\mathcal{L}(x^C_{T_{12}}) = \mathcal{L}(x^C_{T_1}) \cup \mathcal{L}(x^C_{T_2})$, et un label d'arc r entre $x_{T_{12}}$ et le noeud $y : \{C_1, C_2\}$

où r, r_1, r_2 et r sont des noms de rôles, C_1 et C_2 sont des noms de concept.

Exemple 6.1 :

Fusionnant deux noeuds x_1 avec label $\mathcal{L}(x_1) = \{C, D, \exists R.C_2, \forall R.C_3\}$ et x_2 avec label $\mathcal{L}(x_2) = \{C, \exists R.C_4, \forall R.C_3, \exists R_1.C_5\}$ des arbres $\mathbf{T}_1$ et $\mathbf{T}_2$ respectivement (voir la Figure 6.2).

Dans le raisonnement parallèle, des algorithmes de tableaux sont appliqués sur des TBox locales, et la terminaison est ainsi assurée par la stratégie de blocage [Hor97].

D'autre part, l'idée de fusionnement dans cet algorithme est basée sur la technique de mise en cache (introduite dans le chapitre 3). Les résultats du raisonnement obtenus sur la TBox-décomposante selon le fusionnement sont les mêmes résultats que sur la TBox originelle. Ceci est possible parce que les problèmes de satisfaisabilité dans des sous-arbres sont complètement indépendants et n'ont aucun effet sur les noeuds ancêtre [Hor97]. Cependant, cette approche de raisonnement montre la difficulté dans l'exécution de fusionnement. Dans le cas où le nombre de noeuds ouverts dans les tableaux locaux est grand, le fusionnement peut être exponentiel en comportement. Ainsi au lieu de fusionner des tableaux locaux, nous proposons d'employer le raisonnement distribué dans la section suivante.

6.3 Raisonnement dans les logiques de description distribuées

Dans cette section, nous introduisons un algorithme de raisonnement basé sur l'idée principale de la procédure de raisonnement distribué de Luciano Serafini et Andrei Tamilin [ST04a, ST04b], qui prend un concept complexe C comme entrée et retourne le résultat de son essai de (in)satisfaisabilité. Dans cette approche, nous nous intéressons à la différence entre la TBox source (dénotée par $\mathcal{T}_s$) et la TBox cible (dénotée par $\mathcal{T}_t$). Pour souligner cette différence, nous dénotons une TBox-décomposante par $\mathfrak{T} = \langle\{\mathcal{T}_s, \mathcal{T}_t\}, \mathcal{B}_{st}\rangle$ au lieu de $\mathfrak{T} = \langle\{\mathcal{T}_i\}, \mathcal{B}_{ij}\rangle$ Pour simplifier, nous examinons seulement les TBox distribuées qui se composent d'une TBox source et d'une TBox cible. Toutefois, ceci peut être déterminé quand on exécute le raisonnement, c.-à-d., si une requête est posée sur la TBox composante $\mathcal{T}_i$ alors $\mathcal{T}_i$ est assignée la TBox source et

Entrée : Le concept C et la TBox-décomposante $\langle \mathcal{T}_{ij}, \mathcal{B}_{ij} \rangle$

Sortie : Satisfaisable / Insatisfaisable

PROCEDURE SAT-PARA $(C, \langle \mathcal{T}_{ij}, \mathcal{B}_{ij} \rangle)$

1 : BEGIN

2 : $\mathbf{T}_1 = Tab_1(C)$;

3 : $\mathbf{T}_2 = Tab_2(C)$;

4 : $\mathbb{M}_{ij} = Ex(\mathcal{B}_{ij}) \cup Ex(C)$;

5 : **if** ($\mathbf{T}_1$ n'est pas contradictoire) et ($\mathbf{T}_2$ n'est pas contradictoire) **then**

6 : $\mathbf{T} = \emptyset$;

7 : **for each** branche ouvert β_1 dans $\mathbf{T}_1$ **do**

8 : **for each** branche ouvert β_2 dans $\mathbf{T}_2$ **do**

9 : repeat

10 : sélecte noeud $x_1 \in \beta_1$ et noeud $x_2 \in \beta_2$

11 : if $\mathcal{L}_1(x_1) \cap \mathcal{L}_2(x_2) \subset M_{ij}$ then

12 : $\mathcal{L}_{12}(x_{12}) = \mathcal{L}_1(x_1) \cup \mathcal{L}_2(x_2)$

13 : $\mathbf{T} = \mathbf{T} \cup x_{12}$

14 : et if

15 : until ((β_1 est ouvert) et (β_2 est ouvert) et (il n'existe aucun noeud non vérifié dans β_1 ou β_2))

16 : **end for**

17 : **end for**

18 : **end if**

19 : **if** (($\mathbf{T}_1$ est contradictoire) ou ($\mathbf{T}_2$ est contradictoire) ou ($\mathbf{T}$ est contradictoire)) then

20 : retourne insatisfaisable

21 : **Else**

22 : retourne satisfaisable

23 : END.

Fig. 6.1 : Algorithme de tableaux parallèle

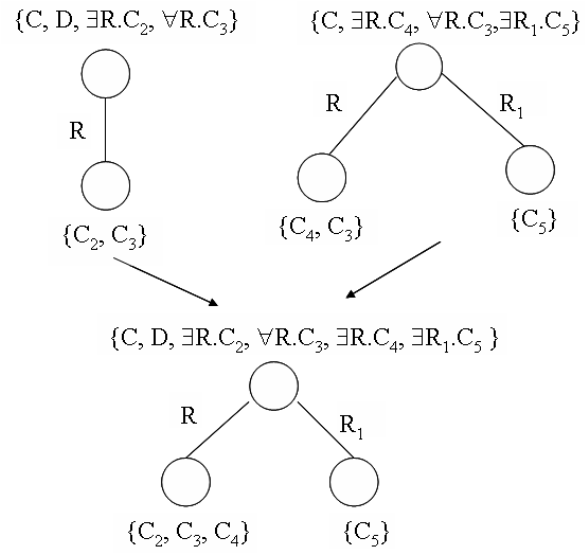


FIG. 6.2 : Fusion de deux noeuds de $\mathcal{T}_1$ et $\mathcal{T}_2$

l'autre devient la TBox cible.

Par exemple, en vérifiant la satisfaisabilité d'une expression de concept C dans $\mathcal{ALC}$ par rapport à $\mathcal{T}$ et par rapport à $\mathfrak{T} = \langle\{\mathcal{T}_s, \mathcal{T}_t\}, \mathcal{B}_{st}\rangle$, où $\mathcal{B}_{st}$ est un ensemble de mappings sémantiques (règles de pont) de $\mathcal{T}_s$ à $\mathcal{T}_t$.

L'idée principale est d'abord de construire un arbre complet (completion tree) local $\mathbf{T}_s$ en parcourant un algorithme de tableaux local sur la TBox source $\mathcal{T}_s$. D'après l'algorithme de tableaux, chaque noeud x généré pendant la construction d'arbre d'accomplissement est marqué par une fonction $\mathcal{L}(x)$ contenant les concepts que x doit satisfaire. Nous essayons ensuite de fermer les branches ouvertes de $\mathbf{T}_s$ en vérifiant la présentation des règles de pont complètes entre $\mathcal{T}_s$ et les autres TBox cible $\mathcal{T}_t$, qui pourraient passer le calcul à ces TBox. Nous notons que la propagation de DTab suit seulement une direction, de s à t, et pas la direction opposée.

Définition 6.2. *Un tableau distribué pour une TBox distribuée $\langle\{\mathcal{T}_s, \mathcal{T}_t\}, \mathcal{B}_{st}\rangle$ dans $\mathcal{ALC}$ est un groupe $D\boldsymbol{T} = \langle\{\boldsymbol{T}_s, \boldsymbol{T}_t\}, \{r_{st}\}_{s\neq t}\rangle$, où $\boldsymbol{T}_s, \boldsymbol{T}_t$ sont des tableaux locaux $\mathcal{ALC}$ sur la TBox source locale $\mathcal{T}_s$, et la TBox cible locale $\mathcal{T}_t$ respectivement, R_{st} est la relation de pont entre $\boldsymbol{T}_s$ et $\boldsymbol{T}_t$, tel qu'il créé les mappings d'un sous-ensemble de noeuds dans $\boldsymbol{T}_s$ à un sous-ensemble d'axiomes dans $\mathcal{T}_t$. Pour un noeud ouvert $x \in \boldsymbol{T}_s$ et un axiome $A_k \in \mathcal{T}_t$, $(x, A_k) \in r_{ij}$ existe au moins une règle de pont complète entre eux.*

Parce que dans ce cas, l'algorithme de tableaux distribué se réalise sur l'ensemble des règles

de pont complet, D**T** doit satisfaire cinq conditions T1-T5 dans la section 2.4.2 et la condition suivante :

- T6 : Si $G \in \mathcal{L}(x), i : G \xrightarrow{\equiv} j : G \in \mathbb{M}_{ij}, G \in Ex(j : A_k)$, et $Tab_i(\mathcal{L}(x)) =$ insatisfaisable alors $\mathcal{L}(x) \longrightarrow \mathcal{L}(x) \cup \{C_{A_k}\}$

où $\mathbb{M}_{ij}$ est un ensemble de règles de pont complètes.

6.3.1 Algorithme

Une procédure de décision basée sur un tableau pour $\mathfrak{T} \models_d C$ est décrite comme suit. Au commencement nous mettons en application l'algorithme de tableaux $\mathbf{T}_1(C)$ ou $\mathbf{T}_2(C)$ sur la TBox $\mathcal{T}_1$ ou la TBox $\mathcal{T}_2$ respectivement. Ceci signifie que nous pouvons initialement traiter une requête posée sur une certaine TBox. Si nous commençons sur $\mathcal{T}_1$, alors $\mathcal{T}_1$ est vue comme la TBox cible et $\mathcal{T}_2$ est la TBox source, et inversement. Donc, pour pouvoir détecter rapidement les contradictions locales, nous exécutons initialement sur les deux TBox. Nous obtiendrons les deux cas suivants :

1. SI $\mathbf{T}_1(C)$ ou $\mathbf{T}_2(C)$ est insatisfaisable (c.-à-d., tous les noeuds de feuille de $\mathbf{T}_1(C)$ ou de $\mathbf{T}_2(C)$ sont les contradictions) alors nous concluons que C est également insatisfaisable par rapport à $\mathcal{T}$.
2. SINON (tous les $\mathbf{T}_1(C)$ et $\mathbf{T}_2(C)$ sont satisfaisables), c'est à dire qu'il existe au moins un noeud feuille non contradictoire de $\mathbf{T}_1(C)$ et un de $\mathbf{T}_2(C)$), alors nous continuons d'appliquer l'algorithme de tableaux sur $\mathcal{T}_2$ pour les noeuds de feuille ouverts de $\mathbf{T}_1(C)$ en utilisant les règles de pont identiques, c.-à.-d., $\mathbf{T}_2(\mathbf{T}_1(C))$ est appliqué seulement avec des noeuds feuille non-contradictoires de $\mathbf{T}_1(C)$. Quelques axiomes de $\mathcal{T}_2$ influencés par des règles de pont entre $\mathcal{T}_1$ et $\mathcal{T}_2$ sont ajoutés au système de contrainte pour $\mathbf{T}_2(\mathbf{T}_1(C))$.

6.3.2 Tableau distribué pour la TBox-décomposante

Afin de construire un modèle distribué pour une ontologie de décomposition, nous commençons par une liste de noeuds d'ABox initiale qui correspond à une ontologie locale. Des nouveaux faits peuvent être ajoutés à la forêt d'ABox en appliquant des règles d'expansion de tableau. Tout d'abord, des noeuds seront produits dans l'arbre d'ABox A_j cible en appliquant des règles d'expansion de tableau traditionnelles. Des nouveaux faits dans les noeuds ouverts de l'arbre d'ABox cible devraient alors être envoyés à l'ABox source A_i, c.-à-d., un fait $C(x)$ est produit dans un noeud ouvert $\mathcal{L}(x)$, $\mathcal{L}(x)$ sera envoyé à l'ABox source s'il y a une règle de pont $i : C \xrightarrow{\equiv} j : C$, donc $C(x)$ et $\neg C(x)$ peuvent être produits dans l'arbre d'ABox source. Toutes les contradictions peuvent être localement détectées. Notons qu'A_i soit construit de $\mathcal{L}(x)$ et que les

Entrée : Le concept C et la TBox-décomposante $\langle \mathcal{T}_{ij}, \mathcal{B}_{ij} \rangle$

Sortie : Satisfaisable /Insatisfaisable

PROCEDURE SAT-DIST($C, \langle \mathcal{T}_{ij}, \mathcal{B}_{ij} \rangle$)

1 : BEGIN

2 : $\mathbf{T}_1 = Tab_1(C)$;

3 : $\mathbf{T}_2 = Tab_2(C)$;

4 : **If** ($\mathbf{T}_1$ n'est pas contradictoire) et ($\mathbf{T}_2$ n'est pas contradictoire) **then**

5 : $\mathbf{T} = Tab_2(C)$ {exécute le raisonnement local dans $\mathcal{T}_2$ et génère l'arbre complet}

6 : **for each** branche ouvert β dans **T do**

7 : **repeat**

8 : sélectionne noeud $x \in \beta$;

9 : $\mathbb{I}_1^{idt}(x) = \{D | 1 : D \stackrel{\equiv}{\rightarrow} 2 : D, D \in L(x)\}$

10 : **if** ($\mathbb{I}_1^{idt}(x) \neq \emptyset$) **then**

11 : **repeat**

12 : sélectionne $D \in \mathbb{I}_1^{idt}(x)$

13 : **if** $D \in Ex(2 : A_k)$ **then**

14 : $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{Ex(2 : A_k)\}$

15 : **end if**

16 : **until** il n'existe pas d'élément non sélectionné dans $\mathbb{I}_1^{idt}(x)$

17 : **if** ($\mathbf{dTab}_1(\mathcal{L}(x))$) n'est pas satisfaisable **then**

18 : ferme {contradictoire dans x}

19 : break ; {vérifier la branche suivante}

20 : **end if**

21 : **end if**

22 : **until** ((β est ouvert) et (il existe des noeuds qui n'ont pas vérifiés dans β))

23 : **end for** {toutes les branches sont vérifiés}

24 : **end If**

25 : **if** (**T** est contradictoire) ou ($\mathbf{T}_1$ est contradictoire) ou ($\mathbf{T}_2$ est contradictoire) **then**

26 : retourne insatisfaisable ;

27 : **else**

28 : retourne satisfaisable ;

29 : **end if**

30 : END.

Fig. 6.3 : Algorithme de tableaux distribué

axiomes contiennent les règles de pont utilisées. Ainsi, la connaissance propagée dans ce cas est juste les axiomes, et ce processus est appelé la *propagation d'axiome*.

Théorème 6.3. *(Terminaison) Pour toute TBox-décomposante $\mathfrak{T}$ et pour tout $\mathcal{ALC}$ concept D, le tableau distribué pour vérifier le satisfaisabilité de D par rapport à $\mathfrak{T}$ s'arrête.*

Preuve : Puisque les faits sont seulement envoyés de l'ABox cible à l'ABox source en employant les règles de pont complètes, il n'y a aucun cycle de message entre les arbres d'ABox et la terminaison de l'algorithme est encore assurée en utilisant la stratégie de blocage.

□

Exemple 6.2 :

Pour une TBox-décomposante $\mathfrak{T} = \langle\{\mathcal{T}_1, \mathcal{T}_2\}, \mathcal{B}_{12}\rangle$, supposons que $\mathcal{T}_1$ contienne un axiome $C \sqcap \exists R.D \sqsubseteq E$, $\mathcal{T}_2$ contienne des axiomes $A \sqsubseteq B$ et $\forall R.B \sqsubseteq C$, et que $\mathcal{B}_{12}$ contienne deux règles de pont $1 : C \xrightarrow{\equiv} 2 : C$ et $1 : R \xrightarrow{\equiv} 2 : R$. Une requête est proposée pour vérifier que le concept C subsume la description de concept $\forall R.A \sqcap D$ ($\forall R.A \sqcap D$ est subsumé par concept C) par rapport à $\mathfrak{T}$.

Ce problème peut être transformé en vérifiant la satisfaisabilité de $((\forall R.A \sqcap D) \sqcap \neg C)$, voir la Figure 6.4. Dans cet exemple, nous nous arrêtons au Tab_2 parce que toutes les branches dans $\mathbf{T}_2$ sont fermées (tous les noeuds de feuille d'arbre d'accomplissement local $\mathbf{T}_2$ sont contradictoires), alors $((\forall R.A \sqcap D) \sqcap \neg C)$ est insatisfaisable par rapport à $\mathcal{T}_2$, c.-à-d., $\mathcal{T}_2 \models \forall R.A \sqcap D \sqsubseteq C$ ou $\mathcal{T}_{12} \models_d \forall R.A \sqcap D \sqsubseteq C$. Nous adresserons un autre exemple :

Exemple 6.3 :

Soit une TBox-décomposante $\mathfrak{T} = \langle\{\mathcal{T}_1, \mathcal{T}_2\}, \mathcal{B}\rangle$, où

$\mathcal{T}_1 = \{\neg TerrestrialAnimal \sqsubseteq Animal, AquaticAnimal \sqsubseteq \neg TerrestrialAnimal\}$,

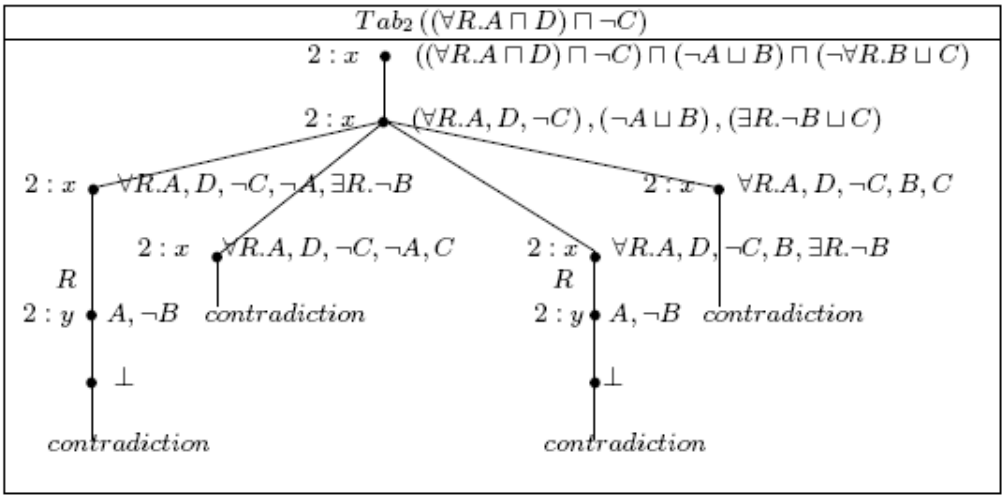


Fig. 6.4 : Exemple de tableau distribué s'arrêtant dans $\mathbf{T}_2$

$\mathcal{T}_2 = \{Fish \sqsubseteq AquaticAnimal\}$,

$\mathcal{B}_{12} = \{1 : AquaticAnimal \stackrel{\equiv}{\rightarrow} 2 : AquaticAnimal\}$.

Une requête est proposée pour vérifier que *Animal* subsume $Fish$ ($Fish$ est subsumé par *Animal*) par rapport à $\mathfrak{T}$.

Une représentation brève pour les notations du concept et du rôle sera facile de suivre l'illustration de l'algorithme comme dans la Figure 6.5, où $\mathcal{T}_1 = \{\neg B_1 \sqsubseteq A_1, A_2 \sqsubseteq \neg B_1\}$, $\mathcal{T}_2 = \{B_2 \sqsubseteq A_2\}$, et $\mathcal{B}_{12} = \{1 : A_2 \stackrel{\equiv}{\rightarrow} 2 : A_2\}$. Nous voulons vérifier que A_1 subsume B_2 (B_2 est subsumé par A_1) par rapport à $\mathfrak{T}$. Ce problème peut être transformé en vérifiant la satisfaisabilité de $B_2 \sqcap \neg A_1$, voir la Figure 6.5.

Dans cet exemple, toutes les branches ouvertes de Tab_2 sont passées à la TBox $\mathcal{T}_1$ par les règles de pont complètes et Tab_1 est appliqué. Toutes les branches dans Tab_1 sont fermées, donc $(B_2 \sqcap \neg A_1)$ est insatisfaisable p.r.à $\mathcal{T}_{12}$, autrement dit $\mathcal{T}_{12} \models_d B_2 \sqsubseteq A_1$.

Théorème 6.4. *(Complétude et correction) Un $\mathcal{ALC}$-concept D est satisfaisable par rapport à une TBox-décomposante $\mathfrak{T}$ de $\mathcal{T}$ si et seulement si la construction de tableau distribué pour D peut générer un arbre complet et non-contradictoire.*

Preuve : Complétude et correction de l'algorithme résultent de l'observation que les expansions $\mathcal{ALC}$ pour la TBox-décomposante enverront des faits du concept d'un noeud ouvert à

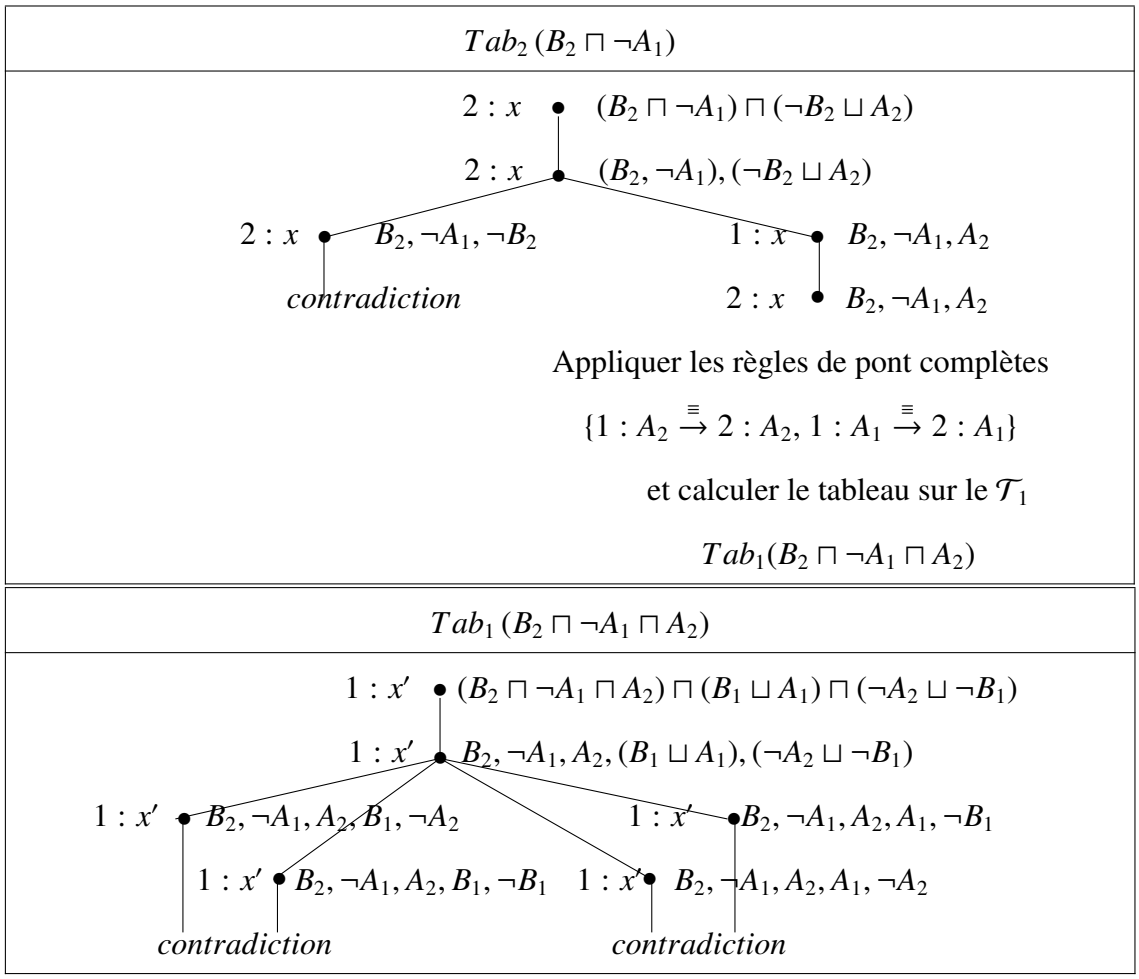


FIG. 6.5 : Exemple de tableau distribué s'arrêtant dans $\mathbf{T}_1$

l'ABox source, et l'inconsistance est détectée quand les deux $C(x)$ et $\neg C(x)$ se produisent dans une ABox locale. L'inconsistance dans l'ABox distribuée doit nécessairement résulter en une inconsistance d'une ABox locale ; et une inconsistance dans une ABox locale impliquera que l'ABox distribuée est aussi inconsistante.

