Transformation d'ontologies basées sur la logique de description : Application dans le commerce électronique

Chan Le Duc

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

Unité de Formation et de Recherche Sciences

École doctorale : « Sciences et Technologies de l'Information et de la Communication »

THÈSE

Présentée pour obtenir le titre de
Docteur ès SCIENCES
de l'Université de Nice-Sophia Antipolis

Spécialité : INFORMATIQUE

par

Chan LE DUC

Transformation d'Ontologies basées sur la Logique de Description Application dans le Commerce Électronique

---

Thèse dirigée par Nhan LE THANH

Soutenue publiquement le 6 juillet 2004 devant le jury composé de :

M. Jean-Marc Fédou	Professeur à l'UNSA	Président
Mme Danièle Hérin	Professeur à l'Université Montpellier II	Rapporteur
Mme Marie-Christine Rousset	Professeur à l'Université Paris XI	Rapporteur
M. Nhan Le Thanh	Professeur à l'UNSA	Directeur
Mme Rose Dieng	Directeur de recherche à l'INRIA	Examinatrice
M. Alain Zarli	Chercheur au CSTB	Examinateur
M. Patrick Morand	Responsable du SIA-TIDS du CSTB	Invité

À ma famille.
À Lan Huong et Thê Dân.

Remerciements

Cette thèse a été financée par le Centre Scientifique et Technique du Bâtiment (CSTB) où une partie de ce travail a été réalisée. Je tiens à remercier Monsieur Bertrand Delcambre, directeur de l'établissement du CSTB Sophia Antipolis, Messieurs Roger Pelletret et Patrick Morand, les responsables du service SIA, de m'avoir accueilli au sein de leur unité. Je remercie également Eric Gaduel, Bruno Fies et Jérôme Zattoni qui m'ont aidé pendant mes premiers jours au CSTB.

L'autre partie de ce travail a été réalisée au sein du projet EXeCO au Laboratoire Informatique Signaux et Systèmes de Sophia Antipolis. Je tiens à remercier tous les membres du Laboratoire qui m'ont aidé à m'intégrer, en particulier l'équipe Langages qui m'a fourni un espace de travail.

Je voudrais exprimer ma gratitude à Monsieur Nhan Le Thanh qui m'a laissé beaucoup de liberté dans la recherche et m'a permis, avec beaucoup de bienveillance, de mener à bien cette thèse.

Je remercie Monsieur Jean-Marc Fédou, professeur à l'UNSA, pour m'avoir fait l'honneur de présider le jury.

Je tiens à remercier Mesdames Danièle Hérin, professeur à l'université Montpellier II, et Marie-Christine Rousset, professeur à l'université Paris Sud d'Orsay, qui m'ont fait l'honneur d'accepter d'être rapporteur de cette thèse. En particulier, je voudrais exprimer toute ma reconnaissance à Madame Marie-Christine Rousset qui a bien voulu discuter ce travail et m'a donné de précieux conseils pour améliorer la présentation de la thèse.

Je remercie également Madame Rose Dieng, directeur de recherche à l'INRIA, d'avoir accepté d'examiner ce mémoire.

Je tiens à remercier Monsieur Alain Zarli, chercheur au CSTB, d'avoir accepté d'examiner ce mémoire et Monsieur Patrick Morand, responsable du service SIA-TIDS du CSTB, d'avoir accepté de participer au jury.

Je remercie également toutes les personnes qui ont contribué directement ou indirectement à la rédaction de ce mémoire. Je pense en particulier à Olivier Ponsini pour avoir répondu à toutes mes questions sur la langue française, relu plusieurs versions de ce mémoire et pour le temps qu'il m'a consacré ; à Alain Kibangou pour avoir relu ce mémoire et pour ses conseils avisés.

Enfin, je voudrais témoigner à Lan Huong, ma femme et à Thê Dân, notre fils, mon infinie tendresse et les remercier de leur patience et de leur soutien.