□

Théorème 6.5. *Soit une TBox $\mathcal{T}$ et sa TBox-décomposante présentée comme une TBox distribuée $\mathcal{T}_{12} = \langle\{\mathcal{T}_i\}_{i\in\{1,2\}}, \mathcal{B}_{ij}\rangle$, alors*

$$\mathcal{T}_{12} \models_d i : X \sqsubseteq j : Y \Leftrightarrow \mathcal{T} \models X \sqsubseteq Y,$$

où X, Y sont des concepts dans $\mathcal{T}$.

Preuve : $(\Longrightarrow)\ \mathcal{T}_{12} \models_d i : X \sqsubseteq j : Y \Rightarrow \mathcal{T} \models X \sqsubseteq Y$

Soit $\mathcal{I}$ une interprétation satisfaisant $\mathcal{T}$ ($\mathcal{I} \models \mathcal{T}$), avec le domaine $\Delta^{\mathcal{I}}$, nous prouvons que $\mathcal{I} \models X \sqsubseteq Y$ (ou $X^{\mathcal{I}} \subseteq Y^{\mathcal{I}}$). En effet,

1. À partir de $\mathcal{I}$, nous définissons $\Im = \langle\{\mathcal{I}_i\}, \{r_{ij}\}\rangle$, où :
   - $\mathcal{I}_i$ est une interprétation par rapport au domaine $\Delta^{\mathcal{I}_i}$ et $\Delta^{\mathcal{I}_i} = \Delta^{\mathcal{I}}$. $\mathcal{I}, \mathcal{I}_i$ et $\Im$ sont dans le même domaine (pour tout $i \in I$). Ainsi, $\mathcal{I}_i$ interprète chaque concept primitif, rôle primitif(ou concepts spéciaux Top, Bot) (i, C) de $\mathcal{T}_i$ par la règle $\#(C)^{\mathcal{I}_i} = C^{\mathcal{I}}$.
   - $\{r_{ij}\}$ sont des relations identiques, c.-à-d., $r_{ij}(C^{\mathcal{I}_i}) \equiv C^{\mathcal{I}_j}$, pour toutes règles de pont identiques entre $\mathcal{T}_i$ et $\mathcal{T}_j$.

   D'abord, nous devons prouver que $\mathcal{I}_i$ et $\Im$ sont des interprétations de $\mathcal{T}_i$ et $\mathcal{T}_{12}$ respectivement. Nous pouvons facilement montrer que $(\Delta_i , \cdot^{\mathcal{I}_i})$ est une interprétation de $\mathcal{T}_i$, parce que :

$$\Delta^{\mathcal{I}_i} \text{ n'est pas vide} \tag{6.1}$$
$$Top^{\mathcal{I}_i} = \Delta^{\mathcal{I}_i} \tag{6.2}$$
$$Bot^{\mathcal{I}_i} = \emptyset \tag{6.3}$$
$$M^{\mathcal{I}_i} \subseteq \Delta^{\mathcal{I}_i} \text{ pour chaque concept } M \text{ de } \mathcal{T}_i \tag{6.4}$$
$$R^{\mathcal{I}_i} \subseteq \Delta^{\mathcal{I}_i} \times \Delta^{\mathcal{I}_i} \text{ pour tous les rôles } R \text{ de } \mathcal{T}_i \tag{6.5}$$

   Selon la décomposition overlay, pour chaque concept défini (rôle) M de $\mathcal{T}$, il existe un concept (rôle) défini $\#M$ dans $\mathcal{T}_i (i \in I)$ et $\#(M)^{\mathcal{I}_i} = M^{\mathcal{I}}$ (parce que $\mathcal{I}_i$ et $\mathcal{I}$ sont dans le même domaine).

   De plus, nous avons $\#(M)^{\mathcal{I}_i} = M^{\mathcal{I}}$, pour chaque concept (rôle) arbitraire M dans $\mathcal{T}_i$ (d'après la proposition 5.14 dans la section 5.2.3). En conséquence,

$$\#(M)^{\mathcal{I}_i} \subseteq \#(N)^{\mathcal{I}_j} \Leftrightarrow M^{\mathcal{I}} \subseteq N^{\mathcal{I}}$$

   Pour chaque axiome $(i, V) \sqsubseteq (i, W)$ of $\mathcal{T}_i$, par la décomposition overlay, il existe un axiome $V \sqsubseteq W$ de $\mathcal{T}$.

   Puisque $\mathcal{I}$ satisfait $\mathcal{T}$ et $(V \sqsubseteq W) \in \mathcal{T}$, alors $\mathcal{I} \models V \sqsubseteq W \Longrightarrow V^{\mathcal{I}} \subseteq W^{\mathcal{I}}$. D'autre part, $\#(M)^{\mathcal{I}_i} = \#(N)^{\mathcal{I}_i} \Leftrightarrow M^{\mathcal{I}} = N^{\mathcal{I}}$, donc

$$\begin{aligned} V^{\mathcal{I}} \subseteq W^{\mathcal{I}} \Rightarrow \#(V)^{\mathcal{I}_i} \subseteq \#(W)^{\mathcal{I}_i} \Leftrightarrow \mathcal{I}_i \models (i, V) \sqsubseteq (i, W) \\ \Rightarrow \Im \models_d i : V \sqsubseteq W \end{aligned} \tag{6.6}$$

   >De (6.1)–(6.6) ci-dessus, nous obtenons

$$\mathcal{I}_i \models \mathcal{T}_i \Longrightarrow \Im \models_d \mathcal{T}_i \tag{6.7}$$

Pour chaque règle de pont identique $i : V \underset{\rightarrow}{\equiv} j : V$. Car $V^{\mathcal{I}} \equiv \#(V)^{\mathcal{I}_i}$ et $V^{\mathcal{I}} \equiv \#(V)^{\mathcal{I}_j} \Longrightarrow \#(V)^{\mathcal{I}_i} \equiv \#(V)^{\mathcal{I}_j}$

$$\Longrightarrow \Im \models i : V \underset{\rightarrow}{\equiv} j : V \tag{6.8}$$

>De (6.7) et (6.8), nous avons $\Im \models \mathcal{T}_{12}$

2. Maintenant, par l'hypothèse et $M^{\mathcal{I}} = \#(M)^{\mathcal{I}_i}$ nous déduisons que

$$\begin{aligned} \Im \models i : X \sqsubseteq j : Y &\Longrightarrow \#(X)^{\mathcal{I}_i} \subseteq \#(Y)^{\mathcal{I}_j} \\ &\Longrightarrow X^{\mathcal{I}} \subseteq Y^{\mathcal{I}} \\ &\Longrightarrow \mathcal{I} \models X \sqsubseteq Y \end{aligned}$$

($\Longleftarrow$) $\mathcal{T} \models X \sqsubseteq Y \Longrightarrow \mathcal{T}_{12} \models i : X \sqsubseteq j : Y$

Soit $\Im = \langle \{\mathcal{I}_i\}, \{r_{ij}\} \rangle$ une d-interprétation satisfaisant $\mathcal{T}_{12}$, où $\mathcal{I}_i$ est une interprétation de $\mathcal{T}_i$, $\{\mathcal{T}_i\}_{i \in I}$ sont dans le même domaine, $\{r_{ij}\}_{i \neq j}$ sont des relations identiques entre deux domaines $\Delta^{\mathcal{I}_i}$ et $\Delta^{\mathcal{I}_j}$. Il devrait être montré que

$$\Im \models i : X \sqsubseteq j : Y((X)^{\mathcal{I}_i} \subseteq \#(Y)^{\mathcal{I}_j})$$

Nous définissons une interprétation $\mathcal{I}$, avec le domaine $\Delta \equiv \Delta^{\mathcal{I}_i}, \forall i \in I$, qui interprète un concept (rôle) primitif C de $\mathcal{T}$ en utilisant le rôle $C^{\mathcal{I}} = \#(C)^{\mathcal{I}_i}$.

Nous devons prouver que $\mathcal{I} \models \mathcal{T}$. En conséquent, pour chaque concept (rôle) arbitraire M de $\mathcal{T}$, il existe un concept (rôle) correspondent $\#M$ in $\mathcal{T}_i$ et nous avons

$$M^{\mathcal{I}} = \#(M)^{\mathcal{I}_i} \text{(d'après la preuve ci-dessus)} \tag{6.9}$$

Pour chaque axiome $V \sqsubseteq W$ de $\mathcal{T}$, par la décomposition overlay, il existe un axiome équivalent $\#V \sqsubseteq \#W$ dans $\mathcal{T}_i$

Parce que $\Im \models_d T_{12} \Longrightarrow \Im \models_d T_i, \forall i \in I \Longrightarrow \mathcal{I}_i \models \mathcal{T}_i$

À partir de $\#V \sqsubseteq \#W \in \mathcal{T}_i$, nous avons $\mathcal{I}_i \models \#V \sqsubseteq \#W \Longrightarrow \#(V)^{\mathcal{I}_i} \subseteq \#(W)^{\mathcal{I}_i} \Longrightarrow V^{\mathcal{I}} \subseteq W^{\mathcal{I}}$

$$\Longrightarrow \mathcal{I} \models V \sqsubseteq W \tag{6.10}$$

>De (6.9) et (6.10), $\Longrightarrow \mathcal{I} \models \mathcal{T}$

Autrement, de $\mathcal{T} \models X \sqsubseteq Y$ (par l'hypothèse) $\Longrightarrow \mathcal{I} \models X \sqsubseteq Y \Longrightarrow X^{\mathcal{I}} \subseteq Y^{\mathcal{I}} \Longrightarrow \#(X)^{\mathcal{I}_i} \subseteq \#(Y)^{\mathcal{I}_j} \Longrightarrow \Im \models i : X \sqsubseteq j : Y$

□

6.4 Impact de la décomposition sur la complexité

Dans cette section, nous voulons donner une analyse simple pour la complexité des algorithmes de satisfaisabilité parallèle et distribué en calculant le temps réalisé par ces deux algorithmes.

Soient $\mathcal{T}$ une TBox et sa TBox-décomposante $\{\mathcal{T}_i, \mathcal{B}_{ij}\}_{i,j\in\{1,2\}}$. Soient l le nombre des concepts et rôles de $Ex(\mathbf{A})$, $m_i = |\mathcal{T}_i|$ le nombre d'axiomes de $\mathcal{T}_i$, a la longueur maximale des axiomes dans $\mathcal{T}$ (le nombre maximal des symboles d'un axiome dans $\mathcal{T}$), et $k = |\mathcal{B}_{ij}|$ le nombre des règles de pont entre $\mathcal{T}_i$ et $\mathcal{T}_j$. Soient p_i le nombre des noeuds ouverts dans le tableau sur $\mathcal{T}_i$ et p_j le nombre des noeuds ouverts dans le tableau sur $\mathcal{T}_j$.

Lemme 6.6. *Les temps pris par la procédure parallèle et la procédure distribuée pour calculer SAT par rapport à une TBox-décomposante sont :*

- *Temps* $(n, k, m_i, p_i, p_j) = O(f_{sat}(m_i) + (k.p_j)^{p_i} + p_i.p_j.f_{fus}(n))$ *, pour la procédure parallèle*
- *Temps* $(m_i, m_j, k, p_i) = O(f_{sat}(m_i) + k.p_i + p_i.f_{sat}(m_j))$ *, pour la procédure distribuée*

où f_{sat} est la fonction pour calculer le temps de la satisfaisabilité, f_{fus} est pour le fusionnement.

Notons que nous ne donnons pas les calculs particuliers de la complexité. Parce que, dans la réalité, la complexité dépend non seulement du nombre d'axiomes mais aussi de l'expressivité du langage de LD utilisé (i.e., les constructeurs dans les axiomes). Dans le but de spécifier des paramètres qui influencent le temps du calcul de satisfaisabilité sur la TBox-décomposante, nous supposons que la complexité de SAT par rapport à la TBox $\mathcal{T}$ ne dépend que du nombre d'axiomes de $\mathcal{T}$. Ceci est la base qui donne des critères de décomposition (dans le chapitre suivant). **Preuve :** Si $O(f_{sat}(m))$ est le temps nécessaire pour une vérification de satisfaisabilité par rapport à une TBox avec m axiomes est, alors $O(f_{sat}(m_i))$ est le temps pour le même problème par rapport à une sous-TBox avec m_i axiomes.

1. *Pour la procédure parallèle* : Dans le pire cas, cette procédure réalise trois phases principales :
   - Vérifier la SAT d'un concept C en parallèle sur deux sous-TBox $\mathcal{T}_i$ et $\mathcal{T}_j$: le temps du calcul de SAT est $max\{O(f_{sat}(m_i))|i \in \{1, 2\}\} = O(f_{sat}(m_i))$.
   - Pour chaque noeud ouvert de $\mathbf{T}_i$ (p_i noeuds), il faut inspecter la présence des éléments de $Ex(\mathcal{B}_{ij})$ ($k = |Ex(\mathcal{B}_{ij})|$ fois pour k règles de pont), pour chaque noeud qui contient des éléments de $Ex(\mathcal{B}_{ij})$ alors vérifier pour tous les noeuds ouverts de $\mathbf{T}_j$ (p_j noeuds). Il faut inspecter $(k.p_j)^{p_i}$ fois, donc le temps nécessaire sur cette phase est $O((k.p_j)^{p_i})$.
   - Un fusionnement est exécuté sur des noeuds ouverts de $\mathbf{T}_i$ et $\mathbf{T}_j$ s'ils contiennent au moins un élément de $Ex(\mathcal{B}_{ij})$. Donc, dans le pire cas, il faut fusionner $(p_i.p_j)$ fois. De

plus, le cardinal de l'ensemble étiqueté des concepts et des rôles sur chaque noeud est toujours inférieur ou égal à l, le temps du calcul pour chaque fusionnement est $O(f_{fus}(n))$. Enfin, cette phase est réalisée en temps $O(p_i.p_i.f_{fus}(n))$.

Le temps dans le pire cas utilisé par l'algorithme parallèle est :

Temps $(n, k, m_i, p_i, p_j) = O(f_{sat}(m_i) + (k.p_j)^{p_i} + p_i.p_j.f_{fus}(n))$

2. *Pour la procédure distribuée* : Cette procédure se compose de trois phases principales :
   - D'abord, la vérification de SAT d'un concept C est réalisée sur une sous-TBox $\mathcal{T}_i$, donc le temps du calcul est $f_{sat}(m_i)$.
   - Pour chaque noeud ouvert dans $\mathbf{T}_i$, nous inspectons la présence des éléments de $Ex(\mathcal{B}_{ij})$ ($k = |Ex(\mathcal{B}_{ij})|$ fois pour k règles de pont). Le temps nécessaire sur cette phase est $O(k.p_i)$.
   - Pour chaque noeud ouvert de $\mathbf{T}_i$ qui contient des éléments de $Ex(\mathcal{B}_{ij})$, la vérification de SAT sera poursuivie sur $\mathcal{T}_j (i \neq j)$ pour ce noeud et avec seulement des "axiomes relatifs" de $\mathcal{T}_j$. Un axiome A dans $\mathcal{T}_j$ est appelé "relatif" à la règle $i : C \xrightarrow{\equiv} j : C$ si $C \in Ex(A)$. On peut avoir plusieurs axiomes qui sont relatifs à une même règle, le nombre des axiomes relatifs à une règle de pont est toujours inférieur ou égal à m_j. Une borne supérieure pour la vérification de SAT sur $\mathcal{T}_j$ est $O(f_{sat}(m_j))$. Dans le pire cas, la vérification de SAT sur $\mathcal{T}_j$ est réalisée p_i fois, donc le temps total du calcul sur cette phase est $O(p_i.f_{sat}(m_j))$.

   Dans le pire des cas, le temps utilisé par l'algorithme distribué est :

   Temps $(m_i, m_j, k, p_i) = O(f_{sat}(m_i) + k.p_i + p_i.f_{sat}(m_j))$

□

À partir de ces calculs, nous trouvons que les éléments de décomposition qui influencent la complexité du raisonnement sont : m_i, m_j et k. Le meilleur cas est $k = 0$ et $m_i \approx m_j$. En effet, si $k = 0$, nous n'avons pas besoin de calculer les fusionnements dans le cas parallèle, ni les vérifications de SAT sur le $\mathcal{T}_j$ dans le cas distribué, et ni les inspections de la présence des règles de pont. Avec $m_i \approx m_j$, alors $max(f_{sat}(m_i), f_{sat}(m_j))$ est minimal. Le temps de calcul dans ce cas est $O(f_{sat}(m_i))$, c'est beaucoup mieux que $O(f_{sat}(m))$ - le temps pour la vérification de SAT sur $\mathcal{T}$.

6.5 Discussion et conclusion

Cette section discutera d'une part de la relation entre la décomposition de l'ontologie et la logique de la description distribuée, et d'autre part des techniques de raisonnement parallèle et distribué sur cette décomposition.

6.5.1 Décomposition de l'ontologie et logique de description distribuée

Nous savons que les sources qui causent la complexité du raisonnement sont deux explosions combinatoires de la ramification-AND et de la ramification-OR [Fra03]. Chaque axiome (inclusions de concept générales (ICGs)) est une des raisons principales qui peut amener à une ramification-OR. Notre but est d'éliminer tous les ICGs possibles d'une ontologie générale (représentée par une TBox) en décomposant l'ensemble des ICGs de cette ontologie en plusieurs sous-ensembles d'ICGs (représentés par une TBox distribuée). Dans le cas de LD générale, notre but vise à réduire le temps de calcul du raisonnement tandis que les résultats du raisonnement dans la décomposition sont préservés de TBox originelle. La décomposition s'applique seulement à l'ensemble des axiomes (ICGs). Tous les concepts et rôles de l'ontologie originelle seront conservés par la décomposition.

Une LDD est une extension d'une LD à partir d'un des défis dans le web sémantique qui peut résoudre le problème du raisonnement avec un grand nombre d'ontologies locales hétérogènes et chevauchantes, dans lesquelles chaque ontologie décrit un domaine de l'application d'une perspective locale et subjective. Par exemple, pour décrire des gens, l'ontologie O_1 utilise le concept "PERSONNE" tandis que l'ontologie O_2 utilise le concept "COUPLE". Il y a des relations claires entre ces deux ontologies, par exemple, si deux personnes dans l'ontologie O_1 ont un rapport au mariage, alors ils forment un couple dans l'ontologie O_2. Ces corrélations sont représentées par les mappings sémantiques (règles de pont). Les ontologies locales, cependant, sont indépendantes, autonomes, et interprètent des domaines différents. Par conséquent, le raisonnement et le traitement de la requête sont seulement mis en application localement, c.-à-d., une requête peut être traitée sur une ontologie locale et la connaissance peut être propagée à d'autres ontologies locales par les mappings sémantiques. Le raisonnement dans une LDD est basé essentiellement sur une ontologie globale qui est combinée à toutes les ontologies locales et les mappings sémantiques entre eux [BS03, ST04b]. En fait, cette approche rencontre quelques inconvénients comme mentionné dans [ST04a]. Premièrement, le coût dans l'espace pour raisonner avec l'ontologie globale est beaucoup plus grand que la somme de coûts de l'espace pour raisonner avec les ontologies locales. Donc une combinaison appropriée des ontologies locales est nécessaire pour le raisonnement global. Deuxièmement, le raisonnement global perd l'autonomie des ontologies locales, c.-à-d., le raisonnement dans les ontologies locales peut être fait par les raisonneurs spécifiques qui sont optimisés pour des langages locaux, tandis que le raisonnement dans l'ontologie globale doit être exécuté par le raisonneur le plus général qui est capable de traiter avec un langage local plus général. Troisièmement, dans certains cas la combinaison des ontologies locales comme dans une ontologie globale n'est pas disponible, ceci peut mener à une inconsistance dans l'ontologie globale ou limiter l'accès aux ontologies locales.

Notre approche est une décomposition de l'ontologie, à partir de l'idée opposée où nous voulons diviser une grande ontologie en de multiples petites ontologies et raisonner dans les systèmes de ces ontologies. Nous avons choisi de travailler avec la décomposition de l'ontologie parce qu'elle fournit le degré le plus élevé de découplage entre des ontologies différentes. C'est important pour notre objectif d'optimisation du raisonnement, car la décomposition nous permet de réduire le nombre d'axiomes de l'ontologie originelle en distribuant les axiomes dans les sous-ontologies, et le raisonnement est exécuté sur ces ontologies. Les ontologies locales (sous-ontologies) dans ce cas sont dans le même domaine avec l'ontologie originelle puisqu'ils partagent l'ensemble de concepts et de rôles de l'ontologie originelle. La décomposition de l'ontologie peut être présentée comme un cas particulier de LDD, où ses mappings sémantiques sont seulement des relations d'identité entre les concepts (rôles) qui apparaissent ensemble dans deux axiomes des ontologies différentes. Les résultats les plus importants sont la conservation des sémantiques et des services d'inférence entre l'ontologie originelle et l'ontologie de décomposition. En outre, les requêtes peuvent être traitées simultanément et renvoyer les résultats sur des ontologies locales. Une requête peut être également décomposée en des sous-requêtes qui sont alors traitées séparément sur les ontologies locales, et les résultats sont finalement combinés. Quand une requête est donnée, un ensemble de règles de pont (extraites à partir des concepts et des rôles qui apparaissent dans l'expression de requête) sera ajouté dans les règles de pont complètes. C'est pour soutenir la propagation de la connaissance des autres ontologies, c.-à-d., en fait, si quelques concepts (rôles) sont dans l'expression de requête qui n'étaient pas présents dans les règles de pont, et le raisonnement ne sera pas adéquat si nous manquons de la connaissance qui doit être transférée dans ces règles de pont supplémentaires. Ainsi le raisonnement dans des TBox source est seulement mis en application avec les axiomes qui contribuent à la présence des règles complètes de pont.

À partir des caractéristiques de la décomposition d'ontologie, nous avons proposé quelques techniques de raisonnement comme mentionné dans la section ci-dessus.

6.5.2 Raisonnement parallèle et distribué

Dans le raisonnement parallèle, des tableaux locaux sont créés simultanément sur des ontologies locales, des noeuds ouverts dans des tableaux locaux sont alors fusionnés et combinés pour trouver les résultats. La fusion est faite par l'ensemble de règles de pont complètes. Cependant, le coût pour fusionner est élevé s'il y a un grand nombre de noeuds ouverts dans les ontologies locales. Des questions peuvent être également décomposées en sous-requêtes et sont réalisées en parallèle sur des ontologies locales, avec des réponses de requête qui sont combinées plus tard.

Le raisonnement distribué réduit au minimum le nombre d'axiomes (ICGs) utilisés. Les résultats sont obtenus sur des ontologies locales. Cette approche convient à des plus grandes ontologies qui peuvent être divisées en des sous-ontologies disjointes, parce que les règles de propagation sont appliquées le moins possible. De plus, les algorithmes de raisonnement sur les ontologies locales peuvent appliquer les techniques d'optimisation pour l'algorithme de tableaux classique (comme les techniques présentées dans le chapitre 3).

Cependant, pour une ontologie dont les ontologies composantes sont complètement disjointes (l'ensemble de règles de pont est vide), le processus de raisonnement s'arête sur les ontologies locales sans phase de fusion (comme dans le cas parallèle) et sans propagation d'axiome (comme dans le cas distribué).

Pour ces deux algorithmes du raisonnement, le temps de calcul dépend de la taille de composante maximale (le nombre des symboles de la TBox composante de la taille maximale) et la taille de la partie commune (le nombre des symboles communs entre les TBox composantes).

Troisième partie

Méthodes de décomposition

Les travaux de recherche présentés dans la partie précédente ont été réalisés avec la supposition de l'existence d'une décomposition d'une ontologie (d'une TBox $\mathcal{T}$) qui est appelée décomposition overlay. Les résultats cruciaux obtenus sur cette décomposition se composent des préservations de sémantique et de mécanisme d'inférence de la TBox originelle. Nous avons proposé deux nouvelles techniques de raisonnement et avons montré leur efficacité du raisonnement pour la TBox décomposante dans les cas où la décomposition est optimale selon les conditions de la définition 5.8. Dans cette partie, nous cherchons la réponse à une nouvelle question : ***"Comment obtenir une telle décomposition overlay avec de bonnes propriétés ?"*** *Pour répondre à cette question, nous adresserons le problème crucial de la décomposition automatique d'une ontologie donnée. Cette démarche est considérée comme une étape d'optimisation dans la conception de l'ontologie qui est aidée par la théorie des graphes. La théorie des graphes a fourni les "belles propriétés" qui adaptent les exigences nécessaires de notre décomposition. Concrètement, les analyses de coupe-sommet (séparateur minimal) dans la méthode arborescente et de coupe-arête (basée sur la coupe normalisée) dans la méthode de calcul de vecteurs propres traduisent ces propriétés. Le premier chapitre (chapitre 7) de cette partie introduira les notions principales de la théorie des graphes et de la théorie spectrale des graphes ainsi que la méthode de décomposition arborescente. Le chapitre 8 présentera deux méthodes de décomposition basées sur le séparateur minimal et la coupe normalisée débutée avec les algorithmes ainsi que quelques résultats empiriques.*

Chapitre 7

Préliminaires de la théorie des graphes

Avant toute chose, ce chapitre introduit, en plus des quelques notions utiles de la théorie des graphes (plus détail dans [Jun99, Tod06, Maz04, Pri94]) et de la théorie spectrale des graphes [Chu97, Spi04], des outils pour aborder le problème de partition de graphe. Nous verrons deux heuristiques de calcul d'une décomposition en découpant récursivement le graphe à l'aide de ses séparateurs minimaux et de la coupe normalisée. Enfin, nous terminons le chapitre par quelques discussions et conclusions.

7.1 Graphes

Un *graphe* (un *graphe non orienté*) G est une paire $G = (V, E)$ qui se compose des ensembles finis $V \neq \emptyset$ et E. Les éléments de V sont appelés les *sommets* de G, et ceux de E sont appelés les *arêtes* de G. Chaque arête (x, y) de E est une paire non ordonnée des extrémités $x, y \in V$.

Pour chaque arête (x, y), nous disons que x et y sont des *adjacents* ou des *voisins* de l'un à l'autre. Alors nous pouvons dire que deux sommets x, y sont *adjacents* s'il y a une arête reliant x et y dans G. Un sommet est *isolé* s'il n'a pas de voisin.

Une *chaîne* μ de longueur k est une séquence de k arêtes $(e_1, e_2, ..., e_k)$, e_i a une extrémité commune avec $e_{i-1}, i > 1$ et une autre avec e_{i+1} si $i < k$. Une chaîne fermée est un *cycle*. Une *corde* d'une chaîne μ est une arête qui relie deux sommets non consécutifs sur μ.

Soit $G = (V, E)$ un graphe non orienté, avec $|V| = n, |E| = m$. Pour $x, y \in V$, l'ensemble de voisins de x est $N(x) = \{y \neq x | (x, y) \in E\}$. Pour $X \subseteq V$, $N(X) = \bigcup_{x \in X} N(x)$. Le *degré* d'un sommet est le cardinal de ses voisins.

Soit $G = (V, E)$ un graphe et V' un sous-ensemble de V. Nous dénotons $E|V'$ l'ensemble de toutes les arêtes e qui ont leurs deux sommets dans V'. Tout graphe $G' = (V', E')$, où $V' \subset V$ et $E' \subset E|V'$ est appelé un *sous-graphe (partiel)* de G.

Un *graphe orienté* $G = (V, U)$ est un graphe, où V est un ensemble de sommets, et U est

un ensemble de couples ordonnés de sommets, appelés *arcs*. Pour un arc (x, y), x est l'*extrémité initiale* (origine), y est l'*extrémité terminale* (extrémité).

Pour un arc (x, y), y est appelé *successeur* de x et x est appelé *prédécesseur* de y. On dit aussi que x, y sont *voisins* l'un de l'autre. L'ensemble des successeurs de x est noté $\Gamma(x)$, et celui des prédécesseurs, $\Gamma^{-1}(x)$.

Un *chemin* (path) μ de longueur k est une séquence de k arcs $(u_1, u_2, ..., u_k)$, avec l'extrémité initiale de u_i doit être l'extrémité terminale de u_{i-1} si $i > 1$, et l'extrémité terminale de u_i doit être l'extrémité initiale de u_{i+1} si $i < k$. Un chemin fermé est un *circuit*.

Le terme de *parcours* regroupe les chemins, les circuits, les chaînes et les cycles.

7.1.1 Quelques graphes particuliers

Un graphe est *complet* si toute paire de sommets est liée par un arc ou une arête. On dénote K_n est un graphe complet avec n sommets (c'est-à-dire, $|V| = n$).

Une *clique* d'un graphe G est un sous-graphe complet. Nous dénotons $G[S]$ - une clique qui est établie par des sommets de graphe G. Une *clique maximale* est une clique de G et elle est maximale par inclusion. On note $\omega(G)$ le nombre de sommets maximal d'une clique de G.

Le *graphe valué* (weighted graph) : à tout arc (resp. arête) est associée une valeur (par exemple : un poids, un coût, une distance, ...). On parle de fonction de valuation définie de $V \times V$ dans $\mathbb{R}$.

Un graphe $G = (V, E)$ est *connexe* si pour chaque paire u, v de V, il existe un chemin de u à v. Si G n'est pas connexe, alors un sous-graphe connexe C de G est appelé *composante connexe*, une composante connexe est maximale s'il n'existe pas de sous-graphe connexe C' de G tel que C est un sous-graphe approprié de C'. Un sommet isolé est considéré comme un composante connexe à lui seul.

Un graphe est *triangulé* si tout cycle de longueur plus de quatre possède une corde.

Un graphe connexe sans cycles est appelé *arbre*. Pour distinguer avec le graphe, nous notons T l'arbre et appelons *noeuds* les sommets et *branches* les arcs d'un arbre.

Une *arborescence* (rooted tree, directed tree) G est un arbre et tous les sommets sont descendants d'un sommet unique r, appelé *racine* de G. La figure 7.1 illustre un arbre et une arborescence.

7.2 Théorie spectrale des graphes

Dans cette section, nous présentons l'application de la décomposition de graphe dans la théorie spectrale des graphes (le détail peut être trouvé dans [Spi04, Chu97]). D'abord, nous

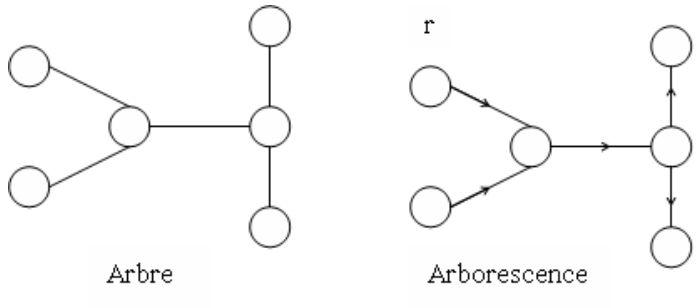


Fig. 7.1 : Un exemple d'arbre et d'arborescence

rappelons brièvement quelques notions de matrice qui seront utilisées dans les sections suivantes.

7.2.1 Vecteurs et matrices

$\mathbb{R}^{m\times n}$ dénote l'espace vectoriel de tous les matrices réelles $m-$par$-n$:

$$A \in \mathbb{R}^{m\times n} \iff A = [a_{ij}] = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}$$

La matrice de *transposition* ($\mathbb{R}^{m\times n} \to \mathbb{R}^{n\times m}$) de A est $C = A^T$, $c_{ij} = a_{ji}$. La matrice *identité*

$$I_n = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

Un ensemble des vecteurs $x_1, ..., x_p$ dans l'espace $\mathbb{R}^m$ est *orthogonal* si $x_i^T x_j = 0$, pour $i \neq j$. La matrice A est

diagonal	*si*	$a_{ij} = 0$, pour $i \neq j$
symétrique	*si*	$A^T = A$
définie positive	*si*	$x^T Ax > 0, 0 \neq x \in \mathbb{R}^n$
semi-définie positive	*si*	$x^T Ax \geq 0$, *pour tous* $x \in \mathbb{R}^n$
orthogonale	*si*	$A^T A = I_n$
positive	*si*	$a_{ij} > 0$, *pour tous* $i,\ j$
non-négative	*si*	$a_{ij} \geq 0$, *pour tous* $i,\ j$.

Les *valeurs propres* d'une matrice $A \in \mathbb{R}^{n\times n}$ sont les n racines de son *polynôme caractéristique* $p(z) = det(zI - A)$. L'ensemble de ces racines est appelé le *spectre* et dénoté par $\lambda(A)$. Si $\lambda(A) = \{\lambda_1, ..., \lambda_n\}$, alors $det(A) = \lambda_1\lambda_2...\lambda_n$.

De plus, si nous définissons la *trace* de A par

$$trace(A) = \sum_{i=1}^{n} a_{ii}, A \in \mathbb{R}^{n\times n}$$

alors $trace(A) = \sum_{i=1}^{n} \lambda_i$

Si $\lambda \in \lambda(A)$ alors les vecteurs non-zéro $x \in \mathbb{R}^n$ qui satisfait

$$Ax = \lambda x$$

sont appelés les *vecteurs propres*. Une valeur propre définit un sous-espace de dimension 1 qui est invariant par rapport à une pré-multiplication par A. Plus général, un sous-espace $S \subset \mathbb{R}^n$ avec la propriété :

$$x \in S \Rightarrow Ax \in S$$

est dit être *invariant* (pour A). Note que si

$$AX = XB, B \in \mathbb{R}^{k\times k}, X \in \mathbb{R}^{n\times k}$$

alors $R(X)$ est invariant et

$$By = \lambda y \Rightarrow A(Xy) = \lambda(Xy).$$

7.2.2 Matrices pour graphes

Sans perte de généralité, considérons le graphe $G = (V, E)$, avec $V = \{1, ..., n\}$. La matrice plus naturelle s'associe avec G et sa matrice d'adjacence, M_G, dont les entrées sont données par :

$$m_{ij} = \begin{cases} 1 & si\ (i,j) \in E \\ 0 & autrement \end{cases}$$

Nous nous intéressons à considérer la matrice laplacienne d'un graphe, L_G, dont les entrées sont données par :

$$l_{ij} = \begin{cases} -1 & si\ (i,j) \in E \\ de_i & si\ i = j, et \\ 0 & autrement \end{cases}$$

où de_i est le degré de sommet i.

Nous allons examiner la signification de *vecteurs propres* et *valeurs propres* de M_G ou L_G avec le graphe G en calculant le *spectre* de L_G. Supposons qu'un graphe G avec $V = \{1, 2, 3, 4\}$ et $E = \{(1, 2), (2, 3), (2, 4), (3, 4)$, comme dans la figure 7.2 La matrice laplacienne pour G est représentée comme suit :

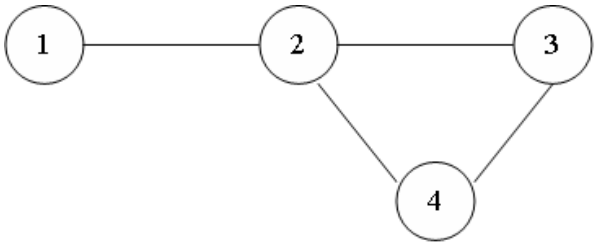


Fig. 7.2 : Un exemple de graphe avec quatre sommets

$$\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ 0 & -1 & -1 & 2 \end{bmatrix}$$

Afin de trouver les vecteurs propres et les valeurs propres, on résout l'équation $det(A - \lambda I) = 0$. On obtient $\lambda_1 = 0, \lambda_2 = 1, \lambda_3 = 3, \lambda_4 = 4$ et les vecteurs propres correspondants $v_1 = (1, 1, 1, 1), v_2 = (-2, 0, 1, 1), v_3 = (0, 0, 1, -1), v_4 = (1, -3, 1, 1)$. Ensuite nous calculons les valeurs obtenues aux sommets quand nous multiplions chaque vecteur avec L_G. Par exemple, pour vecteur $v_1 = (1, 1, 1, 1)$, au deuxième sommet, nous prenons trois fois la valeur qui était 1, ensuite soustrayons la valeur à chacun de ses voisins, aussi 1. Alors, nous obtenons 0. Faisons la même chose pour les sommets restants, nous trouvons le vecteur $(0, 0, 0, 0)$ (voir la figure 7.3). Le calcul similaire pour les autres vecteurs propres en multipliant par L_G, nous obtenons les

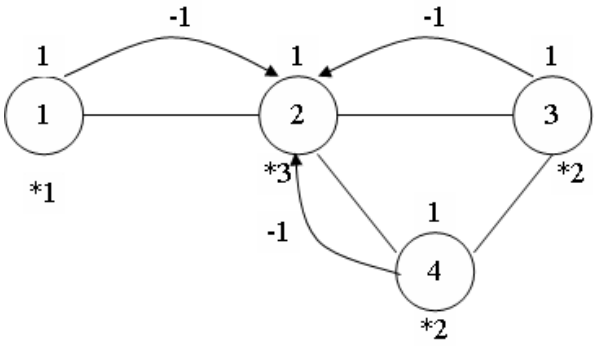


Fig. 7.3 : Le vecteur à tous uns est un vecteur propre avec la valeur propre 0

vecteurs correspondants :

$$\begin{aligned} v_2 = (-2, 0, 1, 1) \quad &\longrightarrow \quad (-2, 0, 1, 1) \\ v_3 = (0, 0, 1, -1) \quad &\longrightarrow \quad (0, 0, 3, -3) \end{aligned}$$

$$v_4 = (1, -3, 1, 1) \qquad \longrightarrow \qquad (4, -12, 4, 4)$$

La matrice laplacienne a quelques propriété élémentaires comme suit :

- En réalité, pour toute matrice laplacienne, le vecteur à tout 1s est un vecteur propre de valeur propre 0, et toutes les autres valeurs propres sont non-négatives. Nous dénotons les valeurs propres de L_G par $0 = \lambda_0 \leq \lambda_1 \leq ... \leq \lambda_{n-1}$.
- La propriété plus importante est qu'un vecteur propre est une fonction sur les sommets. C'est à dire, il assigne un nombre réel pour chaque sommet. Donc si l'on prend deux vecteurs propres, alors on obtient deux nombres réels pour chaque sommet. Ceci suggère un moyen pour décrire le graphe : prendre deux vecteurs propres, u et v, et tracer le sommet i au point de $(u(i), v(i))$. On prend souvent deux vecteurs propres u et v qui correspondent aux deux plus petites valeurs propres non-nulles de la matrice laplacienne. Ceci nous donne une très bonne image d'un graphe (voir [Spi04]) qui est utilisé dans le problème de décomposition de graphe.
- La matrice laplacienne de tout graphe est semi-définie positive. Notons qu'une matrice symétrique est semi-définie positive si toutes ses valeurs propres sont non-négatives.

7.2.3 Conductance du graphe et valeur Fiedler

Dans un graphe, un sous-ensemble des arêtes (sommets) qui déconnecte le graphe est appelé la *coupe*. Les coupes surgissent naturellement dans l'étude de connexité des graphes où les tailles des parties déconnectées ne sont pas examinées. La *conductance* du graphe a pour rôle d'optimiser des relations entre la taille de coupe et les tailles de parties séparées.

Il y a deux types de coupe. La première, *coupe-sommet* est un sous-ensemble des sommets dont la suppression déconnecte le graphe. Pareillement, une *coupe-arête* est un sous-ensemble des arcs dont sa suppression sépare le graphe. La taille d'un sous-ensemble des sommets dépend du nombre des sommets ou nombre des arêtes.

Ensuite, nous définissons la notion de *matrice non-orientée valuée*. Un graphe non-orienté valué $G = (V, E)$ est associé à une fonction de poids $w : V \times V \to \mathbb{R}$ qui satisfait :

$$w(i, j) = w(j, i)$$

et

$$w(i, j) \geq 0$$

Nous notons que si $(i, j) \notin E(G)$, alors $w(i, j) = 0$. Les graphes non-valués sont justement le cas particulier où tous les poids sont 0 ou 1.

La matrice laplacienne L_G de G est donnée par :

$$l_{ij} = \begin{cases} -w(i,j) & si\ (i,j) \in E \\ de_i - w(i,j) & si\ i = j, et \\ 0 & autrement \end{cases}$$

Dans ce contexte, le degré de_i d'un sommet i est défini $de_i = \sum_j w(i,j)$,

Le *volume d'un ensemble des sommets* de G : $vol(V) = \sum_{i \in V} de_i$,

Le *volume d'un ensemble des arêtes* de G : $vol(E) = \sum_{(i,j) \in E} w(i,j)$

Nous examinons une partition (S, T) de l'ensemble des sommets du graphe G avec la définition de la conductance de coupe :

$$\Phi(S) = \frac{\sum_{i \in S, j \notin S} w_{i,j}}{min(\sum_{k \in S} de_k, \sum_{k \notin S} de_k)}$$

et la *borne d'arête* (edge boundary) $\partial(S)$ de S se compose de toutes les arêtes avec exactement une extrémité dans S, i.e.,

$$\partial(S) = \{(i,j) \in E | i \in S, j \in T\}.$$

Parce que T est le complément de S $(T = V - S)$, alors clairement $\partial(S) = \partial(T) = E(S,T)$, où $E(S,T)$ dénote l'ensemble des arêtes avec une extrémité dans S et une extrémité dans T. Pareillement, nous pouvons définir la *borne de sommet* (vertex boundary) $\delta(S)$ de S être l'ensemble de tous les sommets i n'appartenant pas à S mais adjacents à un sommet de S, i.e.,

$$\delta(S) = \{j \notin S | (i,j) \in E(G), i \in S\}.$$

Nous pouvons écrire

$$\Phi(S) = \frac{vol(\partial(S))}{min(vol(S), vol(T))}$$

Enfin, la conductance de G est définie par

$$\Phi(G) = min_{S \subset V} \Phi(S)$$

D'une manière similaire, on définit une conductance analogue pour *l'expansion de sommet* (au lieu de l'*expansion d'arête*). Pour un sous-ensemble $S \subseteq V$, on a

$$\phi(S) = \frac{vol(\delta(S))}{min(vol(S), vol(T))}$$

et

$$\phi(G) = min_{S \subset V} \phi(S)$$

Néanmoins, dans le chapitre suivant, nous examinerons une technique de décomposition utilisant l'expansion de sommet.

La plus petite valeur propre non-nulle λ_1 de matrice laplacienne est souvent appelée la valeur Fiedler [Spi04]. Une version de l'inéquation de Cheeger [Spi04] dit que cette conductance est intimement liée à λ_1.

Théorème 7.1.

$$2\Phi \geq \lambda_1 \geq \frac{\Phi^2}{2de}$$

où "de" est une borne supérieure de degré de tout sommet dans le graphe.

Par l'inéquation de Cheeger, si λ_1 est petite, il est possible de couper le graphe en deux parties sans couper de trop nombreuses arêtes. Si λ_1 est large, alors toute coupe du graphe doit couper plusieurs arêtes.

7.3 Décomposition arborescente

La décomposition de graphe a été développée avec plusieurs méthodes comme la *décomposition modulaire* [CH94], la *décomposition linéaire*, la *décomposition en branches* [Tod06]. Notre approche est proche la *décomposition arborescente*.

7.3.1 Définitions

La décomposition arborescente d'un graphe $G = (V, E)$ est un couple $(\{V_i, i \in I\}, T = (I, E_T))$ où le V_i est l'ensemble de sommets de G, appelé le *sac* de la décomposition, et T est un arbre avec :

1. $\bigcup_{i \in I} V_i = V$
2. pour chaque arête $(u, v) \in E$, il existe un $i \in I$ tels que $u, v \in V_i$
3. pour tout $v \in V$, l'ensemble T_v des sommets $i | v \in V_i$ induit un sous-arbre de T.

La *largeur* d'une décomposition $(\{V_i, i \in I\}, T = (I, E_T))$ est $\max_{i \in I} |V_i| - 1$. La *largeur arborescente* de G, notée $tw(G)$, est la largeur minimum de toutes les décompositions arborescentes de G. Cette valeur détermine donc l'intérêt d'utiliser la méthode de décomposition. Le problème de trouver une décomposition arborescente de largeur minimale est un problème NP-difficile. Néanmoins, il existe un algorithme linéaire pour résoudre ce problème, mais le facteur constant induit par l'algorithme le rend impossible à appliquer à un graphe de taille moyenne. Dans certaines classes particulières de graphes, le calcul de la largeur arborescente se fait en un temps

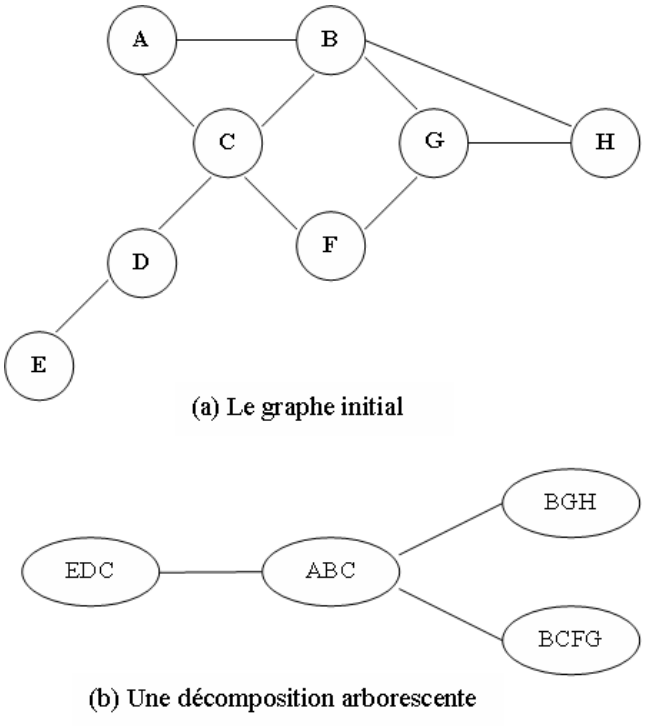


FIG. 7.4 : Un graphe et une décomposition arborescente

polynomial (arbres, graphes triangulés, ...). Lorsque l'arbre n'est constitué que d'un chemin (ou d'une chaîne), on parle de décomposition linéaire (path-decomposition) et de largeur (arborescente) linéaire (pathwidth).

Les composants V_i sont appelés *regroupements* ou *clusters*. Notons qu'une décomposition arborescente triviale d'un graphe quelconque est un arbre d'un seul noeud dont le cluster associé contient tous les noeuds du graphe originel. Nous prenons un exemple de décomposition comme dans la figure 7.4. Pour un graphe donné, on peut trouver plusieurs décompositions arborescentes possibles, donc la recherche d'une bonne décomposition est très importante. Il existe deux manières d'estimation d'une *bonne* décomposition arborescente : La première vision très classique constate les clusters d'une bonne décomposition comme les composants du graphe qui sont les plus fortement connectées. Dans la deuxième vision, on n'examine pas au niveau des clusters, mais au niveau des *séparateurs* entre ceux-ci. Nous nous intéressons à minimiser ce séparateur. Dans notre contexte, la largeur de décomposition est le cardinal de la partie la plus large. La largeur arborescente est la plus petite largeur de la plus grande partie.

Si le graphe originel a des bonnes propriétés, la recherche d'une décomposition optimale va être assez immédiate. La meilleure propriété du graphe pour la décomposition arborescente est triangulée.