Résumé

Ce travail s'inscrit dans le cadre de la formalisation des connaissances pour l'échange de données dans le domaine du Commerce Électronique. Cette formalisation basée sur la Logique de Description (LD) a pour objectif d'établir la transparence sémantique de l'échange de données entre des acteurs de profil différent. Lorsque les connaissances sont formalisées dans les ontologies des acteurs, la transparence sémantique de l'échange de données peut être assurée par la transformation des ontologies en question. Généralisées à partir des problèmes posés dans les modèles d'échanges en usage, deux instances du problème de transparence sémantique (PTS) sont formulées comme des inférences permettant de transformer des ontologies. La première instance du problème porte sur l'échange de données sans contexte entre deux acteurs qui utilisent deux ontologies représentées par deux langages LD différents. Cette instance peut se réduire au calcul d'équivalence ou d'approximation sémantique d'un terme défini dans l'ontologie de l'expéditeur et ce terme est inexprimable dans l'ontologie du destinataire. En se basant sur un algorithme existant pour ce calcul, un optimal algorithme qui améliore la performance de l'algorithme existant en terme d'espace, est introduit. La deuxième instance du problème porte sur l'échange de données dans lequel les informations contextuelles représentées comme les règles de contexte sont prises en compte. Afin de formaliser ces règles de contexte, la révision d'une ontologie et les règles de révision devraient être introduites dans les ontologies. Une partie importante de ce travail étudie la révision d'une ontologie basée sur la LD et développe les algorithmes pour les opérations de révision et pour la procédure d'extension de la base de connaissances déclenchée par les règles de révision. Enfin, le projet ONDIL (ONtology DesIgn Layers) est présenté comme un prototype qui implémente les algorithmes développés et dont un champs d'application est la conception et la maintenance des ontologies dans le secteur de la construction.

Mots clés : représentation de connaissances, logique de description, commerce électronique, ontologie, approximation, révision de la terminologie, règles de contexte, règles de révision.

Abstract

This work deals with knowledge formalization for data exchange in Electronic Commerce area. This formalization which is based on Description Logics (DL) is aimed at establishing semantic transparency for data exchange between different agents. When knowledges are formalized in ontologies of agents, the semantic transparency would be ensured by ontology transformation. Generalized from problems which arise in data exchange models in use, two instances of the semantic transparency problem are identified and formalized as inferences allowing to transform ontologies. The first instance of the problem arises when two agents carry out data exchanges in which their ontologies use different DL. This instance can be reduced to semantic equivalence or approximation which allows us to compute the best approximation of a concept description expressed in a more expressive DL. Based on an existing algorithm for computing the approximation $\mathcal{ALC}$-$\mathcal{ALE}$, we propose an optimal algorithm which improves the performance of the existing algorithm in space. The second instance of the problem arises when context information is taken into account in data exchange processes. In order to formalize context information implied in actual data models, revision operations and revision rules should be introduced into ontologies. An important part of this work is concentrated on revision problems of DL-based ontologies and develops algorithms for computing revision operations and for performing extension of knowledge base launched by revision rules. The algorithms which are developed in this work are implemented in ONDIL system supporting design and maintenance of ontologies used in construction sector.

Keywords : knowledge representation, description logic, electronic commerce, ontology, approximation, terminology revision, context rule, revision rule.

Table des matières

Introduction

De nos jours, l'apparition des NTIC[1] permet de remplacer progressivement la communication sur papier des données administratives, commerciales, techniques, *etc.* par une version électronique. La nouvelle manière d'échanger des données a besoin non seulement de réorganiser la procédure et le contenu d'échange mais aussi de les formaliser. En effet, d'une part il est souhaitable que les données échangées soient toujours lisibles par l'homme pour des raisons de vérification et de législation, donc les documents structurés et composés de termes du langage naturel sont nécessairement conservés ne serait-ce que pour la présentation de données. D'autre part, l'automatisation du traitement de ces documents exige d'expliciter et de formaliser les *connaissances*, c'est-à-dire le protocole d'échange, les vocabulaires utilisés et les règles d'interprétation. De plus, l'échange de données se déroule entre des domaines ou sous-domaines d'application, définis comme acteurs, dont les activités peuvent être complexes et variées. Les informations sur les statuts des acteurs définissent un *contexte d'échange* qui permet de déterminer exactement la signification des données reçues. Cela explique pourquoi plusieurs formalismes doivent être utilisés dans le système d'échange existant pour capturer des connaissances de nature très diverse. En outre, certains formalismes envisagés disposent d'une sémantique informelle c'est-à-dire qu'elle n'est pas traitable par la machine. Cela empêche la vérification de la cohérence sémantique du système. Plus important, si ce dernier emploie plusieurs formalismes, la compatibilité sémantique entre eux est indispensable. Cette compatibilité peut être assurée par un formalisme commun avec la sémantique formelle, appelé *formalisme hybride*, en lequel les autres formalismes se traduisent en conservant la sémantique de chacun. Le formalisme hybride doit être capable non seulement de représenter les connaissances hétérogènes du domaine mais aussi d'effectuer les inférences nécessaires pour prendre en compte le contexte d'échange. Une des approches dans ce sens est la construction d*'ontologies* [**Fen01 (34)**]. En général, les ontologies sont développées pour fournir aux sources d'information une sémantique traitable par la machine. Dans le contexte du système d'échange, une ontologie disposant d'un formalisme expressif permettra de modéliser de façon cohérente des informations de nature diverse en fonction du domaine de l'acteur.