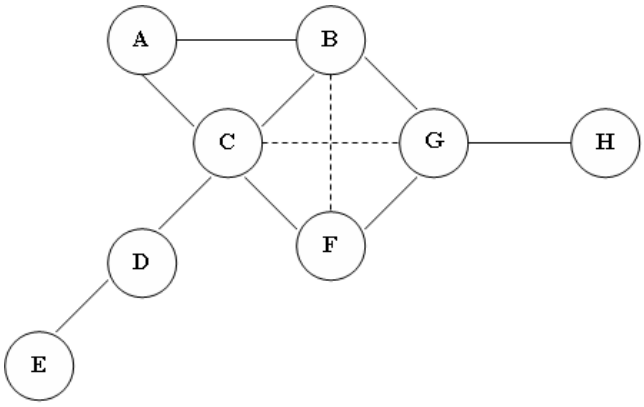


Fig. 7.5 : Une triangulation

7.3.2 Problèmes de triangulation

Le calcul de largeur arborescente est exprimable comme un *problème de triangulation*

Définition 7.2. *(Triangulation) Un graphe $H = (W, F)$ est une **triangulation** du graphe $G = (V, E)$ si $V = W, E \subseteq F$ et H est un graphe triangulé. En d'autre termes, H est un sur-graphe triangulé de G. H est une triangulation minimale de G si H est une triangulation de G et pour tout F' et $E \subseteq F' \subset F$, le graphe $H' = (V, F')$ n'est pas une triangulation de G.*

Une *triangulation* de graphe G est obtenue en ajoutant des arêtes à G pour construire un graphe triangulé. La figure 7.5 est une triangulation du graphe de la figure 7.4(a).

Définition 7.3. *(Arbre de cliques) Un arbre de cliques d'un graphe non orienté $G = (V, E)$ est un arbre $T = (I, E_T)$, dont chaque noeud $i \in I$ est associé à un sous-ensemble de sommets Ω_i, tels que les propriétés suivantes soient satisfaites :*

1. *chaque $\Omega_i, i \in I$ est une clique maximale de G, et*
2. *pour tout couple de cliques distinctes Ω_i et Ω_j, l'ensemble $\Omega_i \cap \Omega_j$ est contenu dans toutes les cliques sur la chaîne reliant Ω_i et Ω_j dans l'arbre T.*

Remarquons qu'un arbre de cliques de G est également une décomposition arborescente de G, dont les sacs sont précisément les cliques maximales de G. Alors, la largeur de cette décomposition est $\omega(G) - 1$.

Une relation entre le graphe triangulé et l'arbre de cliques est dans le théorème de Gavril [Gav74] :

Théorème 7.4. *[Gav74] Un graphe G est triangulé si et seulement s'il admet un arbre de cliques.*

Ce théorème fournit une manière très simple de trouver une décomposition arborescente pour le graphe triangulé. Cette décomposition est clairement la largeur minimale, parce que la largeur arborescente du graphe est minimisée par la taille de la plus grande clique. Par conséquent, nous constatons qu'étant donné une décomposition arborescente d'un graphe G, il existe une triangulation de G qui a la même décomposition. La figure 7.5 illustre une triangulation du graphe dans la figure 7.4(a) qui associe à la décomposition dans la figue 7.4(b).

Alors, le calcul de largeur arborescente peut être examiné comme un problème de triangulation. En particulier, ceci est exprimé dans la proposition suivante :

Proposition 7.5. *[Tod06] La largeur arborescente de G est égale au minimum, parmi toutes les triangulation H de G, de* $\omega(H) - 1$.

Cette proposition mène à une piste pour l'approximation d'une décomposition arborescente de largeur minimale d'un graphe quelconque.

Quelques algorithmes calculant la largeur arborescente et la triangulation minimale peuvent être trouvés dans [Tod06]. Dans ce mémoire, nous nous intéressons à la deuxième vision qui consiste plutôt à l'étude des séparateurs minimaux entre clusters, c'est à dire aux sommets qui se trouvent à la fois dans au moins deux clusters voisins.

7.3.3 Séparateurs minimaux

Définition 7.6. *(Séparateur) Un ensemble de sommets* S *est appelé un* (a, b)*-séparateur si* $\{a, b\} \subset V \setminus S$ *et chaque chemin reliant a et b dans G traverse au moins un sommet contenu dans* S*. Si* S *est un* (a, b)*-séparateur et il ne contient pas un autre* (a, b)*-séparateur, alors* S *est appelé un* (a, b)*-séparateur minimal.*

Définition 7.7. *(Connexité) Soit* $N(a, b)$ *le moindre cardinal d'un* (a, b)*-séparateur. La connexité du graphe G est le plus petit* $N(a, b)$ *pour tout couple de* $a, b \in V$ *dont* a, b *ne sont pas adjacents.*

Nous avons une autre définition comme suit : Soit $N(a, b)$ le moindre cardinal d'un a, b- séparateur. La *connexité* du graphe G est le plus petit de $N(a, b)$ pour tous $a, b \in V$ qui ne sont pas connectés par une arête. Un (a, b)-séparateur de cardinal minimal est dit être un (a, b)-séparateur minimal. Deux sommets a, b sont dits adjacents s'il y a une arête reliant a et b dans G.

Remarquons qu'un séparateur minimal peut être strictement contenu dans un autre séparateur minimal. Par exemple, dans la figure 7.4(a), le (D, B)-séparateur $\{C\}$ est strictement dans le A, F-séparateur $\{B, C\}$.

Soient S un ensemble de sommets de G, $C_G(S)$ l'ensemble des composantes connexes de $G - S$ et $C \in C_G(S)$ une composante connexe. Si tout sommet de S a au moins un voisin dans C, on dit que C est une *composante fortement connexe associée* à S (C est une *composante fortement connexe par rapport* à S) dans G.

Lemme 7.8. *[Tod06] S est un séparateur minimal de G si et seulement si $G - S$ a au moins deux composantes fortement connexes par rapport à S.*

Le nombre de séparateurs minimaux d'un graphe de n sommets peut être exponentiel par rapport à n. La complexité du calcul a été améliorée dans [BBC99].

Lemme 7.9. *[BBC99] Le temps du calcul de l'ensemble Δ_G des séparateurs minimaux de G est dans $O(n^3|\Delta_G|)$.*

Ensuite, nous examinons les séparateurs minimaux des graphes triangulés :

Théorème 7.10. *Un graphe G est triangulé si et seulement si tous ses séparateurs minimaux sont des cliques.*

Pour calculer les séparateurs d'un graphe, nous considérons le problème du *flot maximal*.

7.3.3.1 Problème du flot maximal

Définition 7.11. *(Flot) [Pri94] Soit un graphe orienté valué $G = (X, A, C, s, t)$ avec :*

- X un ensemble de N sommets et un ensemble A de M arcs ;

- C une application de A dans $\mathbb{N}$, C_{ij} désignant la capacité maximale de l'arc (i, j) ; on notera U le maximum des capacités.

- Deux sommets particuliers s et t, appelés la source et le puits

*Un tel graphe est appelé **réseau de transport**. Un flot de débit F est une application $f : A \to \mathbb{N}$ vérifiant (f_{ij} est le flux sur (i, j)) :*

- $\forall (i, j) \in A, 0 \leq f_{ij} \leq C_{ij}$, le flot sur chaque arc est compatible avec la capacité ;

- $\forall i \neq s,t, \Sigma_{j\in\Gamma(i)} f_{ij} = \Sigma_{j\in\Gamma^{-1}(i)} f_{ij}$, *le flot est conservé en chaque sommet ;*

- $F = \Sigma_{j\in\Gamma(s)} f_{sj} = \Sigma_{j\in\Gamma^{-1}(t)} f_{jt}$, *le débit total est égal au flot entrant par s et sortant par t.*

Le *problème du flot maximal* est de trouver un flot maximisant F. Pour résoudre ce problème, tout d'abord, on doit donner quelques notions :

Soit $G = (X, A, C, s, t)$ un réseau de transport, une *coupe* (coupe-sommet) de G est une partition $X = S \cup T | S, T \subseteq X, S \cap T = \emptyset, s \in S, t \in T$. La *capacité d'une coupe* (S, T) est la somme des capacités des arêtes sortant de la coupe :

$$C(S,T) = \Sigma_{(i,j),i\in S,j\in T} C_{ij}$$

La coupe est *minimale* si, pour toute coupe (S', T') alors $C(S,T) \leq C(S', T')$

Lemme 7.12. *[Jun99] Soient $G = (X, A, C, s, t)$ un réseau de transport, (S, T) une coupe et f un flot. Alors*

$$F = \Sigma_{i\in S, j\in T} f_{ij} - \Sigma_{i\in S, j\in T, i\neq s} f_{ji}$$

et donc $F \leq C(S,T)$, ceci montre que la capacité d'une coupe minimale est une borne supérieure sur la valeur d'un flot.

Une chaîne μ de s à t dans G est appelée une *chaîne améliorante* (par rapport à f) si, pour tout arc (i, j) non saturé et emprunté dans le sens du parcours (*arc avant*) de μ, la condition $f_{ij} < C_{ij}$ est satisfaite, et pour tout arc (i, j) non nul et emprunté dans le sens inverse du parcours (*arcs arrières*) de μ, la condition $f_{ij} > 0$.

[Jun99] montre également un résultat : Un flot f sur un réseau de transport G est maximal si et seulement s'il n'y a pas de chaîne améliorante par rapport à f. Ford et Fulkerson ont proposé le premier et le plus simple des algorithmes pour chercher le flot maximal dans un réseau de transport en recherchant cette chaîne améliorante. Nous allons aborder en détail un des ces algorithmes dans le chapitre suivant. Une propriété de séparateurs découle directement de la définition de la décomposition arborescente : si tous les sommets d'un séparateur sont retirés du graphe originel g, alors G n'est plus connexe. Donc les séparateurs sont des coupes.

En réalité, d'après le contexte, une coupe peut représenter soit un ensemble d'arêtes, soit un ensemble de sommets. Pour notre cas, nous examinerons deux méthodes de décomposition correspondant à chacun de ces deux choix. La première basée sur la "coupe minimale" du séparateur minimal du G (i.e., l'ensemble de sommets du séparateur minimal du G), et la deuxième basée sur la "coupe normalisée" (l'ensemble d'arêtes). Ces deux techniques seront étudiées plus en détail dans le chapitre suivant.

7.4 Conclusion

Ce chapitre a présenté des notions préliminaires dans la théorie des graphes et dans la théorie spectrale des graphes concernant nos présentations dans le chapitre suivant. Nous avons également quelques approches de décomposition générale qui consistent à mettre au point des techniques applicables à n'importe quel graphe. Dans le chapitre suivant, nous étudions deux techniques concrètes de décomposition qui reposent sur le séparateur minimal et le calcul de vecteurs propres du graphe.

Chapitre 8

Approches de G-décomposition

Dans ce chapitre, nous poursuivons la réponse à la question "comment obtenir une *bonne décomposition* de l'ontologie représentée par la logique de description ?" Nous commençons avec une vue d'ensemble de la modélisation conceptuelle qui s'oriente vers un schéma optimisé. Notre décomposition sera considérée comme une phase d'optimisation dans le processus de conception de l'ontologie.

Ensuite, nous proposons les critères d'une "bonne décomposition", ainsi que l'argument pour l'utilisation des techniques de partition de graphe à résoudre notre approche. Une décomposition de l'ontologie est dite "bonne" si elle préserve la sémantique et les services d'inférence de l'ontologie originelle (5.2.3 ou [PLT06]) et permet d'améliorer l'efficacité du raisonnement. En théorie des graphes, il existe plusieurs techniques de partition de graphe efficaces qui peuvent appliquer pour décomposer l'ontologie. Une méthode de décomposition de l'ontologie qui applique ces techniques est appelée *G-décomposition*.

Afin d'affirmer l'effet de G-décomposition, nous présentons une méthodologie pour G-décomposition et concrétisons par deux algorithmes de décomposition basés sur le séparateur minimal et sur la coupe normalisée. Ces algorithmes sont réalisés sur deux types de graphe représentant l'ontologie, le graphe de symboles et le graphe d'axiome [PLT07]. Ces graphes essentiellement conçus d'un point de vue syntaxique, c.-à-d. qu'elles décrivent caractères de syntaxe des axiomes d'une ontologie donnée. Enfin nous présentons quelques expériences qui ont été réalisées.

8.1 Conception optimisée de l'ontologie

Comme nous avons présenté dans la section 1.3, la modélisation conceptuelle offre une meilleure lisibilité / compréhension du contenu d'une ontologie, et une gestion plus efficace pour les grandes ontologies et des bases des connaissances associées. Les approches de LD

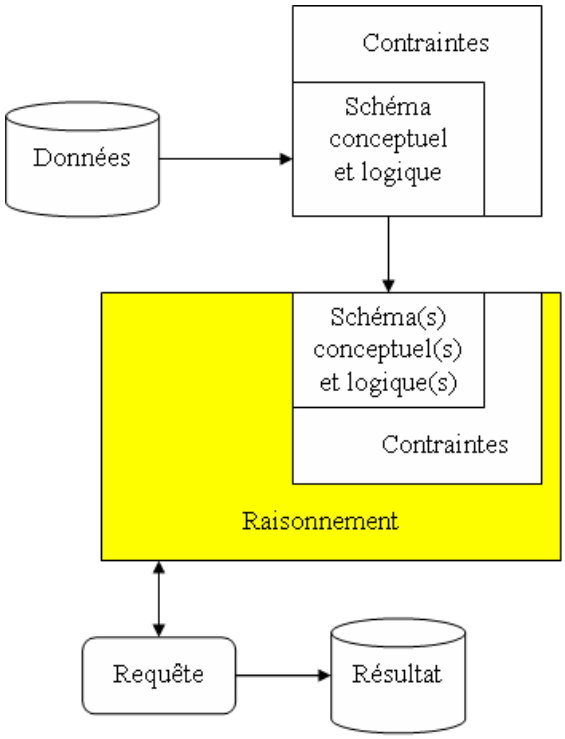


FIG. 8.1 : Optimisation de conception de l'ontologie

fournissent des grandes capacités de l'inférence de nouvelles connaissances à partir des connaissances définies explicitement. Pour exploiter les aspects de l'optimisation dans le traitement d'une grande ontologie de LD, nous nous intéressons à chercher des techniques d'optimisation dans la phase de conception de l'ontologie. Notre approche consiste à la décomposition d'une ontologie donnée en une ontologie distribuée se composant plusieurs sous-ontologies. De point de vue conceptuel, ceci est illustré succinctement dans la figure 8.1.

Une ontologie est vue ici comme un schéma conceptuel et logique. Le raisonnement est réalisé sur plusieurs schémas (sous-ontologies). Dans le chapitre 6, nous avons montré que le problème du raisonnement peut être efficacement résolu sur le système des sous-ontologies (présenté par l'ontologie distribuée) en donnant le même résultat qu'avec l'ontologie originelle. De plus, la gestion, le maintien et le développement des sous-ontologies sont plus faciles qu'une grande ontologie originelle.

8.2 Critères de décomposition

L'aspect sensible du choix d'une "bonne décomposition" de TBox est basé sur la réponse aux questions suivantes :

1. Quels sont les critères précis pour une bonne décomposition ?
2. Comment une telle décomposition peut-elle être calculée efficacement ?

- Afin de répondre à la première question, nous revoyons les recherches dans le chapitre 5 : Soit une ontologie LD, nous souhaitons chercher une décomposition qui satisfait les conditions de la définition 5.8 et minimise les deux formules données dans la section 6.4. L'analyse de ces formules n'a pas spécifié une complexité particulière du temps pour $f_{sat}(m)$, parce que f_{sat} dépend non seulement du nombre d'axiomes, mais aussi de l'expressivité du langage de LD utilisé (i.e., les constructeurs du langage de LD particulier). Néanmoins, les résultats suggèrent également une relation similaire entre la décomposition et le comportement du calcul. Ils soulignent que la taille de lien (le nombre des règles de pont) entre deux sous-ontologies avec la taille sont deux paramètres les plus pertinents pour le problème de satisfaisabilité. Les paramètres doivent être minimisés dans l'ordre suivant :

  1. k - le nombre de règles de pont.
  2. m_i - le nombre d'axiomes de la sous-TBox $\mathcal{T}_i$. Ceci est le nombre des symboles de la plus grande partie (la plus grande sous-ontologie). Donc, m_i est minimal quand $m_i \approx m_j, \forall j \neq i$ (dans notre cas, $i, j \in \{1, 2\}$).
  3. l - le nombre maximal d'axiomes qui sont relatifs à une règle de pont. Dans la section 6.4, ce paramètre n'est pas donné, parce que nous avons dit que $l = m_j$ dans le pire des cas.

  Si le nombre n des sous-ontologies est plus que 2, nous prenons k_i le nombre des règles de pont de $\mathcal{T}_i$ (les règles de pont qui prennent $\mathcal{T}_i$ comme l'ontologie de source ou l'ontologie cible). k_i signifie la degré de *dissociation* entre $\mathcal{T}_i$ et les autres. À ce moment, m_i est influencé par n, plus le nombre n est grand, plus m est petit. Dans ce cas, le paramètre n sera ajouté à la fin dans l'ordre au-dessus. De plus, si le paramètre l est petit, alors le temps du calcul pour f_{sat} sur $\mathcal{T}_j$ est aussi petit.
  Alors, en minimisant deux paramètres k et m_i, nous pouvons conclure que : les sous-TBox sont plus "disjointes" que possibles, et le cardinal des sous-TBox est équilibré. Ces critères nous guident dans la recherche des méthodes de décomposition convenables et efficaces. Concrètement, nous proposons deux méthodes de décomposition reposant sur la partition de graphe. La première basée sur des séparateurs minimaux, et la deuxième

basée sur des coupes normalisées. Les critères proposés au-dessus se sont reportés dans ces méthodes de décompositions :

- Pour la méthode basée sur des séparateurs minimaux : À partir de l'analyse de la complexité de nos algorithmes de raisonnement sur la TBox-décomposante, nous fournissons une métrique d'identification des paramètres qui influencent la décomposition : la taille de la partie de communication entre les TBox composantes, la taille de chaque composante, et la topologie du graphe de décomposition. Cette méthode peut être vue comme une approche syntaxique basée sur les structures des ICGs.
- Pour la méthode basée sur des coupes normalisées : L'idée est de minimiser l'association entre les TBox composantes et de maximiser l'association dans chaque composante. Cette méthode peut être vue comme une approche sémantique basée sur la relation sémantique entre les ICGs.

• Afin de répondre à la deuxième question, comme nous avons présenté dans les chapitres 5 et 6. La signification de cette décomposition est la préservation de la sémantique et des services d'inférence de l'ontologie originelle. Les algorithmes de raisonnement sont très efficaces pour les requêtes locales.

8.3 Vers la théorie des graphes

Le problème de décomposition de graphe a été bien développé et largement appliqué dans plusieurs domaines comme la segmentation d'image [SM00], le regroupement des données (data clustering) [DHZ+01], etc. Par exemple, selon le travail dans [SM00], l'ensemble de points dans un espace caractéristique arbitraire est représenté par un graphe non-orienté valué. Dans ce graphe, les sommets sont les points et une arête est formée par chaque paire de sommets, le poids sur chaque arête est une fonction de similarité entre deux sommets qui construit cette arête. Pour le regroupement des données, soient les attributs des points de données et la similarité (ou une métrique d'affinité) entre deux points quelconques, alors la matrice symétrique contenant les similarités entre toutes les paires de points forme une matrice d'adjacence pondérée d'un graphe non-orienté. Ainsi ces problèmes deviennent le problème de partition de graphe. De plus, sous le point de vue de partition de graphe, il permet de définir des critères plus appropriés pour la partition en calculant une *coupe* de graphe. Par exemple, *NCut* dans [SM00] ou *Min-max Cut* dans [DHZ+01]. Les critères de la partition visent à un principe : minimiser la *similarité* (ou l'*association*) entre deux sous-graphes et maximiser la similarité (ou l'association) dans un sous-graphe.

Une autre approche est basée sur la *décomposition arborescente* (tree decomposition) qui divise un graphe selon le *séparateur minimal* avec la *largeur minimale (largeur arborescente)*.

Dans cette méthode, on cherche des séparateurs minimaux, et puis le graphe est divisé par un séparateur avec la largeur minimale. Le séparateur minimal (coupe de graphe) assure que la connexion entre des sous-graphes est la plus petite. La largeur arborescente est la largeur de la plus grande partie, donc elle garantit l'équilibre cardinal des parties.

En résumé, dans les techniques de décomposition (partition) de graphe, on essaie de partitionner un graphe en des parties qui sont à peu près de la même taille, tout en minimisant le nombre d'arêtes entre ces parties. La partition de graphe est un élément fondamental dans plusieurs algorithmes. Par exemple, pour un graphe qui représente la communication exigée dans un calcul, la partition de graphe est utilisée pour partitionner le problème pour des processeurs parallèles. De plus, la partition de graphe est également utilisée comme un élément dans plusieurs algorithmes "diviser pour régner" (voir dans [Maz04]).

À partir des avantages ci-dessus, nous utilisons quelques techniques de partition de graphe dans la décomposition de l'ontologie. En vue de réaliser ce travail, nous essayons de transformer l'ontologie en des structures de graphe, en particulier, en deux formes, appelées *graphe de symbole* et *graphe d'axiomes* qui seront introduites dans le chapitre suivant. L'idée est de découper le graphe obtenu en plusieurs sous-graphes qui s'agencent selon certaines règles. Ces sous-graphes correspondent aux sous-ontologies. Afin de pouvoir résoudre efficacement le problème de raisonnement sur les sous-ontologies (comme introduit dans la partie précédente), nous devons chercher une "bonne" décomposition.

8.4 Méthodologie de G-décomposition

G-décomposition *est la méthode de décomposition de l'ontologie qui applique des techniques de partition de graphe*. Nous rappelons que la décomposition overlay d'une TBox est représentée par une TBox distribuée (TBox-décomposante) qui se compose d'un ensemble de sous-TBox et d'un ensemble de liens entre ces sous-TBox (mappings sémantiques). Cette décomposition ne considère que les axiomes, et les mappings sémantiques contiennent les concepts et les rôles communs de chaque paire de sous-TBox.

D'autre part, la décomposition d'un graphe est représentée par un *graphe d'intersection (graphe de décomposition)* dont chaque *sommet* est un sous-graphe et les *arêtes* représentent les connexions de chaque paire de sommets.

Par conséquent, nous pouvons représenter une TBox-décomposante par le graphe d'intersection. Chaque sous-TBox correspond à un sommet de graphe d'intersection, et chaque ensemble de mappings sémantiques reliant deux sous-TBox est représenté comme le label d'une arête de graphe.

Nous proposons une méthodologie de décomposition de l'ontologie comme un processus

qui se compose de trois phases principales (illustrées dans la table 8.1) : Transformation d'une TBox en un graphe, la décomposition de ce graphe en des sous-graphes, et la transformation des sous-graphes composants en la TBox distribuée.

PROCEDURE DECOMP-TBOX($\mathcal{T} = (\mathbf{C}, \mathbf{R}, \mathbf{A})$)

$\mathcal{T} = \{\mathbf{C}, \mathbf{R}, \mathbf{A}\}$ est une TBox, avec l'ensemble des concepts **C**, l'ensemble des rôles **R**, et l'ensemble des axiomes **A**.

(1) TRANS-TBOX-GRAPHE($\mathcal{T} = (\mathbf{C}, \mathbf{R}, \mathbf{A})$)

Construire un graphe $G = (V, E)$ de cette TBox, où chaque sommet $v \in V$ est un concept dans **C** ou un rôle dans **R** (ou un axiome dans **A**), chaque arête $e = (u, v) \in E$ si u et v apparaissent dans le même axiome (ou u et v ont au moins un concept (rôle) commun).

(2) DECOMP-GRAPHE($G = (V, E)$)

Décomposer le graphe $G = (V, E)$ obtenu dans la procédure TRANS-TBOX-GRAPHE en un graphe d'intersection $G_0 = (V_0, E_0)$, avec chaque sommet dans $v_0 \in V_0$ est un sous-graphe, chaque arête $e_0 = (v_0, u_0) \in E_0$ si v_0 et u_0 sont liés.

(3) TRANS-GRAPHE-TBOX($G_0 = (V_0, E_0)$)

Transformer le graphe $G_0 = (V_0, E_0)$ en une TBox distribuée, avec chaque sous-TBox correspondant à un sous-graphe, et les mappings sémantiques correspondant aux arêtes de E_0.

Tab. 8.1 : Algorithme générique de G-décomposition

1. *Transformation d'une ontologie en un graphe* (TRANS-TBOX-GRAPHE)
   - Cette phase construit des graphes $G = (V, E)$ qui représentent l'ensemble des axiomes **A** d'une TBox $\mathcal{T} = (\mathbf{C}, \mathbf{R}, \mathbf{A})$ et les relations entre eux.
   - Nous rappelons que les axiomes sont constitués à partir des concepts, des rôles et des constructeurs d'un langage de LD particulier. Les constructeurs ne sont pas considérés ici. Nous pouvons représenter *directement* ou *indirectement* les axiomes et leurs relations par un graphe. "Directement" signifie que les axiomes sont représentés directement par les sommets et leur relations sont les arêtes du graphe. "Indirectement" veut

dire que les éléments d'axiome (leur concepts et rôles) sont représentés dans un graphe. De plus, la coupe est un groupe des sommets ou des arêtes du graphe dont le graphe est divisé à lui-même. Cela nous guide de la recherche des manières de représentation de TBox par un graphe. Dans un graphe, les concepts et les rôles apparaissant dans les axiomes d'une TBox doivent être codés par les sommets ou les arêtes. Concrètement, nous introduisons deux types de graphes comme suit :

– La transformation de TBox en un *graphe de symbole* est réalisée en deux étapes. Premièrement, chaque axiome est décomposé en un ensemble des concepts et des rôles apparaissant dans cet axiome (selon la définition d'expansion 5.10 dans la section 5.2.2). Deuxièmement, chaque concept (rôle) est représenté par un sommet du graphe, il existe une arête entre deux sommets si leurs concepts et rôles sont dans le même axiome.
– La transformation de TBox en un *graphe d'axiome* est également réalisée en deux étapes. Le premier est comme au-dessus. Le deuxième présente chaque axiome par un sommet du graphe, si deux axiomes partagent au moins un concept ou un rôle alors ils sont liés par une arête.

Proposition 8.1. *Les algorithmes de transformation d'une ontologie en un graphe (graphe de symbole ou graphe d'axiome) sont linéaires dans le nombre d'axiomes et le nombre de concepts et de rôles apparaissant dans les axiomes. Les graphes obtenus sont uniques.*

Ces résultats sont facilement déduits à partir des définitions de graphe de symbole et de graphe d'axiome.

2. *Décomposition d'un graphe en sous-graphes* (DECOMP-GRAPHE)
   – Un principe élémentaire de partition (décomposition) de graphe est de couper un graphe G en des sous-graphes G_i à une *coupe*. La décomposition d'un graphe résultera dans un *graphe d'intersection* $G_0 = (V_0, E_0)$ dont chaque sommet dans V_0 présente un sous-graphe G_i ($G_i \in V_0$), une arête $e_0 = (G_i, G_j) \in E_0$ représente la relation entre deux sommets G_i, G_j et est étiquetée par l'ensemble de concepts et rôles communs de G_i et G_j.
   – La tâche principale de cette phase est de chercher une "bonne coupe" dans un graphe donné. Une coupe est dite "bonne" si elle satisfait les *critères proposés de décomposition*. Un critère général dans la décomposition de graphe est de diviser le graphe en parties (sous-graphes) les plus *disjointes* possibles. En particulier, chaque coupe divise le graphe en deux parties. Donc, une coupe exprime ici la connexion entre deux parties. Il y a clairement un rapport strict entre la coupe et le degré de disjonction des parties : si la coupe est minimisée alors le degré de disjonction est maximisé. Une *coupe minimale* est la valeur dont elle est minimisée.

– Les algorithmes de décomposition effectuent trois étapes principales : la première cherche une coupe minimale du graphe ; la deuxième coupe le graphe en deux parties à cette coupe (il peut être exécuté récursivement d'après le but de décomposition), et la troisième présente ces parties dans un graphe d'intersection dont chaque sommet représente une partie et chaque arête représente la connexion (coupe) de chaque paire de sommets.
La complexité de ces algorithmes dépend du pas de recherche d'une coupe minimale.

3. *Transformation du graphe d'intersection en la TBox distribuée* (TRANS-GRAPHE-TBOX) : Cette phase transforme le graphe d'intersection $G_0 = (V_0, E_0)$ en une TBox distribuée $\{\mathcal{T}_i, \mathcal{B}_{ij}\}$ dans laquelle chaque sous-TBox $\mathcal{T}_i$ représente un sommet (sous-graphe) $G_i \in V_0$, et les mappings sémantiques (règles de pont) $\mathcal{B}_{ij}$ représentent les étiquettes sur l'arête $(G_i, G_j) \in E_0$.

8.5 Décomposition basée sur le séparateur minimal

La décomposition de l'ontologie sera examinée seulement depuis ses axiomes, nous représenterons donc l'ensemble d'axiomes par des graphes. Pour simplifier la notation, désormais une TBox $\mathcal{T}$ peut être brièvement représentée par le seul ensemble d'axiomes $\mathbf{A}$.

8.5.1 Représentation d'une TBox par le graphe de symbole

Soit $\mathbf{A}$ l'ensemble d'axiomes de TBox $\mathcal{T}$, alors $Ex(\mathbf{A})$ est l'ensemble de concepts et des rôles qui apparaissent dans les expressions de $\mathbf{A}$. Pour simplifier, nous utilisons la notation de *symbole* au lieu de concept (rôle), c.-à-d., un symbole est un concept (rôle) dans une TBox. Un *graphe de symbole* sera présenté comme suit :

Définition 8.2. *(Graphe de Symbole) Un graphe $G = (V, E)$, où V est un ensemble de sommets et E est un ensemble d'arêtes, est appelé un graphe de symbole de $\mathcal{T} = \mathbf{A}$ si chaque sommet $v \in V$ est un symbole de $Ex(\mathbf{A})$, chaque arête $e = (u, v) \in E$ si $u, v \in V$ et u, v se trouvent dans un même axiome de $\mathbf{A}$.*

Alors, avec un ensemble d'axiomes $\mathbf{A}$, nous pouvons construire un graphe de symbole $G = (V, E)$ en prenant chaque symbole dans $Ex(\mathbf{A})$ comme un sommet et en reliant deux sommets par une arête si leurs symboles se trouvent dans un même axiome de $\mathbf{A}$. Selon cette méthode, chaque axiome est présenté en tant qu'une clique dans le graphe de symbole.

$(A_1) : C_1 \sqcap C_2 \sqsubseteq X$	$(A_6) : Y \sqcap T \sqsubseteq H$
$(A_2) : C_3 \sqsubseteq \neg C_2$	$(A_7) : C_3 \sqsubseteq X$
$(A_3) : X \sqcap C_4 \sqcap C_5 \sqsubseteq Y$	$(A_8) : \neg C_3 \sqsubseteq C_2$
$(A_4) : \neg C_4 \sqsubseteq \neg Y$	$(A_9) : \neg X \sqsubseteq \neg Y$
$(A_5) : Y \sqcap C_6 \sqsubseteq H$	$(A_{10}) : \neg C_5 \sqsubseteq \neg Y$

Fig. 8.2 : Exemple de TBox $\mathcal{T}$

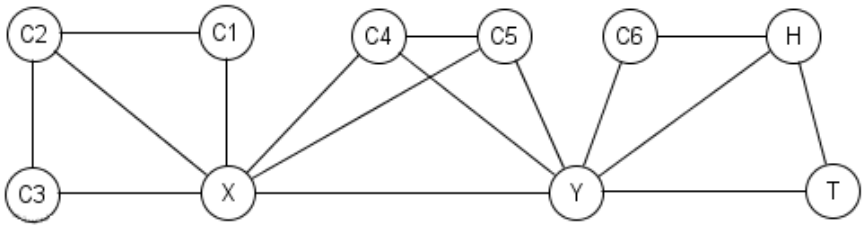


Fig. 8.3 : Graphe de symbole de TBox $\mathcal{T}$

Exemple 8.1 :

Examinons la TBox $\mathcal{T}$ dans la figure 8.2, alors le nombre d'axiomes de $\mathcal{T}$ est $m = 10$ et les axiomes sont étendus comme suit :

$Ex(\mathcal{T}) = \{C_1, C_2, C_3, C_4, C_5, C_6, X, Y, H, T\}$;

$|Ex(\mathcal{T})| = 10$

Cette TBox $\mathcal{T}$ peut être représentée par un graphe de symbole comme dans la Figure 8.3. À partir du graphe de symbole, nous considérons la méthode de décomposition arborescente basée sur des séparateurs minimaux. Cette méthode a été proposée par Seymour et Robertson dans le cadre de leur théorie sur les mineurs d'un graphe comme introduit dans le chapitre précédent. L'idée est de diviser l'ensemble de sommets en des ensembles disjoints $V_1, ..., V_n$ par le séparateur minimal, tels que le nombre de sommets dans le plus grand V_i et le nombre de sommets partagés entre les ensembles différents V_i, V_j sont minimaux. Le séparateur minimal est ici considéré comme une coupe de sommets du graphe.

Le résultat de cette décomposition est représenté par un graphe étiqueté (graphe d'intersection) qui s'appelle le *graphe de décomposition* ou l'*arbre de décomposition* [Jun99, AM05, SG92] $G_s = (V_s, E_s)$. Supposons que le graphe G représentant une TBox $\mathcal{T}$ est divisé en n sous-graphes $G_{i,i\leq n}$, alors un graphe de décomposition de G est défini comme suit :

Entrée : la TBox $\mathcal{T}(\mathbf{A})$ et M limite le nombre des symboles dans une partie (une sous-TBox $\mathcal{T}_i$).

Sortie : L'arbre $G_s = (V_s, E_s)$, et $\{\mathcal{T}_i\}$.

PROCEDURE DIVISION-TBOX($\mathbf{A}, M$)

(1) Transformer $\mathbf{A}$ en un graphe non orienté $G(V, E)$ avec $V = Ex(\mathbf{A})$ et $E = \{(l_1, l_2)|\exists A \in \mathbf{A}, l_1, l_2 \in Ex(\mathbf{A})\}$.

(2) Soit $G_s(V_s, E_s)$ un graphe non orienté avec $V_s = \{\{V\}\}$ et $E_s = \emptyset$.

(3) Exécuter DIVISION-GRAPH(G, M, nil, nil).

(4) Pour chaque $v \in V_s$, soit $\{\mathcal{T}_v = \{A \in \mathbf{A}|Ex(A) \subset v\}$. Retourne $\mathcal{T}_{v,v\in V_s}$ et G_s.

Tab. 8.2 : Un algorithme de décomposition de l'ensemble d'axiomes

Définition 8.3. *(Graphe de décomposition) [AM05] Graphe de décomposition est un graphe étiqueté $G_s = (V_s, E_s)$ dans lequel chaque sommet $v \in V_s$ est une partie (sous-graphe) G_i, chaque arête $e_{ij} = (v_i, v_j) \in E_s$ est marquée par l'ensemble de symboles partagés entre G_i et G_j, où $i \neq j \leq n$.*

Ce graphe de décomposition est équivalent à la décomposition overlay dans le chapitre 5, où G_i correspond à la TBox composante $\mathcal{T}_i$, l'arête e_{ij} correspond à l'ensemble de règles de pont identiques $\mathcal{B}_{ij}$.

Définition 8.4. *(Arbre de décomposition) [AM05] Si un graphe de décomposition est l'arbre (i.e., un graphe sans cycle), alors il est appelé l'arbre de décomposition.*

8.5.2 Algorithmes

Dans cette section, nous présentons un algorithme de décomposition récursif en utilisant l'algorithme de Even [AM05]. Il prend un graphe de symbole $G = (V, E)$ (résultant de la transformation d'une TBox $\mathcal{T}$) comme entrée et retourne un arbre de décomposition et l'ensemble de sous-graphes séparés de G. L'idée de l'algorithme de décomposition est de chercher une partie de connexion du graphe (coupe) en calculant le séparateur minimal du graphe, et puis le graphe est découpé par ce séparateur. Le processus de décomposition d'une TBox en un arbre de

décomposition est illustré par la procédure DIVISION-TBOX dans la table 8.2. Elle prend une TBox $\mathcal{T}$ avec l'ensemble d'axiomes, la limitation sur la taille d'une partie (une borne inférieure, M) comme entrée. La division considère initialement la TBox $\mathcal{T}$ comme une grande partie, et à chaque itération récursive elle casse une des parties en deux. Elle présente la structure arborescente des parties dans une variable globale G_s. Cette structure arborescente et l'ensemble de parties sont retournés comme résultat de la DIVISION-TBOX. La DIVISION-TBOX utilise la procédure DIVISION-GRAPHE pour décomposer un graphe qui représente la TBox $\mathcal{T}$ comme dans la table 8.3.

La procédure DIVISION-GRAPHE prend l'entrée qui se compose d'un graphe de symboles $G = (V, E)$ de $\mathcal{T}$, un paramètre limité, M (comme ci-dessus), et deux sommets a, b qui sont initialement assignés aux nils. Cette procédure met à jour la variable globale G_p pour représenter le processus de la décomposition. Dans chaque appel récursif, elle trouve un séparateur minimum des sommets a, b dans G. Si un des a, b est nil ou tous les deux sont nils, elle trouve le séparateur minimum global entre tous les sommets et le sommet non-nil (ou tous les autres sommets). Ce séparateur coupe le graphe G en deux parties G_1, G_2 et le processus continue récursivement sur ces parties.

Les variantes différentes de l'algorithme rendent des structures différentes pour le graphe de décomposition (graphe d'intersection) du résultat de division. La DIVISION-GRAPHE retourne un ensemble des parties. Si nous changeons l'étape 2(c) pour trouver un séparateur minimal ne sont pas inclus a, b, nous obtenons des arbres arbitraires. Si nous n'agrégeons pas R en r dans l'étape (6), nous obtenons des graphes arbitraires.

Dans la théorie des graphes, il existe plusieurs algorithmes pour trouver un séparateur minimal (voir dans [Jun99, KK98]), dans cette section, nous introduisons un algorithme de Ford-Fulkerson [Pri94, Jun99, AM05] : Cet algorithme prend deux sommets a, b et un graphe non-orienté G. Elle transforme G en un graphe orienté $\overline{G}$. Chaque sommet u de G est transformé en deux sommets (u', u'') dans $\overline{G}$ (correspondent au sommet d'*entrée* et sommet de *sortie*), et une arête orientée (arcs) les reliant dans la direction de l'entrée à la sortie $((v', \vec{v''}))$. Chaque arête (u, v) de G est transformé en deux arcs $(\vec{u''}, v'), (\vec{v''}, u')$ dans $\overline{G}$. Elle exécute un algorithme de max-flot sur $\overline{G}$. Le flot produit f a un débit de $N(a, b)$. Pour extraire un séparateur minimal, elle produit un réseau de $\overline{G}$ et le flot trouvé f dans l'étape (5). Le réseau contient un ensemble de sommets de $\overline{G}$. L'ensemble d'arcs entre cet ensemble des sommets et le reste de $\overline{G}$ correspond au séparateur.

La Figure 8.4 illustre le graphe de symbole de TBox $\mathcal{T}$ présentée par la Figure 8.3. Dans cet exemple, nous avons deux séparateurs minimaux $\{X\}$ et $\{Y\}$. Si on choisit $\{X\}$ pour diviser $\mathcal{T}$, alors nous collectons deux groupes de symboles $\{C_1, C_2, C_3, X\}$ et $\{X, C_4, C_5, C_6, Y, H, T\}$. Par conséquent, nous obtenons deux sous-TBox respectivement :

Entrée : Le graphe de symbole $G = (V, E)$, M est la limitation du nombre de symboles dans une partie (comme introduit au-dessus), et a, b sont dans V ou sont nil.

Sortie : L'arbre de décomposition $G_s = (V_p, E_s)$.

PROCEDURE DIVISION-GRAPHE(G, M, a, b)

(1) Si $|V| \leq M$, alors retourne V

(2) (a) Si $a, b = nil$, trouve un séparateur minimal R dans G.

(b) Si non, si $b = nil$, trouve un séparateur minimal R de a dans G.

(c) Si non, trouve les séparateurs minimaux R_a de a dans G, et R_b de b dans G.

Soit R le plus petit de R_a, R_b.

(3) Si $R = V$ alors retourne V.

(4) Soit G_1, G_2 les deux sous-graphes de G séparés par R, avec R inclus dans tous les deux sous-graphes.

(5) Soit $V_s \leftarrow V_s \setminus \{\{V\}\} \cup \{\{V_1\}, \{V_2\}\}$ et $E_s \leftarrow E_s \cup \{(\{V_1\}, \{V_2\})\}$. On change les arêtes qui étaient reliées à $\{V\}$ pour les relier à un de $\{V_1\}, \{V_2\}$

(6) Créer G'_1, G'_2 de G_1, G_2 respectivement en agrégeant les sommets dans R en un seul sommet r, enlever toutes les arêtes soi-même et relier r avec les arêtes à tous les sommets connectés par des arêtes à un sommet quelconque dans R.

(7) Exécuter DIVISION-GRAPHE(G'_1, M, r, a), DIVISION-GRAPHE(G'_2, M, r, b). Remplacer r dans les noeuds de V_p par les membres de R.

Tab. 8.3 : Un algorithme de décomposition du graphe

Entrée : Un graphe non orienté $G = (V, E)$, et deux sommets $a, b \in V$.

Sortie : Le (a, b)-séparateur R dans G.

PROCEDURE MIN-SEP-V($G = (V, E), a, b$)

(1) Construire un graphe orienté $\overline{G}(\overline{V}, \overline{E})$ de $G(V, E)$ comme suit.

Pour chaque sommet $v \in V$, ajouter deux sommets v', v'' (sommets d'entrée et de sortie) dans $\overline{V}$ avec une arête $e_v = (v', \overrightarrow{v''})$ (arête interne).

Pour chaque arête $e = (u, v)$ dans G, ajouter deux arêtes $e' = (u'', \overrightarrow{v'})$ et $e'' = (v'', \overrightarrow{u'})$ dans $\overline{G}$ (arêtes externes).

(2) Définir un réseau N, avec le graphe orienté $\overline{G}$, la source a'', le puits b' et les capacités d'unité pour toutes arêtes, $N = (\overline{G}, 1, a", b')$.

(3) Calculer le flot maximal f dans le réseau.

Un flot sur N est une application $f : E \to R_0^+$ qui satisfait les conditions :

(F1) $0 \leq f(e) \leq c(e)$ pour chaque arête e

(F2) pour chaque sommet $v \neq a, b$, on a que $\Sigma_{e^+=v} f(e) = \Sigma_{e^-=v} f(e)$, où e^- et e^+ dénotent les sommets de début et de fin, respectivement.

(4) Placer les capacités de toutes arêtes externes dans G à l'infini.

(5) Construire le réseau $\{V_i\}_{i \leq l}$ de $\overline{G}$ utilisant f. Soit $S = \bigcup_{i \leq l} V_i$.

(6) Soit $R = \{v \in V | v' \in S, v'' \notin S\}$. R est un (a, b)-séparateur minimal dans G.

Tab. 8.4 : Un algorithme de recherche du séparateur minimal

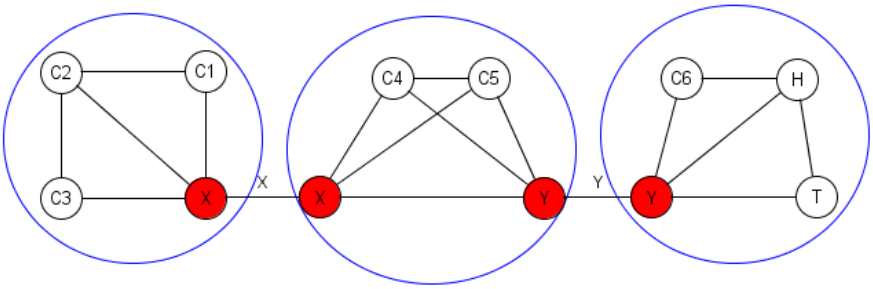


Fig. 8.4 : Décomposition du graphe de symbole $\mathcal{T}$

$\mathcal{T}_1 = \{A_1, A_2, A_7, A_8\}$ et $\mathcal{T}_2 = \{A_3, A_4, A_5, A_6, A_9, A_{10}\}$. De même, si on choisit $\{Y\}$, on obtient deux sous-TBox $\mathcal{T}_1 = \{A_1, A_2, A_3, A_4, A_7, A_8, A_9, A_{10}\}$ et $\mathcal{T}_2 = \{A_5, A_6\}$.

Nous allons prendre le premier résultat, car il y a une meilleure balance entre les nombres d'axiomes de deux sous-TBox. Ainsi, les axiomes dans la TBox originelle sont distribués en $\mathcal{T}_1$ et $\mathcal{T}_2$ avec $N_1 = 4$, $N_2 = 6$ respectivement.

8.5.3 Analyse de la complexité

Cette méthode examine chaque axiome comme un graphe complet (une clique) qui représente l'ensemble de symboles dans son expression. Donc le paramètre m_i - le nombre d'axiomes de la plus grande partie (sous-TBox) est remplacé par M - le nombre des symboles de la plus grande partie.

– Le temps d'exécution de DIVISION-TBOX dépend de la procédure DIVISION-GRAPHE et l'étape (1). La complexité de l'étape (1) dépend du nombre d'axiomes de $\mathcal{T}$ et du nombre de symboles dans les axiomes. Avec m le nombre d'axiomes de $\mathcal{T}$ et a le nombre de symboles du plus grand axiome du $\mathcal{T}$, le temps d'exécution de cette étape est $O(ma)$. La complexité de DIVISION-GRAPHE est évaluée dans la proposition suivante :

  Proposition 8.5. *La procédure DIVISION-GRAPHE prend le temps* $O(|V|^{5/2} \times |E|)$.
  La preuve de cette proposition peut être trouvée dans [AM05].

– Il existe plusieurs algorithmes pour trouver le flot maximal comme la méthode Simplex de Ford et Fulkerson [FF62, Jun99], la méthode push-relabel de Goldberg et Tarjan [Jun99] (la borne du temps est $O(|V|.|E|.log\frac{|V|^2}{|E|}$). Quand l'algorithme de Dinitz est utilisé pour résoudre le problème de réseau, l'algorithme de Even et Tarjan prend la complexité du temps $O(|V|^{1/2}|E|)$ [Eve79].
  De plus, l'algorithme de recherche d'un (a, b)-séparateur minimal utilisant l'algorithme de flot maximal de Ford-Fulkerson est dans le temps $O(t(|V| + |E|))$, où t est la largeur arborescente du G. Enfin, afin de calculer la connexité d'un graphe et un séparateur minimal, sans donner une paire (a, b), nous vérifions la connexité de c sommets (c est la connexité du graphe) à tous les autres sommets. Quand on utilise l'algorithme de Ford-Fulkerson pour un graphe de largeur arborescente t, cette procédure prend le temps $O(c.t.|V|.(|V| + |E|))$, où $c \geq 1$ est la connexité de G. Pour les cas de $c = 0, 1$ il y a des algorithmes de temps linéaire.

Comme nous l'avons dit dans le chapitre 6, $f_{sat}(m_i)$ n'est pas calculé particulièrement. Néanmoins, la recherche d'une "décomposition optimale" dépend des paramètres dans la section 8.2. Les règles de pont produisent les exigences du raisonnement sur la TBox cible, donc le nombre de règles de pont k est un facteur qui influence intensivement le temps de SAT-DIST et de SAT-

PARA sur la TBox-décomposante. Chaque axiome de sous-TBox produit des "branches OR" dans le processus d'exécution du raisonnement. Donc le nombre d'axiomes de la plus grande sous-TBox m_i influence également le temps de SAT-DIST et de SAT-PARA sur la TBox- décomposante. Ce propre m_i est la largeur de l'arbre de décomposition. Par conséquent, si nous supposons que $f_{sat}(m) = \Theta(2^m)$, le problème de recherche de la décomposition optimale pour les SAT-PARA et SAT-DIST est équivalent à la recherche de largeur arborescente (triangulations de nombre de clique minimum). Dans ce mémoire, nous ne considérons que le cas de deux TBox composantes, donc les algorithmes de divisions ne doivent pas exécuter l'étape récursive. Avec le cas général (i.e., les décompositions récursives), M devrait être choisi à 1, k devrait être choisi à n ($n = |Ex(\mathbf{A})|$), et l'algorithme arrêtera uniquement la décomposition récursive lorsqu'il atteindra un graphe qui est une clique.

8.6 Décomposition basée sur la coupe normalisée

Cette approche est aussi basée sur la formulation de théorie du graphe de groupement. L'ensemble d'axiomes est représenté par un graphe d'axiome connexe $G = (V, E)$, où les sommets du graphe sont les axiomes de TBox, et une arête est formée entre chaque paire de sommets. Le poids sur chaque arête, $w(i, j)$, est une fonction de l'association entre des sommets i et j. Le principe de cette approche est de chercher une partition des ensembles disjoints $V_1, ..., V_n$ de V dont l'association entre les sommets de chaque V_i est maximale et l'association des sommets entre les V_i, V_j différents est minimale. Ce principe est bien appliqué dans la segmentation d'image [SM00], ou le groupement de données (data clustering) [DHZ+01].

8.6.1 Représentation d'une TBox par le graphe d'axiome

Définition 8.6. *(Graphe d'Axiome) Un graphe non-orienté valué $G = (V, E)$, où V est un ensemble de sommets et E est un ensemble d'arêtes avec les valeurs de poids, est appelé un graphe d'axiome si chaque sommet $v \in V$ est un axiome dans la TBox $\mathcal{T}$, chaque arête $e = (u, v) \in E$ si $u, v \in V$ et il y a au moins un symbole partagé entre u et v, et le poids sur $e(u, v)$ est une valeur qui présente la similarité entre u et v.*

Exemple 8.2 :

La TBox $\mathcal{T}$ dans la figure 8.2 ci-dessus est représentée par un graphe d'axiome comme dans la Figure 8.5.

À partir du point de vue de graphe, nous présentons la TBox $\mathcal{T}$ par un graphe d'axiome comme suit : chaque sommet $v \in V$ correspond à un axiome dans $\mathcal{T}$, et chaque arête $(u, v) \in E$ est assignée par un poids $w(u, v)$. Ce poids devrait être choisi tel qu'il reflète le degré d'association (similarité) entre les sommets liés par chaque arête.

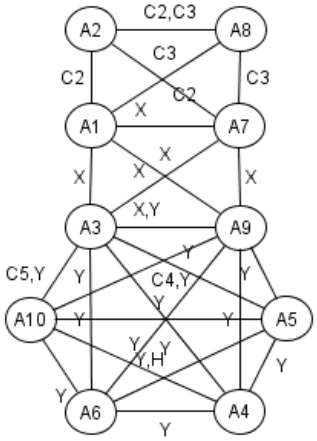


Fig. 8.5 : Graphe d'axiome de TBox $\mathcal{T}$

En utilisant juste les symboles communs entre chaque paire d'axiomes, nous pouvons simplement définir une fonction de poids $p : V \times V \rightarrow \mathbb{R}$ qui envoie une paire de sommets vers un nombre réel. En particulier, à chaque arête (i, j) est assignée une valeur w_{ij} qui décrit la connexion (l'association) entre deux axiomes A_i et A_j comme : $w_{ij} = \frac{n_{ij}}{n_i + n_j}$, où $i, j = \overline{1, m}, i \neq j$, m est le nombre d'axiomes dans $\mathcal{T}$ ($m = |\mathbf{A}|$), n_i, n_j est le nombre de symboles de A_i, A_j ($n_i = |A_i|$) respectivement, n_{ij} est le nombre de $A_i \cap A_j$ ($n_{ij} = |A_i \cap A_j|$).

8.6.2 Coupe normalisée

Tout d'abord, nous supposons qu'un graphe $G = (V, E)$ a été divisé en deux parties disjointes avec les deux ensembles de sommets A, B respectivement, où $A \cup B = V, A \cap B = \emptyset$. Il peut être simplement réalisé en enlevant des arêtes qui relient deux parties. Ensuite, on calcule le degré de *dissociation* entre ces deux parties et le degré d'*association* des éléments dans chaque partie. La coupe du graphe est ici considérée comme un ensemble d'arêtes dont le total de leurs poids est minimal. Une décomposition est bonne si l'association dans chaque partie est minimisée et la dissociation entre les parties différentes est minimisée.

La recherche d'une bonne décomposition est traitée en calculant la *coupe normalisée* (normalised cut). La coupe normalisée a été utilisée comme un critère dans la partition d'image. La minimisation de ce critère peut être formulée comme un problème de valeur propre généralisé. Les vecteurs propres de ce problème peuvent être utilisés pour construire des bonnes partitions de l'image et le processus peut être récursivement poursuivi si nécessaire.

Nous représentons le graphe G par une matrice valuée $N \times N$, avec N le nombre de sommets de G. Cette section décrit une méthode de division comme dans la segmentation d'image en uti-

lisant des vecteurs propres. La mesure de dissociation entre A et B s'appelant *coupe normalisée (NCut)* [SM00] est définie comme suit :

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \tag{8.1}$$

Une mesure d'association normalisée totale dans les parties avec une décomposition donnée :

$$Nassoc(A,B) = \frac{assoc(A,A)}{assoc(A,V)} + \frac{assoc(B,B)}{assoc(B,V)} \tag{8.2}$$

où $cut(A,B) = \sum_{u\in A, v\in B} w(u,v)$ est le poids total des arêtes qui ont été enlevées, et $assoc(A,V) = \sum_{u\in A, t\in V} w(u,t)$ est la connexion totale de sommets dans A à tous les sommets dans le graphe, et $assoc(B,V)$ est pareillement définie ; $assoc(A,A)$ et $assoc(B,B)$ sont des poids totaux des arêtes connectant des sommets dans A et dans B respectivement. La division optimale d'un graphe réduit à minimiser non seulement NCut mais aussi maximiser Nassoc dans les parties.

Nous pouvons simplement voir que $Ncut(A,B) = 2 - Nassoc(A,B)$. C'est une propriété importante d'une décomposition. Parce que deux critères trouvés dans l'algorithme de décomposition, minimiser la dissociation entre les parties et maximiser l'association dans chaque partie, sont identiques en réalité et peuvent être satisfaites simultanément.

Malheureusement, minimiser la coupe normalisée est exactement NP-complet, même pour le cas particulier des graphes sur des grilles [SM00]. Toutefois, les auteurs dans [SM00] ont également indiqué que si le problème de coupe normalisée est plongé dans le domaine de valeur réelle, alors une solution approximative peut être efficacement trouvée.

8.6.3 Partition optimale

Dans cette section, nous introduisons brièvement quelques calculs dans [SM00] pour trouver une partition optimale. Étant donné un graphe $G = (V,E)$, nous supposons une partition (A,B) de V ($A \subseteq V, B \subseteq V, A \cap B = \emptyset, A \cup B = V$).

La solution de partition peut être représentée par un vecteur indicateur $\mathbf{x}$ avec $N = |V|$ dimensions :

$$\begin{cases} x_i = 1, \text{ si le sommet i est dans A} \\ x_i = -1, \text{ autrement} \end{cases}$$

Soit $\mathbf{d}(i) = \sum_j w(i,j)$ la connexion totale du sommet i à tous les autres sommets. Soient $\mathbf{D}$ une matrice diagonale $N \times N$ avec $\mathbf{d}$ sur sa diagonale, $\mathbf{W}$ une matrice symétrique NxN avec $W(i,j) = w_{ij}$, $\mathbf{e}$ un vecteur $N \times 1$ à tous uns. Alors, $\frac{\mathbf{e}+\mathbf{x}}{2}$ et $\frac{\mathbf{e}-\mathbf{x}}{2}$ sont des vecteurs d'indicateur pour $x_i > 0$ et $x_i < 0$ respectivement, $k = \frac{\sum_{x_i>0} d_i}{\sum_i d_i}$, $b = \frac{k}{1-k}$. Nous pouvons ré-écrire $NCut(A,B)$ comme suit :

$$
\begin{aligned}
NCut(A,B) &= \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \\
&= \frac{\sum_{(x_i>0,x_i<0)} -w_{ij}x_ix_j}{\sum_{x_i>0} d_i} + \frac{\sum_{(x_i<0,x_i>0)} -w_{ij}x_ix_j}{\sum_{x_i<0} d_i} \\
&= \frac{(\mathbf{e}+\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{e}+\mathbf{x})}{k\mathbf{e}^T\mathbf{D}\mathbf{e}} + \frac{(\mathbf{e}-\mathbf{x})^T(\mathbf{D}-\mathbf{W})(\mathbf{e}-\mathbf{x})}{(1-k)\mathbf{e}^T\mathbf{D}\mathbf{e}} \\
&= \frac{(\mathbf{x})^T(\mathbf{D}-\mathbf{W})\mathbf{x}+\mathbf{e}^T(\mathbf{D}-\mathbf{W})\mathbf{e}}{k(1-k)\mathbf{e}^T\mathbf{D}\mathbf{e}} + \frac{2(1-2k)\mathbf{e}^T(\mathbf{D}-\mathbf{W})\mathbf{x}}{k(1-k)\mathbf{e}^T\mathbf{D}\mathbf{e}}
\end{aligned}
$$

Assignant $\mathbf{y} = (\mathbf{e} + \mathbf{x}) - b(\mathbf{e} - \mathbf{x})$, il est facile de voir que :

$$\mathbf{y}^T\mathbf{D}\mathbf{e} = \sum_{x_i>0} \mathbf{d}_i - b \sum_{x_i<0} \mathbf{d}_i = 0 \tag{8.3}$$

car $b = \frac{k}{1-k} = \frac{\sum_{x_i>0} \mathbf{d}_i}{\sum_{x_i<0} \mathbf{d}_i}$, et

$$
\begin{aligned}
\mathbf{y}^T\mathbf{D}\mathbf{y} &= \textstyle\sum_{x_i>0} \mathbf{d}_i + b^2 \sum_{x_i<0} \mathbf{d}_i \\
&= b \textstyle\sum_{x_i<0} \mathbf{d}_i + b^2 \sum_{x_i<0} \mathbf{d}_i \\
&= b(\textstyle\sum_{x_i<0} \mathbf{d}_i + b \sum_{x_i<0} \mathbf{d}_i) \\
&= b\mathbf{e}^T\mathbf{D}\mathbf{e}.
\end{aligned}
$$

Alors, on a :

$$min_{\mathbf{x}} NCut(\mathbf{x}) = min_{\mathbf{y}} \frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} \tag{8.4}$$

avec les conditions $\mathbf{y}^T\mathbf{D}\mathbf{e} = 0$ et $y_i \in \{2, -2b\}$.

Nous trouvons que l'expression 8.4 est le quotient de Rayleigh [GVL89]. Nous rappelons que la propriété du quotient de Rayleigh est "*Soit* $\boldsymbol{A}$ *une matrice symétrique réelle. Avec la contrainte que* $\boldsymbol{x}$ *est orthogonal aux* $j-1$ *vecteurs propres les plus petits* $\boldsymbol{x}_1, ...\boldsymbol{x}_{j-1}$*, le quotient* $\frac{\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}}$ *est minimisé par le prochain plus petit vecteur propre* $\boldsymbol{x}_j$*, et sa valeur minimum est la valeur propre correspondante* λ_j".

Si $\mathbf{y}$ est relâché pour prendre des valeurs réelles, on peut minimiser l'équation 8.4 en résolvant le système de valeur propre généralisé :

$$
\begin{aligned}
&\Leftrightarrow \quad (\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda(\mathbf{D}-\mathbf{W})\mathbf{y} \\
&\Leftrightarrow \quad \lambda\mathbf{y}^T\mathbf{D}\mathbf{y} = \mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y} \\
&\Leftrightarrow \quad (\mathbf{D}-\mathbf{W})\mathbf{y} = \lambda\mathbf{D}\mathbf{y}
\end{aligned} \tag{8.5}
$$

Cependant, nous avons deux contraintes sur $\mathbf{y}$ qui arrivent de la condition sur le vecteur d'indicateur $\mathbf{x}$ correspondant. Avec la première contrainte $\mathbf{y}^T\mathbf{D}\mathbf{e} = 0$ sur $\mathbf{y}$, nous pouvons montrer qu'elle est automatiquement satisfaite par la solution du système propre généralisé. En effet, cette équation est transformée en un système propre standard :

$$\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\mathbf{z} = \lambda\mathbf{z} \tag{8.6}$$

où $\mathbf{z} = \mathbf{D}^{\frac{1}{2}}\mathbf{y}$. La condition correspondante est également satisfaite ici. On peut facilement vérifier que $\mathbf{z}_0 = \mathbf{D}^{\frac{1}{2}}\mathbf{e}$ est un vecteur propre de 8.6 avec la valeur propre 0. De plus, $\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\mathbf{z}$ est la matrice "*semi-définie positive symétrique* (symmetric positive semidefinite)", car $(\mathbf{D}-\mathbf{W})$ est la matrice laplacienne, alors ses valeurs propres sont toujours réelles et positives. Donc $\mathbf{z}_0$ est le plus petit vecteur propre de l'équation 8.6, et tous les vecteurs propres de 8.6 sont perpendiculaires aux autres. En particulier, $\mathbf{z}_1$ est le deuxième plus petit vecteur propre et perpendiculaire à $\mathbf{z}_0$, donc $\mathbf{z}_1^T\mathbf{z}_0 = 0$. En transformant en arrière l'équation 8.5, on a :

- $\mathbf{y}_0 = \mathbf{e}$ est le vecteur propre le plus petit avec la valeur propre 0.
- $\mathbf{0} = \mathbf{z}_1^T\mathbf{z}_0 = (\mathbf{D}^{1/2}\mathbf{y}_1)^T\mathbf{D}^{1/2}\mathbf{e} = \mathbf{y}_1^T\mathbf{D}\mathbf{e}$

où $\mathbf{y}_1$ est le deuxième vecteur propre le plus petit de 8.5.

Comme un résultat, on obtient :

$$\mathbf{z}_1 = arg.min_{\mathbf{z}^T\mathbf{z}_0=0}\frac{\mathbf{z}^T\mathbf{D}^{-\frac{1}{2}}(\mathbf{D}-\mathbf{W})\mathbf{D}^{-\frac{1}{2}}\mathbf{z}}{\mathbf{z}^T\mathbf{z}} \tag{8.7}$$

et alors,

$$\mathbf{y}_1 = arg.min_{\mathbf{y}^T\mathbf{D}\mathbf{e}=0}\frac{\mathbf{y}^T(\mathbf{D}-\mathbf{W})\mathbf{y}}{\mathbf{y}^T\mathbf{D}\mathbf{y}} \tag{8.8}$$

Par conséquent, le deuxième plus petit vecteur propre de l'équation 8.5 est la solution à valeurs réelles pour notre problème de coupe normalisée. Parce que la solution à notre problème originel n'est pas nécessaire pour satisfaire la deuxième contrainte sur $\mathbf{y}$, où y_i prend sur deux valeurs discrètes. En réalité, on relâche cette contrainte dans le domaine de valeurs réelles pour faciliter le problème d'optimisation en premier lieu. Les auteurs dans [SM00] ont montré comment cette solution à valeurs réelles peut être transformée dans une forme discrète.

Un argument similaire peut aussi être fait pour montrer que le vecteur propre avec la troisième plus petite valeur propre est la solution à valeurs réelles qui optimise le problème de partition sur deux premières parties. Cet argument peut être étendu pour indiquer qu'on peut diviser les graphes existants, en utilisant chaque fois le vecteur propre avec la prochaine plus petite valeur propre. Néanmoins, en pratique, l'erreur de l'approximation entre la solution à valeurs réelles et la solution à valeurs discrètes est accumulée par tout vecteur propre pris, et tous les vecteurs propres doivent satisfaire une contrainte d'orthogonalité mutuelle globale. Par conséquent, les solutions basées sur des vecteurs propres plus grands deviennent non fiables. Il est donc préférable de redémarrer pour résoudre le problème de partition sur chaque sous-graphe.

Un résultat intéressant est que, tandis que le deuxième plus petit vecteur propre se rapproche seulement de la solution optimale de coupe normalisée, il est exact de minimiser le problème suivant :

$$inf_{\mathbf{y}^T\mathbf{D}\mathbf{e}=0}\frac{\sum_i\sum_j(\mathbf{y}(i)-\mathbf{y}(j))^2 w_{ij}}{\sum_i \mathbf{y}(i)^2\mathbf{d}(i)} \tag{8.9}$$

dans le domaine de valeur réelle, où $\mathbf{d}(i) = \mathbf{D}(i,i)$.

Entrée : la TBox $\mathcal{T}$ avec l'ensemble d'axiomes **A**

Sortie : le graphe de décomposition $G_p = (V_p, E_p)$ et $\{\mathcal{T}\}$.

PROCEDURE DIVISION-TBOX-NC(**A**)

(1) Transformer un ensemble d'axiomes **A** en un graphe d'axiome $G = (V, E)$
avec $V = \{v | v \in \mathbf{A}\}$ et $E = \{(u, v) | u, v \in V, w(u, v) = \frac{|u \cap v|}{|u \cup v|}\}$

(2) Soit $G_p(V_p, E_p)$ un graphe non orienté avec $V_p = \{\{V\}\}$ et $E_p = \emptyset$

(3) Exécuter DIVISION-GRAPHE-A($G = (V, E)$)

(4) Pour chaque $v \in V_p$, soit $\{\mathcal{T}_v = \{A \in \mathbf{A} | A = v\}\}$. Retourner $\mathcal{T}_{v, v \in V_p}$ et G_p.

Tab. 8.5 : Algorithme de décomposition de TBox basée sur la coupe normalisée

8.6.4 Algorithmes

L'algorithme de décomposition de TBox basé sur la coupe normalisée [SM00] est illustré par la procédure DIVISION-TBOX-NC dans la table 8.5. Cette procédure prend une TBox $\mathcal{T}$ avec l'ensemble d'axiomes **A** comme entrée. Elle transforme **A** en un graphe d'axiome $G = (V, E)$, où chaque axiome A_i de **A** est un sommet $i \in V$, chaque arête $(i, j) \in E$ est assignée par un poids $w(i, j) = \frac{|Ex(A_i) \cap Ex(A_j)|}{|Ex(A_i) \cup Ex(A_j)|}$. Ensuite, le processus est réalisé comme la procédure DIVISION-TBOX dans la section précédente.

La DIVISION-TBOX-NC utilise la procédure DIVISION-GRAPHE-A pour diviser le graphe d'axiome représentant $\mathcal{T}$ (voir la table 8.6). Cette procédure prend le graphe d'axiome G comme entrée, calcule les matrices **W, D**. **W** est une matrice valuée $N \times N$ avec $w(i, j)$ calculé comme ci-dessus. **D** est une matrice diagonale $N \times N$ avec les valeurs $\mathbf{d}(i) = \sum_j w(i, j)$ sur sa diagonale. Ensuite, nous résolvons l'équation $(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}$ avec les contraintes $\mathbf{y}^T \mathbf{D}\mathbf{e} = 0$ et $y_i \in \{2, -2b\}$, où **e** est un vecteur $N \times 1$ à tous uns, pour trouver les plus petites valeurs propres. La deuxième plus petite valeur propre est choisie et elle est la valeur minimale même de NCut. Nous prenons le vecteur propre qui correspond à cette valeur propre pour diviser G en deux parties G_1, G_2. Enfin, la DIVISION-GRAPHE-A met à jour la variable G_p comme dans la méthode basée sur le séparateur minimal. Cette procédure peut être exécutée récursivement, dans chaque appel récursif sur G_i, elle trouve un vecteur propre avec la deuxième plus petite valeur propre et le processus continue sur ce G_i.

La figure 8.6 illustre le NCut obtenu (dénoté par la ligne bleue) de l'exemple dans la figure 8.3, et le graphe du résultat de décomposition est montré dans la figure 8.7. Ce résultat est le même dans la méthode basée sur un séparateur minimal.

Entrée : le graphe d'axiome $G = (V, E)$

Sortie : le graphe de décomposition $G_p = (V_p, E_p)$

PROCEDURE DIVISION-GRAPHE-A($G(V, E)$)

(1) Trouver la valeur minimale de NCut en résolvant l'équation $(D - W)x = \lambda Dx$ pour des vecteurs propres avec les plus petites valeurs propres.

(2) Utiliser le vecteur propre avec la deuxième plus petite valeur propre pour décomposer le graphe en deux sous-graphes G_1, G_2.

(3) Soit $V_p \leftarrow V_p \setminus \{\{V\}\} \cup \{\{V_1\}, \{V_2\}\}$ et $E_p \leftarrow E_p \cup \{(\{V_1\}, \{V_2\})\}$. On change les arêtes qui étaient reliés à $\{V\}$ pour les relier à un de $\{V_1\}, \{V_2\}$

(4) Après que le graphe est cassé en deux parties, nous pouvons employer récursivement DIVISION-GRAPHE-A(G_1), DIVISION-GRAPHE-A(G_2).

Tab. 8.6 : Algorithme de décomposition du graphe d'axiome

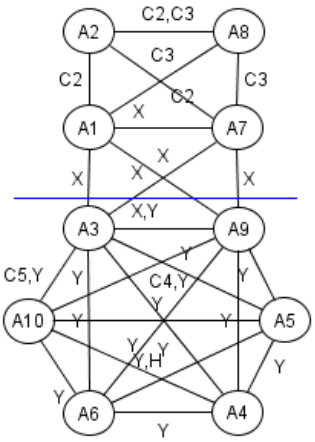


Fig. 8.6 : NCut du graphe d'axiome de $\mathcal{T}$

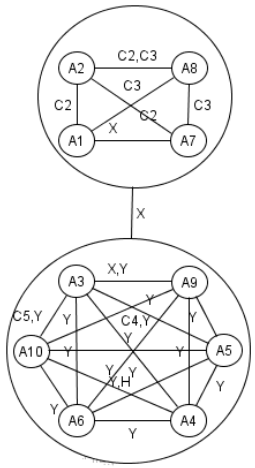


Fig. 8.7 : Décomposition du graphe d'axiome de $\mathcal{T}$

8.6.5 Analyse de la complexité

La résolution d'un problème de valeur propre standard pour tous les vecteurs propres prend $O(n^3)$ opérations, où n est le nombre de sommets du graphe. Cela devient inapplicable pour le cas d'un grand nombre d'axiomes. Cependant, notre graphe de décomposition a quelques

propriétés importantes :

- Seulement quelques premiers vecteurs propres sont nécessaires pour la décomposition de graphe,
- L'exigence de précision pour les vecteurs propres est faible.

Avec ces propriétés, notre problème peut être pleinement exploité par la méthode de Lanczos [GVL89]. Le temps d'exécution d'un algorithme de Lanczos est $O(mn) + O(mM(n))$, où m est le nombre maximal de calculs de matrice-vecteur exigés, $M(n)$ est le coût d'un calcul de matrice-vecteur de $\mathbf{Ax}$, où $\mathbf{A} = \mathbf{D}^{-1/2}(\mathbf{D} - \mathbf{W})\mathbf{D}^{-1/2}$. Si la structure dans $\mathbf{W}$ est épars, alors le calcul de matrice-vecteur $\mathbf{A}$ est seulement de $O(n)$, avec n est le nombre de sommets.

Le coût de produit scalaire de $\mathbf{A}$ avec un vecteur $\mathbf{x}$ est $O(n)$. En effet, soit $\mathbf{y}_i = \mathbf{A}_i.x = \sum_j \mathbf{A}_{ij}\mathbf{x}_j$. Pour un i fixé, $\mathbf{A}_{ij}$ est seulement non-zéro si le sommet j est dans un voisinage spatial de i. Donc, il y a seulement un nombre fixé d'opérations exigées pour chaque $\mathbf{A}_i.\mathbf{x}$, et le coût total de calcul $\mathbf{Ax}$ est $O(n)$.

8.7 Evaluation et expériences

Nous avons appliqué deux algorithmes de décomposition basés sur le séparateur minimal et sur la coupe normalisée pour décomposer une TBox. Dans cette section, nous présentons quelques modules principaux qui sont exécutés dans nos expériences. Afin d'illustrer nos résultats, nous prenons une TBox qui est extraite du fichier "tambis.xml" dans le système de FaCT. Cette TBox, appelée Tambis1, se compose de 30 axiomes (voir dans la partie d'annexe 1).

- *Transformation de l'ontologie en graphe de symbole* : Ce module peut lire un fichier représentant une TBox en XML. Le fichier lu est transformé en un graphe de symbole. La figure 8.8 illustre le graphe de symbole de TBox Tambis1 avec les sommets étiquetés par les noms de concept et de rôle.
- *Transformation de l'ontologie en graphe d'axiome* : Ce module réalise la même fonction avec le module au-dessus, il résulte en un graphe d'axiome. La figure 8.9 décrit le graphe d'axiome de TBox Tambis1 avec les sommets étiquetés par les symboles A_i ($i \in \{0, .., 29\}$).
- *Décomposition basée sur le séparateur minimal* : décompose un graphe d'axiome en un arbre où les noeuds de feuille sont les axiomes. La figure 8.10 présente cette décomposition pour Tambis1.
- *Décomposition basée sur la coupe normalisée* : décompose un graphe de symbole en un arbre où les noeuds de feuille sont les axiomes. La figure 8.11 illustre cette décomposition pour Tambis1.

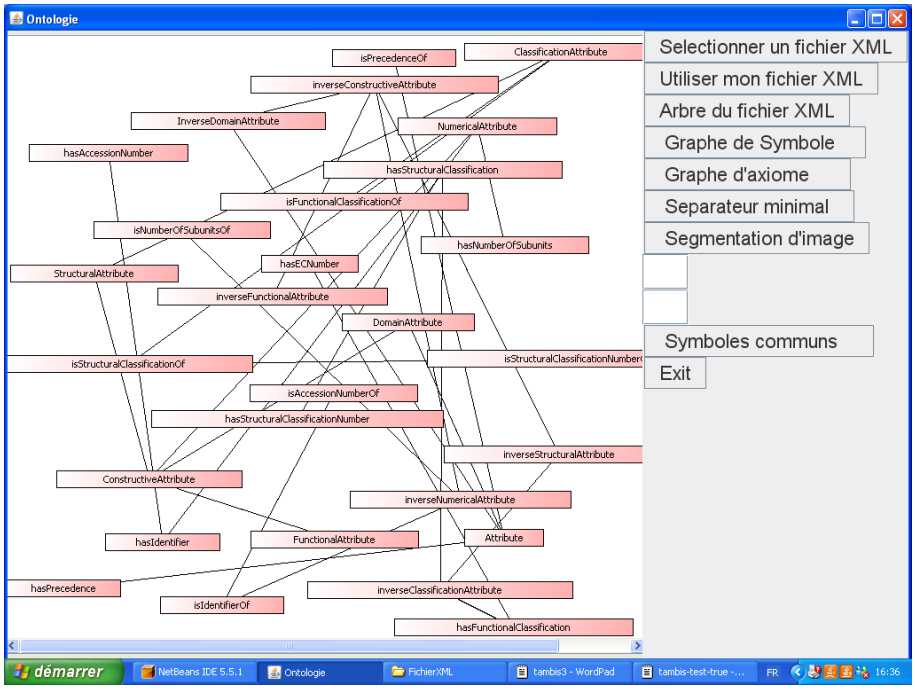


FIG. 8.8 : Graphe de symbole de Tambis1

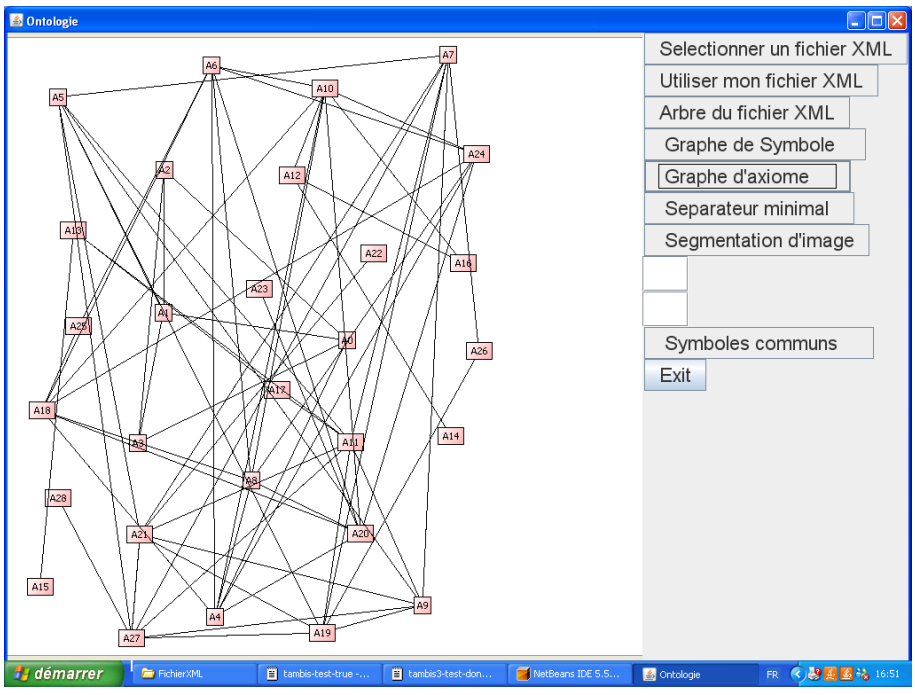


FIG. 8.9 : Graphe d'axiome de Tambis1

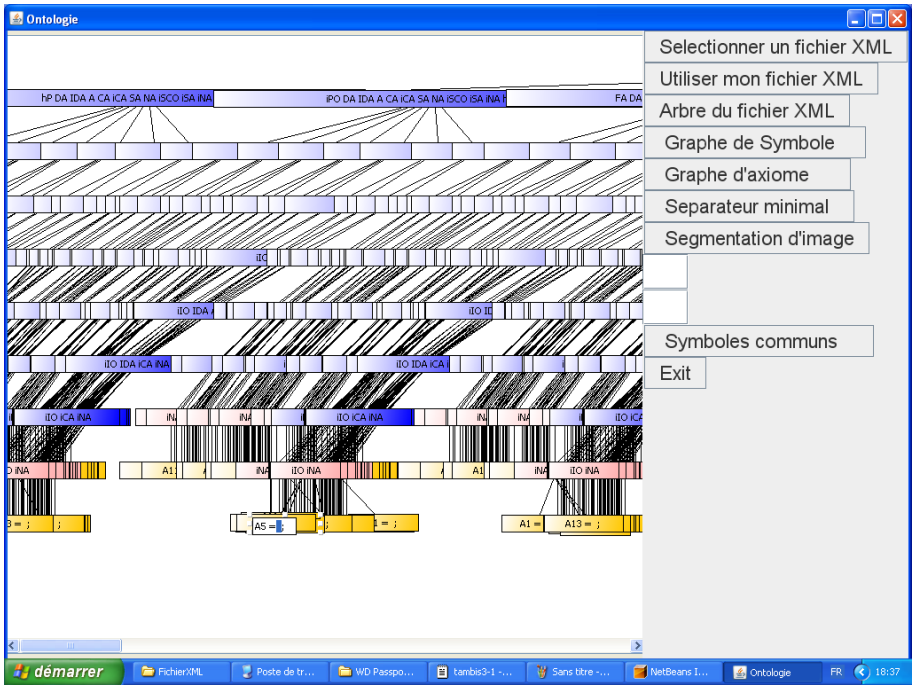


FIG. 8.10 : Arbre de décomposition basée sur le séparateur minimal de Tambis1

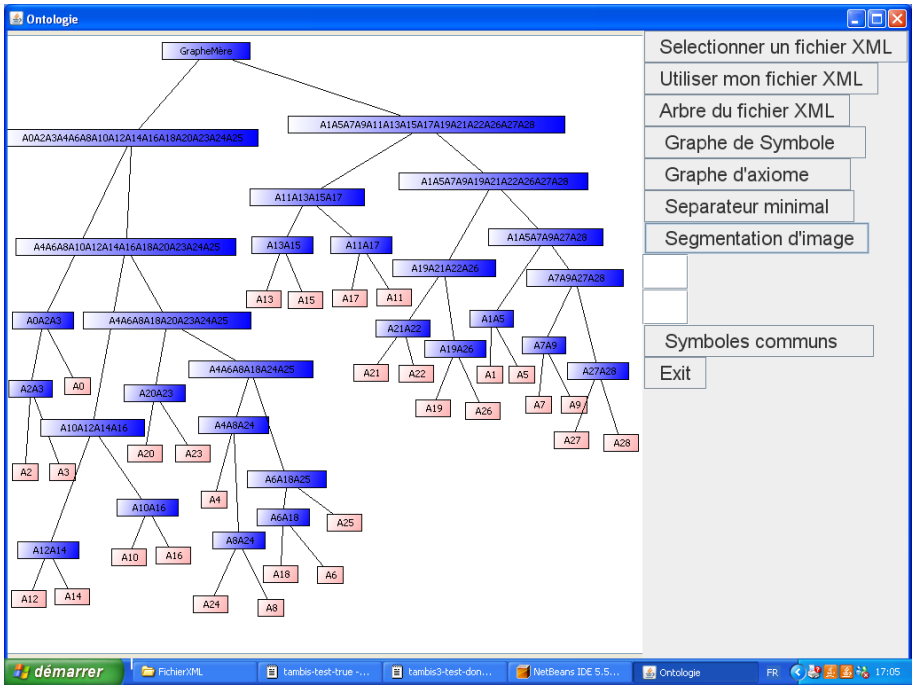


FIG. 8.11 : Arbre de décomposition basée sur la coupe normalisée de Tambis1

Ces deux méthodes renvoient les résultats qui satisfont les propriétés proposées de notre décomposition. Tous les concepts, les rôles et les axiomes sont préservés sur l'exécution de décomposition. Les axiomes et leurs relations sont bien exprimées par le graphe de symbole et le graphe d'axiomes. L'ensemble des axiomes dans la TBox originelle est alors réduit par la distribution régulière en des sous-TBox. Les techniques de décomposition se concentrent pour trouver une bonne décomposition. La méthode de décomposition basée sur le séparateur minimal minimise (réduit au minimum) le nombre de symboles partagés entre les parties composantes et essaie un équilibrage du nombre d'axiomes dans ces parties. Nous devons retrouver les axiomes après la décomposition. C'est possible parce que les axiomes onr été codés par des cliques dans le graphe de symbole. Cependant, en réalité la difficulté de ce problème est qu'il existe quelques cliques du graphe de symbole et du graphe d'intersection qui ne sont pas exactement des axiomes.

L'avantage possible de la méthode de décomposition basée sur la coupe normalisée est qu'elle conserve les axiomes. Après la décomposition, nous pouvons directement trouver les axiomes dans les parties composantes. En outre, la mesure du NCut est normalisée, elle exprime la dissociation entre les parties différentes et l'association dans chaque partie de la décomposition. Cependant, l'efficacité de cette méthode dépend du choix des paramètres appropriés pour calculer la relation de similarité entre deux axiomes (le poids des arêtes entre les sommets du graphe d'axiome).

Nous avons testé sur des parties de TBox utilisées dans le système de FaCT, comme Veda-all, modkit, people, platt, et tambis. Les résultats montrent que pour les axiomes dont les expressions sont plus complexes, l'application de méthode de coupe normalisée est beaucoup efficace (par exemple, Veda-all, modkit), tandis que la méthode de séparateur minimal est plus efficace avec les axiomes simples (par exemple, platt, tambis).

8.8 Discussion et conclusion

Comme nous avons défini la décomposition overlay dans le chapitre 5, seulement les axiomes de l'ontologie originelle sont considérés et doivent être conservés par la décomposition. Donc, la coupe de graphe n'est examinée qu'avec les concepts et rôles. Le premier algorithme de décomposition de TBox basé sur le séparateur minimal est réalisé sur le graphe de symbole, parce que le séparateur minimal est un sous-ensemble des sommets. La coupe normalisée est calculée sur les arêtes, donc le graphe d'axiome est utilisé dans le deuxième algorithme de décomposition de TBox dans LDs basés sur la coupe normalisée. Ces algorithmes de décomposition sont suffisants pour notre but dans les conditions des TBox composantes. Cependant, le choix d'une méthode de décomposition dépend de la structure de TBox originelle. Intuitivement, avec

une TBox qui contient un nombre large de symboles et les axiomes complexes, on peut choisir la méthode du graphe d'axiome. Et pour une TBox qui contient beaucoup d'axiomes, on peut utiliser la méthode du graphe de symbole. Nous proposons aussi quelques propriétés essentielles d'une bonne décomposition qui influencent la performance du raisonnement comme fourni dans [PLT06]. Les transformations du graphe d'une TBox dépendent encore sur une méthode effective pour analyser les caractéristiques probables d'une TBox donnée.

Dans ces deux types de graphe, nous pouvons décrire les axiomes et les relations entre eux. Pourtant, ils sont "syntaxiquement" représentés. Nous disons "syntaxiquement", c'est parce que les graphes décrivent les axiomes A_i et les relations entre eux par les concepts et les rôles qui apparaissent dans leur expressions, tandis que chaque axiome A_i est une expression de l'ensemble de concepts, de rôles et de constructeurs d'un langage de LD particulier. Quand nous présentons A_i par $Ex(A_i)$, c'est à dire les symboles de $Ex(A_i)$ ont le même rôle. Par exemple, on a deux axiomes :

$$A_1 : \forall \text{ travailler.UNIVERSITE} \sqcup \text{ PROFESSEUR} \sqsubseteq \text{EMPLOYE}$$
$$A_2 : \text{PROFESSEUR} \sqsubseteq \exists \text{ travailler.UNIVERSITE} \sqcap \text{EMPLOYE}$$

Ces deux axiomes sont représentés par l'ensemble {travailler, UNIVERSITE, PROFESSEUR, EMPLOYE}. A_1 et A_2 ont la même représentation dans le graphe de symbole, et sont liés par une arête avec le poids à 1 dans le graphe d'axiome. Cependant, les sémantiques de A_1 et A_2 sont différentes.

Nous remarquons que, le graphe de décomposition obtenu $G_s = (V_s, E_s)$ dans l'algorithme de décomposition basé sur le séparateur minimal est différent de $G_p = (V_p, E_p)$. C'est parce que les sommets dans V_s représentent les symboles, tandis que les sommets dans V_p représentent les axiomes. Afin de trouver une TBox distribuée avec les sous-TBox correspondants aux sous-ensembles d'axiomes, $G_s(V_s, E_s)$ doit être transformé en $G_p(V_p, E_p)$. C'est possible par la structure graphique d'un axiome, chaque axiome est représenté par une clique. Cependant l'inverse n'est pas toujours vrai, c.-à-d., tous les cliques ne sont pas toujours des axiomes. Nous pouvons résoudre ce problème en faisant une marque pour les symboles dans un même axiome.

Ce chapitre a montré que nous pouvons représenter une TBox donnée par deux types de graphes de symbole et de graphe d'axiome. Deux méthodes de décomposition utilisant ces deux types qui prennent la coupe comme un ensemble des sommets ou des arêtes. Cependant, nous ne pouvons pas toujours obtenir une bonne décomposition, parce que la condition préalable pour une bonne décomposition dépend des expressions des axiomes de l'ontologie originelle, i.e., la topologie de son graphe. Par exemple, si le graphe de symbole représentant l'ontologie est une clique, ou si le graphe d'axiome représentant l'ontologie a les mêmes valeurs de poids sur les arêtes similaires, alors la conservation de ces ontologies est préférable à leur décomposition.

Conclusion et perspectives

Cette thèse conclut avec une révision des travaux présentés et l'estimation des objectifs proposés dans l'introduction. La portée des résultats principaux est résumée, les limites sont discutées et les directions pour des travaux futurs sont suggérées.

Résumé de thèse

Le raisonnement efficace dans une grande base de connaissance en logique de description est un défi actuel en raison des inférences "insurmontables", même pour des langages logiques de description relativement inexpressifs. Le raisonnement dans la logique de description consiste essentiellement au test de la relation de subsomption entre les concepts. Par conséquent, on cherche toujours les expédients pour optimiser ce raisonnement. Des techniques d'optimisation pour améliorer la performance du raisonneur d'une LD se divisent donc naturellement en trois niveaux. Le premier est le niveau conceptuel considérant des techniques pour optimiser les structures d'axiomes dans la terminologie (TBox). Le deuxième est le niveau algorithmique examinant des techniques pour réduire les exigences de stockage de l'algorithme de tableaux et optimisant le test de la relation de subsomption (satisfaisabilité). Le troisième est l'optimisation de requête cherchant des stratégies d'exécution optimales d'une requête d'interrogation dans une base de connaissances.

Dans ce mémoire, nous avons étudié une approche de décomposition de l'ontologie s'appelant la "décomposition overlay" qui vise à deux objectifs principaux : l'optimisation du raisonnement et la méthodologie de conception des ontologies.

- D'une part l'optimisation dans laquelle nous cherchons à diviser une ontologie dans un ensemble de sous ontologies dont chacune contient une partie de l'ensemble d'axiomes de l'ontologie originelle permet d'obtenir ainsi une réduction du temps relatif de raisonnement ;
- D'autre part la méthodologie de conception : qui permet de remplacer une ontologie par un ensemble d'ontologies dans une organisation plus ou moins "optimale".

En effet, pour le premier objectif, la présence des axiomes est une des raisons importantes

causant une augmentation exponentielle de la taille de l'espace de recherche exploré par les algorithmes d'inférence. Intuitivement, le fait de pouvoir raisonner parallèlement sur plusieurs sous-ontologies ayant chacune un espace de recherche réduit, peut conduire à une réduction du temps relatif du raisonnement. La décomposition overlay de l'ontologie résulte en une *ontologie-décomposante (TBox-décomposante)* représentée par la logique de description distribuée. Une propriété importante de cette ontologie est d'être interprétée dans le même domaine de l'ontologie originelle. Ceci est une base qui nous suggère la proposition de deux algorithmes de raisonnement pour cette ontologie-décomposante.

Concernant l'objectif de méthodologie de conception, à partir des outils et des propriétés dans la théorie de graphe pour traiter le problème de partition de graphe, nous introduisons deux méthodes de décomposition de l'ontologie reposant sur la décomposition heuristique des graphes. Une méthode est reposée sur la décomposition selon les séparateurs minimaux des graphes triangulaires et la seconde sur la décomposition selon la mesure des coupes normalisées des régions d'un graphe. Les algorithmes dans ces deux méthodes sont réalisés sur deux types de graphe, *graphe de symbole* et *graphe d'axiome*, qui sont transformés d'une ontologie.

Contributions

Dans ce mémoire, nous avons étudié un ensemble d'éléments de la théorie de décomposition overlay qui permet de transformer une ontologie en la logique de description (LD) dans un ensemble des ontologies en la logique de description distribuée (LDD). Nous avions pour objectif de montrer la préservation sémantique et d'inférence d'une telle décomposition, et de proposer des algorithmes spécifiques de décomposition respectant certains critères d'optimisation.

L'originalité de notre travail est de considérer la décomposition comme une technique permettant l'obtention, dès la phase de conception de l'ontologie, d'un schéma d'organisation logique optimal qui assure une inférence parallèle et/ou distribuée. Plus précisément, les résultats suivants peuvent être mis à l'actif de notre travail de recherche :

1. **Définition de la méthode de décomposition overlay**
   La décomposition overlay consiste à transformer une ontologie de LD (TBox) en une ontologie distribuée de LDD (TBox distribuée) s'appelant l'ontologie-décomposante (TBox-décomposante). La TBox-décomposante se compose des sous-TBox et des règles de pont spéciales, dites règles d'identité, entre des sous-TBox. Chaque sous-TBox contient tous les concepts, tous les rôles et un sous ensemble des axiomes de la TBox originelle. Les ensembles des axiomes de sous-TBox sont disjoints. Cette technique de décomposition peut être appliquée pour toute classe de langages de LD.

2. **Préservation de la sémantique dans la méthode de décomposition overlay**
   Une première question posée est que "*Est-ce que la décomposition cause des pertes de l'information contenue dans l'ontologie originelle ?*". À partir d'une étude approfondie des propriétés, nous avons montré que, la décomposition overlay préserve la sémantique de l'ontologie originelle. Ce résultat nous permet de confirmer la direction de recherche choisie.

3. **Préservation de l'inférence des algorithmes de tableaux**
   Jusqu'à ce jour, les mécanismes d'inférence dans les langages de LD expressifs reposent sur l'algorithme de tableau. Il est important d'étudier la question suivante : "*Est-ce que la méthode de décomposition overlay supporte toujours l'inférence par l'algorithme de tableau ?* ". Nous avons étudié en détail les mécanismes d'inférence sur la Tbox-décomposante et montré que l'inférence par des algorithmes de tableaux est préservée (complète et correcte) sur la Tbox-décomposante. Ce résultat est important pour assurer l'applicabilité de la décomposition overlay.

4. **Proposition de deux algorithmes de tableaux parallèle et distribué**
   Le travail d'étude des mécanismes d'inférence à base des algorithmes de tableaux, nous a permis de concevoir deux algorithmes de tableaux spécifiques pour l'inférence dans la TBox-décomposante. Naturellement, un premier algorithme est une dérivée des algorithmes de tableaux distribués dans la LDD. Cependant, cette technique ne permet pas de traiter une requête d'une manière parallèle sur les ontologies composantes de la TBox-décomposante. Nous avons proposé un second algorithme, dit inférence parallèle, permettant combler cette lacune. Cependant, l'algorithme parallèle nécessite éventuellement une phase de fusion des tableaux composants qui pourrait être coûteuse. Une étude approfondie d'une méthode d'approximation des requêtes mérite d'être poursuivie pour réduire voire supprimer la phase de fusion. Les deux algorithmes nous permettent de traiter toutes les requêtes locales et même globales et de retourner les mêmes résultats du raisonnement sur la TBox originelle. Surtout, ils sont réellement efficients pour les requêtes qui peuvent être traitées localement.

5. **Définition des critères d'une *bonne décomposition* overlay**
   Il est très important d'étudier dans une deuxième étape, les méthodes de décomposition universelles permettant l'obtention d'une TBox-décomposante de la décomposition overlay. Nous avons étudié tout d'abord les critères d'une bonne décomposition overlay en nous basant sur des paramètres qui influencent la complexité des algorithmes de raisonnement sur la TBox-décomposante. De plus, ils sont également confirmés dans le problème de partition de graphe. Concrètement, ces critères sont de *minimiser la partie commune entre les TBox composantes (minimiser le nombre des règles de pont)* et *équilibrer le*

cardinal d'axiomes entre les TBox composantes.

6. **Définition de la méthodologie G-décomposition**
   Les études des critères de bonne décomposition overlay nous a mené à l'idée de l'utilisation des techniques de décomposition des graphes dans la définition de la décomposition overlay. Nous avons défini une méthodologie de décomposition basée sur la partition de graphe s'appelant G-décomposition. Cette méthodologie se compose de trois phases. La première est de transformer une TBox en un graphe (graphe de symbole ou graphe d'axiome). La deuxième est de décomposer ce graphe en sous-graphes représentés par un graphe d'intersection. La troisième est de transformer le graphe d'intersection en une TBox distribuée dans laquelle chaque sous-TBox correspond à un sommet et les règles de pont correspondent aux arêtes du graphe d'intersection. La méthodologie G-décomposition peuvent être appliquée à toutes TBox de LD.

7. **Proposition de l'algorithme de G-décomposition basé sur le séparateur minimal**
   L'étude approfondie de certains éléments de la théorie des graphes nous a permis de déterminer deux techniques de décomposition des graphes correspondant aux attentes de la méthodologie G-décomposition. Il s'agit, en premier, de la technique de décomposition des graphes par le séparateur minimal. À partir de cette technique, nous avons défini le premier algorithme de G-décomposition. Dans une première phase de la méthodologie, nous avons défini un type de graphes, appelés graphes de symbole, pour représenter d'une manière schématique une TBox en retenant les propriétés qui nous s'intéressent. En particulier l'interconnexion des concepts et rôles d'un axiome sous forme de cliques. Dans la deuxième phase de la méthodologie, nous appliquons l'algorithme de décomposition arborescente du graphe de symbole par le séparateur minimal avec la largeur arborescente. Le séparateur minimal s'adapte au premier critère de décomposition défini ci-dessus. La largeur arborescente exprime la satisfaction du deuxième critère de décomposition.

8. **Proposition de l'algorithme de G-décomposition basé sur la coupe normalisée**
   Une deuxième technique de décomposition de graphe qui a retenu notre attention est la décomposition par la coupe normalisée issue de la théorie spectrale des graphes. En appliquant cette technique, nous avons défini un deuxième algorithme de G-décomposition. Dans la première phase de G-décomposition, nous avons défini un autre type de graphes, appelés graphes d'axiome, pour représenter une TBox. Dans ce graphe, chaque sommet représente un axiome, l'ensemble des poids sur les arêtes (reliant deux sommets) est une mesure de la similarité (association) entre deux sommets. Dans la deuxième phase de G-décomposition, l'idée de l'algorithme est de calculer une coupe normalisée du graphe représenté par une matrice valuée. La recherche d'une décomposition optimale est vue comme la minimisation de cette coupe normalisée qui a un double objectif. Le premier

est de minimiser le degré d'association entre les sous-graphes. Le deuxième est de maximiser le degré d'association dans chaque sous-graphe. Ces objectifs sont équivalents avec deux critères de décomposition ci-dessus. Nous avons minimisé la coupe normalisée en résolvant le système de valeur propre généralisé.

Discussion et limites

- **La complexité des algorithmes de raisonnement n'est pas réduite dans le cas général**
  En général, notre approche de décomposition de l'ontologie ne réduit pas la complexité du raisonnement. Dans le pire des cas, le raisonnement sur l'ontologie-décomposante tient le temps et l'espace comme sur l'ontologie originelle. Comme nous l'avons remarqué dans le chapitre 8, une condition préalable pour assurer une bonne décomposition dépend de la topologie du graphe représentant l'ontologie.
- **Les propriétés de la décomposition overlay sont fortes**
  Dans l'approche de décomposition overlay, la supposition forte qui consiste à garder tous les concepts et tous les rôles de l'ontologie originelle, peut conduire à un coût appréciable pour le maintien et le développement de l'ontologie.
- **Il manque un critère pour comparer les résultats obtenus par deux algorithmes de décomposition**
  Il nous manque un critère pour comparer généralement les résultats obtenus par les deux algorithmes de décomposition. Si le graphe de symbole représentant une TBox est une clique, alors il n'est pas divisible par la méthode basée sur le séparateur minimal, parce que dans ce cas, tous les sommets sont liés deux à deux, donc tous les sommets ont le même *degré d'association*. Cependant, dans ce cas, la méthode basée sur la coupe normalisée est encore applicable car la coupe dépend des valeurs des poids assignés sur les arêtes du graphe.
- **Le graphe de symbole et graphe d'axiome n'expriment pas complètement la sémantique des axiomes et la relation sémantique entre eux**
  Comme nous en avons discuté dans le chapitre 8, dans la méthode basée sur la coupe normalisée, le calcul de la mesure d'association entre deux axiomes n'est pas seulement basé sur la relation entre le nombre des symboles communs et le nombre total des symboles de deux axiomes, mais il doit également prendre en compte la présence des constructeurs de concept et de rôle dans les langages de LD utilisés. Par exemple, soit trois axiomes $A_1 : C_1 \sqcap \exists R.C_2 \sqsubseteq C_3$, $A_2 : C_2 \sqsubseteq C_4$, $A_3 : C_1 \sqsubseteq C_5$, alors c'est mieux si on peut donner une fonction w telle que $w_{12} \neq w_{13}$.

Dans le cadre des travaux de cette thèse, nous n'avons pas étudié les techniques d'optimisation

au niveau de requête.

Perspectives

Le travail effectué dans ce mémoire peut être poursuivi dans plusieurs directions. Dans cette section, nous présentons les possibilités d'étendre et d'approfondir les recherches suivantes :

- **Étude des techniques d'optimisation de raisonnement sur le niveau de requête**
  Pour le niveau de requête, nous proposons une direction de recherche qui peut s'effectuer avec une TBox-décomposante : la *décomposition de requête*. Une requête posée est réécrite en une expression conjonctive [GR00]. Ensuite, elle pourrait être décomposée en des *sous-requêtes* (chaque sous-requête est un conjonct de cette conjonction) et traitée localement sur les TBox composantes, les résultats seront combinés pour obtenir une réponse finale. Néanmoins, ceci est un problème incomplet parce qu'on ne traite que les requêtes dans la forme d'une expression conjonctive.
- **Recherche des paramètres appropriés pour calculer les coupes de graphe représentant l'ontologie**
  Les paramètres appropriés pour calculer les coupes de graphe de symbole et de graphe d'axiome peuvent améliorer les techniques de décomposition de l'ontologie.
- **Réduire les contraintes de définition de la décomposition overlay**
  Nous pouvons réduire les contraintes de définition de la décomposition overlay à une supposition que les sous ontologies ne contiennent que des concepts et des rôles qui apparaissent dans les expressions de leurs axiomes. Ceci nécessite une autre stratégie pour développer les algorithmes de tableau tels qu'ils peuvent assurer les résultats de préservation comme indiqués dans le travail de ce mémoire ou ils peuvent obtenir l'amélioration de la performance des algorithmes du raisonnement.
- **Traitement avec plusieurs ontologies composantes dans la décomposition**
  Dans le cadre de cette thèse, nous avons traité le cas plus simple avec deux ontologies composantes dans la décomposition. Le travail dans le futur devra traiter le cas général de décomposition avec n (n>2) ontologies composantes. Nous pouvons appliquer les algorithmes de raisonnement proposés avec la condition que le graphe d'intersection soit un arbre. Cette condition vise à éviter l'application cyclique de propagation des règles de pont.
- **Développement de nouvelles méthodes de décomposition**
  Enfin, il est nécessaire de développer de nouvelles méthodes de décomposition de l'ontologie qui puissent améliorer la performance des algorithmes de raisonnement.

Bibliographie

[AFWZ02] A. Artale, E. Franconi, F. Wolter, and M. Zakharyaschev. A temporal description logic for reasoning about conceptual schemas and queries. In *Proceedings of the JELIA-02, volume 2424 of LNAI*, pages 98–110, Spinger, 2002.

[AM05] E. Amir and S. McIlraith. Partition-based logical reasoning for first-order and propositional theories. In *Artificial Intelligence, Volume 162*, pages 49–88, February 2005.

[Baa90] F. Baader. Terminological cycles in KL-ONE-based knowledge representation languages. In *Proceedings of the Eighth National Conference on Artificial Intelligence, AAAI-90*, pages 621–626, Boston (USA), 1990.

[Baa96] F. Baader. Logic-based knowledge representation. *KI*, 3/96 :8–16, 1996.

[BB02] A. Borgida and R. J. Brachman. *Conceptual Modelling with Description Logics*. In Description Logic Handbook, F Baader, D. Calvanese, D. McGuinness, D. Nardi, D. P. Patel-Schneider (Eds.), Cambridge University Press, 349-372, 2002.

[BBC99] A. Berry, J.P. Bordat, and O. Cogis. Generating all the minimal separators of a graph. In *Workshop on Graph-theoretic Concepts in Computer Science (WG'99), volume 1665 of Lecture Notes in Computer Science*, pages 167–172, Springer-Verlag, 1999.

[BBLS94] Domenico Beneventano, Sonia Bergamaschi, Stefano Lodi, and Claudio Sartori. Terminological logics for schema design and query processing in OODBs. In *Knowledge Representation Meets Databases*, 1994.

[BCH06a] J. Bao, D. Caragea, and V. Honavar. A distributed tableau algorithm for package-based description logics. In *Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence*, pages 404–410, 2006.

[BCH06b] J. Bao, D. Caragea, and V. Honavar. On the semantics of linking and importing in modular ontologies. *I. Cruz et al. (Eds.) : ISWC 2006, LNCS 4273*, pages 72–86, 2006.

[BDS93] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, pages 1 :109–138, 1993.

[BFT95] Paolo Bresciani, Enrico Franconi, and Sergio Tessaris. Implementing and testing expressive description logics : a preliminary report. In *Proceedings of DL-95, 4th International Workshop on Description Logics*, pages 131–139, Roma, IT, 1995.

[BHN+93] F. Baader, B. Hollunder, B. Nebel, H.J. Profitlich, and E. Franconi. An empirical analysis of optimization techniques for terminological representation systems, or : Making KRIS get a move on. DFKI Research Report RR-93-03, Deutsches Forschungszentrum für Künstliche Intelligenz, Kaiserslautern, 1993.

[BHS02] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the semantic web. In *KI Künstliche Intelligenz*, 4, 2002.

[BKD+02] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. Enabling knowledge representation on the web by extending RDF schema. *Computer Networks*, 39(5) :609–634, 2002.

[BL84] Ronal J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *Proc. of the 4th National Conference on Artificial Intelligence (AAAI-84)*, pages 34–37, 1984.

[BL99] T Berners-Lee. *Weaving the Web*. Harpur, San Francisco, 1999.

[BLSW00] F. Baader, C. Lutz, H. Sturm, and F. Wolter. Fusions of description logics. In F. Baader and U. Sattler, editors, *Proceedings of the International Workshop in Description Logics 2000 (DL2000)*, number 33 in CEUR-WS, pages 21–30, Aachen, Germany, August 2000. RWTH Aachen. Proceedings online available from http ://SunSITE.Informatik.RWTH-Aachen.DE/Publications/CEUR-WS/Vol-33/.

[BMD+94] Buchheit, Martin, F.M. Donini, W. Nutt, and A. Schaerf. Terminological system revisited : Terminology = schema + views. In *Proc. of the 12th Nat. Conf. on Artificial Intelligence (AAAI-94), 199-204. Seattle, USA*, 1994.

[BN03] F. Baader and W. Nutt. *Basic Description Logics*. Spinger, 2003.

[Bra78] R. J. Brachman. Structured inheritance networks. In *Research in Natural language understanding*, 1978.

[BS85] R. J. Brachman and J. G. Schmolze. An overview of the kl-one knowledge representation system. In *Cognitive Science*, pages 9(2) :171–216, 1985.

[BS01] F. Baader and U. Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69 :5–40, 2001.

[BS03] A. Borgida and L. Serafini. Distributed description logics : Assimilating information from peer sources. *Journal of Data Semantics*, pages 1 :153–184, 2003.

[Cal96] Diego Calvanese. Reasoning with inclusion axioms in description logics : Algorithms and complexity. In *European Conference on Artificial Intelligence*, pages 303–307, 1996.

[CH94] A. Cournier and M. Habib. A new linear algorithm for modular decomposition. In *Proceedings of Trees in Algebra and Programming - CAAP'94, volume 787 of Lecture Notes in Computer Science*, pages 64–84, 1994.

[Chu97] Fan R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[CLN98] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. *Logics for Databases and Information Systems, J. Chomicki et G. Saake (Eds.)*, pages 229–264, Kluwer Academic Publisher, 1998.

[CY92] F. Cuppens and K. Yazdanian. A "natural" decomposition of multi-level relations. In *Proceedings of the 1992 IEEE Symposium on Security and Privacy*, page 273, 1992.

[DHZ+01] Chris H. Q. Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of ICDM 2001*, pages 107–114, 2001.

[DLNS96] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Reasoning in description logics. In Gerhard Brewka, editor, *Foundation of Knowledge Representation*, pages 191–236. CSLI-Publications, 1996.

[Eve79] S. Even. *Graph Algorithms*. Computer Science Press, 1979.

[FF62] Jr. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[Fra02] E. Franconi. Description logics for conceptual design, information access, and ontology integration. *PhD course*, Copenhagen, 2002.

[Fra03] M. Donini. Francesco. *Complexity of Reasoning*. Description Logic Handbook, 2003.

[Fre95] J. W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1995.

[Fre96] J. W. Freeman. Hard random 3-sat problems and davis-putnam procedure. *Artificial Intelligence*, pages 81 :183–198, 1996.

[Gav74] F. Gavil. The intersection graphs of a path in a tree are exactly the chordal graphs. *Journal of Combinatorial Theory*, pages 16 :47–56, 1974.

[GPS04a] B. C. Grau, B. Parsia, and E. Sirin. Working with multiple ontologies on the semantic web. In *International Semantic Web Conference*, pages 620–634, 2004.

[GPS04b] B. C. Grau, B. Parsia, and E. Sirin. Tableau algorithms for e-connections of description logics. Technical report, University of Maryland Institute for Advanced Computer Studies, Technical Report, (UMIACS), TR-2004-72, 2004.

[GR93] G. Grahne and K. J. Räihä. Database decomposition into fourth normal form. In *Proceedings of the 9th International Conference on Very Large Data Bases*, 1993.

[GR00] F. Goasdou'e and M. Rousset. Rewriting conjunctive queries using views in description logics with existential restrictions, 2000.

[Gru93] T. Gruber. A translation approach to portable ontology specifications. In *Knowledge Acquisition*, pages 5(2) :199–220, 1993.

[GS98] C. Ghidini and L. Serafini. Distributed first order logics. *D. Gabbay and M. de Rijke, editors, Frontiers Of Combining Systems 2, Studies in Logic and Computation*, pages 121–140, Research Studies Press, 1998.

[GVL89] G. H. Golub and C. F. Van Loan. *Matrix computations*. John Hopkins Press, 1989.

[Hay79] P. J. Hayes. The logic of frames. *Frames Conception and Text Understanding*, Walter de Gruyter and Co., 1979.

[HNSS90] B. Hollunder, W. Nutt, and M. Schmidt-Schauß. Subsumption algorithms for concept description languages. In *ECAI*, pages 348–353, 1990.

[Hor97] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[Hor98] Ian Horrocks. Using an expressive description logic : FaCT or fiction ? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[Hor02] Ian Horrocks. DAML+OIL : a description logic for the semantic web. *Bull. of the IEEE Computer Society Technical Committee on Data Engineering*, 25(1) :4–9, March 2002.

[HPS98] I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Proc. of the Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX 1998)*, pages 27–30, 1998.

[HPS99a] Ian Horrocks and P. F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3) :267–293, 1999.

[HPS99b] Ian Horrocks and Peter F. Patel-Schneider. Optimizing description logic subsumption. *J. of Logic and Computation*, 9(3) :267–293, 1999.

[Jun99] D. Jungnickel. *Graphs, Networks and Algorithms*. Spinger, 1999.

[KK98] T. Kloks and D. Kratsch. Listing all minimal separators of a graph. *SIAM J. Comput.*, pages 27(3) :605–613, 1998.

[KLWZ03] O. Kutz, C. Lutz, F. Wolter, and M. Zakharyaschev. E-connections of description logics. In *Proceedings of the International Workshop on Description Logics (DL-2003)*, pages 178–187, Rome, 2003.

[KWZ02] O. Kutz, F. Wolter, and M. Zakharyaschev. Connecting abstract description systems. In *Proceedings of the 8th Conference on Principles of Knowledge Representation and Reasoning (KR 2002)*, pages 215–226. Morgan Kaufmann, 2002.

[LB87] H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, pages 3 :78–93, 1987.

[LD04] C. Le-Duc. *Transformation d'Ontologies basées sur la Logique de Description - Application dans le Commerce Electronique*. PhD thesis, Université de Nice - Sophia Antipolis, 2004.

[Mac91] R. MacGregor. The evolving technology of classification-based knowledge representation systems. In *Principles of Semantic Networks*, pages 385–400, Morgan Kaufmann, Los Altos, 1991.

[Maz04] F. Mazoit. *Décompositions algorithmiques des graphes*. PhD thesis, L'Ecole normale supérieure de Lyon, 2004.

[MDW91] E. Mays, R. Dionne, and R. Weida. K-rep system overview. In *SIGART Bulletin*, 2(3), 1991.

[Min75] M. Minsky. A framework for representing knowledge. *The Psychology of Computer Vision. McGraw-Hill*, 1975.

[Neb88] B. Nebel. Computational complexity of terminological reasoning in back. *Journal of Artificial Intelligence*, pages 34(3) :371–383, 1988.

[Neb90] B. Nebel. Teminological reasoning is inherently intractable. In *Artificial Intelligence*, pages 43 :235–249, 1990.

[Neb91] B. Nebel. Terminological cycles : Semantics and computational properties. In *Principles of Semantic Networks*, pages 331–362. ed. John F. Sowa. Morgan Kaufmann, San Mateo, CA, 1991.

[Pel91] C. Peltason. The back system - an overview. In *SIGART Bulletin*, pages 2(3) :114–119, 1991.

[PLT06] T. A. L. Pham and N. Le-Thanh. Decomposition-based reasoning for large knowledge bases in description logics. In *Proceedings of the 13th ISPE International*

Conference on Concurrent Engineering : Research and Applications, pages 288–295, France, September 2006.

[PLT07] T. A. L. Pham and N. Le-Thanh. Some approaches of ontology decomposition in description logics. In *Proceedings of the 14th ISPE International Conference on Concurrent Engineering : Research and Applications*, pages 534–542, Brasil, July 2007.

[Pri94] C. Prins. *Algorithmes de graphes*. Eyrolles, 1994.

[PTS08] T. A. L. Pham, N. Le Thanh, and P. Sander. Decomposition-based reasoning for large knowledge bases in description logics. *Integrated Computer-Aided Engineering*, 15(1/2008) :53–70, 2008.

[PVG96] Tai Joon Park and Allen Van Gelder. Partitioning methods for satisfiability testing on large formulas. In *Proceedings 13th International Conference on Automated Deduction*, volume 1104 of *LNAI*, pages 748–762. Springer-Verlag, July 1996.

[Qui68] M. Quillian. Semantic memory. In *Semantic Information Processing*, MIT Press, Cambridge, 1968.

[SBT05] L. Serafini, A. Borgida, and A. Tamilin. Aspects of distributed and modular ontology reasoning. In *IJCAI*, pages 570–575, 2005.

[SG92] K. Shoikhet and D. Geiger. Finding optimal triangulations via minimal vertex separators. In *Proceedings of the 3rd International Conference*, pages 270–281, Cambridge, MA, October 1992.

[SM00] J. Shi and J. Malik. Normalized cuts and image segmentation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8)*, pages 888–905, August 2000.

[Smu68] R. M. Smullyan. First-order logic. Springer-Verlag, Berlin,1968.

[Sow84] J. F. Sowa. Conceptural structures : Information processing in mind and machine. *Addison Wesley*, 1984.

[Spi04] Daniel A. Spielman. Spectral graph theory and its applications. Lecture, 2004.

[SSS91] M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, pages 48 :1–26, 1991.

[ST04a] L. Serafini and A. Tamilin. Drago : Distributed reasoning architecture for the semantic web. Technical report, Technical Report T04-12-05,ITC-irst, 2004.

[ST04b] L. Serafini and A. Tamilin. Local tableaux for reasoning in distributed description logics. In *Description Logics Workshop 2004, CEUR-WS Vol 104*, 2004.

[THPS07] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimising terminological reasoning for expressive description logics. *J. of Automated Reasoning*, 2007. To appear.

[Tod06] I. Todinca. Décompositions arborescentes de graphes : calcul, approximations, heuristiques. Habilitation à diriger des recherches, Université d'Orléans, le 1er décembre 2006.

[VJF98] F. Volot, M. Joubert, and M. Fieschi. Review of biomedical knowledge and data representation with conceptual graphs. *Methods of Information in Medicine*, 37(1) :86–96, 1998.

[Woo75] W. A. Woods. What's in a link : foundations for semantic networks. *Representation and Understanding : Studies in Cognitive Science*, pages 35–82, Academic Press, London, 1975.

Annexe

La TBox Tambis1 est présentée dans le fichier .XML suivant :

```
<?xml version="1.0" encoding="UTF-8" ?>
<KNOWLEDGEBASE>
<DEFROLE NAME="Attribute"/>
<DEFROLE NAME="DomainAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="DomainAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="Attribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="InverseDomainAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="InverseDomainAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="Attribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="hasPrecedence"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="hasPrecedence"/>
```

```
</ROLE>
<ROLE>
<PRIMROLE NAME="Attribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="isPrecedenceOf"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="isPrecedenceOf"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="Attribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="ConstructiveAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="ConstructiveAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="DomainAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="inverseConstructiveAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="inverseConstructiveAttribute"/>
```

```
</ROLE>
<ROLE>
<PRIMROLE NAME="InverseDomainAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="StructuralAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="StructuralAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="ConstructiveAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="inverseStructuralAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="inverseStructuralAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseConstructiveAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="FunctionalAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="FunctionalAttribute"/>
```

```
</ROLE>
<ROLE>
<PRIMROLE NAME="ConstructiveAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="inverseFunctionalAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="inverseFunctionalAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseConstructiveAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="NumericalAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="NumericalAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="ConstructiveAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="inverseNumericalAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="inverseNumericalAttribute"/>
```

```
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseConstructiveAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="hasIdentifier"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="hasIdentifier"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="NumericalAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="isIdentifierOf"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="isIdentifierOf"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseNumericalAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="hasAccessionNumber"/>
<FUNCTIONAL>
<PRIMROLE NAME="hasAccessionNumber"/>
</FUNCTIONAL>
```

<IMPLIESR>

<ROLE>

<PRIMROLE NAME="hasAccessionNumber"/>

</ROLE>

<ROLE>

<PRIMROLE NAME="hasIdentifier"/>

</ROLE>

</IMPLIESR>

<DEFROLE NAME="isAccessionNumberOf"/>

<FUNCTIONAL>

<PRIMROLE NAME="isAccessionNumberOf"/>

</FUNCTIONAL>

<IMPLIESR>

<ROLE>

<PRIMROLE NAME="isAccessionNumberOf"/>

</ROLE>

<ROLE>

<PRIMROLE NAME="isIdentifierOf"/>

</ROLE>

</IMPLIESR>

<DEFROLE NAME="hasNumberOfSubunits"/>

<FUNCTIONAL>

<PRIMROLE NAME="hasNumberOfSubunits"/>

</FUNCTIONAL>

<IMPLIESR>

<ROLE>

<PRIMROLE NAME="hasNumberOfSubunits"/>

```
</ROLE>
<ROLE>
<PRIMROLE NAME="NumericalAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="isNumberOfSubunitsOf"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="isNumberOfSubunitsOf"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseNumericalAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="ClassificationAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="ClassificationAttribute"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="StructuralAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="inverseClassificationAttribute"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="inverseClassificationAttribute"/>
```

```
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseStructuralAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="isStructuralClassificationOf"/>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="isStructuralClassificationOf"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="ClassificationAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="hasStructuralClassification"/>
<FUNCTIONAL>
<PRIMROLE NAME="hasStructuralClassification"/>
</FUNCTIONAL>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="hasStructuralClassification"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseClassificationAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="hasStructuralClassificationNumber"/>
```

<FUNCTIONAL>

<PRIMROLE NAME="hasStructuralClassificationNumber"/>

</FUNCTIONAL>

<IMPLIESR>

<ROLE>

<PRIMROLE NAME="hasStructuralClassificationNumber"/>

</ROLE>

<ROLE>

<PRIMROLE NAME="hasStructuralClassification"/>

</ROLE>

</IMPLIESR>

<DEFROLE NAME="isStructuralClassificationNumberOf"/>

<FUNCTIONAL>

<PRIMROLE NAME="isStructuralClassificationNumberOf"/>

</FUNCTIONAL>

<IMPLIESR>

<ROLE>

<PRIMROLE NAME="isStructuralClassificationNumberOf"/>

</ROLE>

<ROLE>

<PRIMROLE NAME="isStructuralClassificationOf"/>

</ROLE>

</IMPLIESR>

<DEFROLE NAME="isFunctionalClassificationOf"/>

<IMPLIESR>

<ROLE>

<PRIMROLE NAME="isFunctionalClassificationOf"/>

```
</ROLE>
<ROLE>
<PRIMROLE NAME="ClassificationAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="hasFunctionalClassification"/>
<FUNCTIONAL>
<PRIMROLE NAME="hasFunctionalClassification"/>
</FUNCTIONAL>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="hasFunctionalClassification"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="inverseClassificationAttribute"/>
</ROLE>
</IMPLIESR>
<DEFROLE NAME="hasECNumber"/>
<FUNCTIONAL>
<PRIMROLE NAME="hasECNumber"/>
</FUNCTIONAL>
<IMPLIESR>
<ROLE>
<PRIMROLE NAME="hasECNumber"/>
</ROLE>
<ROLE>
<PRIMROLE NAME="hasFunctionalClassification"/>
```

```
</ROLE>

</IMPLIESR>

</KNOWLEDGEBASE>
```