Néanmoins, l'utilisation de telles ontologies ne répond pas suffisamment aux problèmes qu'un système d'échange de données pose. Un des problèmes essentiels est la *transparence sémantique* entre les ontologies, c'est-à-dire que des données transmises par un acteur doivent être *interprétables* par un autre acteur du système.

[1]Nouvelles Technologies de l'Information et de la Communication

Un certain nombre d'études ont été réalisées au cours des dernières années quant au domaine du Commerce Électronique, en particulier avec le développement de standards tels que XML/EDI[2], ebXML[3] et FLBC[4]. Cependant, ces actions ne répondent pas ou partiellement aux problèmes cités. Le premier ne se préoccupe pas de la sémantique des documents échangés car il étend simplement le standard EDI déjà connu en donnant aux messages une représentation en XML. Les lacunes du premier ont poussé de grandes entreprises industrielles à proposer le second qui est un canevas plus complet afin d'améliorer la communication entre acteurs différents. Toutefois, les problèmes majeurs persistent : plusieurs formalismes (diagrammes UML, règles de contexte, *etc.*) sont utilisés de façon indépendante et la formalisation de la sémantique n'est pas prise en compte. Quant au troisième, les auteurs souhaitaient des modifications profondes sur la manière de traiter la sémantique. Ils ont proposé d'utiliser les dérivés modaux de la *Logique du Premier Ordre* pour formaliser la sémantique des langages utilisés dans le commerce. Afin de devenir une concurrente des autres standards, cette approche doit surmonter au moins deux inconvénients : un lexique unique est construit pour tout le commerce et la calculabilité des dérivés modaux proposés n'est pas encore suffisamment étudiée.

Afin de rendre les idées plus concrètes, un exemple simple, intuitif et informel pourrait être approprié pour démontrer la nécessité de la sémantique formelle et de la transparence sémantique.

Exemple. (*Exemple d'Introduction*)
- Définitions partagées :
  (1) **EntrepriseEurop** est un ensemble d'**Entreprises** qui ont au moins un *associé* **Européen**.
  (2) **EntrepriseAmér** est un ensemble d'**Entreprise**s qui ont au moins un *associé* **Américain**.
  (3) **ProdRésAuPolluant** (Produit Résistant Au Polluant) est un **Produit**.
- Acteur A_EUROPEEN : **ProdRésAuPolluant** est un **Produit** résistant à la fois au **Bruit** et au **Feu**.
- Acteur A_AMERICAIN : **ProdRésAuPolluant** est un **Produit** résistant au **Bruit**.
- Echange : l'acteur A_EUROPEEN qui est un **EntrepriseEurop**, *vend* un produit PRODUIT-A de **ProdRésAuFeu** à l'acteur A_AMERICAIN qui est un **EntrepriseAmér**.

Dans les systèmes actuels, cet échange correspond à une facture qui contient une entrée décrivant PRODUIT-A. Cette entrée peut ne pas convenir à l'acteur A_AMERICAIN car le concept **ProdRésAuFeu** est différemment défini dans son dictionnaire. Nous avons les cas suivants :

[2]eXtended Markup Language for Electronic Data Interchange
[3]Electronic Business eXtended Markup Language
[4]Formal Language for Business Communication

(1) Si le dictionnaire de A_AMERICAIN est traditionnellement organisé, une intervention humaine éventuelle permet d'interpréter la définition du terme reçu et d'identifier un terme approprié dans son dictionnaire.

(2) Si le dictionnaire de A_AMERICAIN est formalisé, la recherche de la définition qui est la plus proche de la définition reçue, peut être effectuée par la machine.

(3) En supposant que les dictionnaires soient formalisés, la règle suivante est ajoutée à la base de connaissances partagée :
   - S'il y a un acteur Y associé à un **Européen** qui *vend* un **Produit** X à un acteur Z associé à un **Américain**, alors le produit X doit être défini selon l'acteur **EntrepriseEurop** *i.e.* ce produit peut également résister au **Feu**.

   Dans ce cas, la facture est composée en utilisant la définition de **Produit** selon la norme de l'entreprise européenne. Cela implique une modification du dictionnaire de l'acteur A_AMERICAIN pour que l'échange en question puisse aboutir.

Il est évident que l'approche d'EDI n'autorise pas un tel échange. Un accord préalable entre les acteurs a pour objectif d'éviter ce type de scénarios. Un tel accord comporte les connaissances implicites qui ne sont pas traitables par la machine. Cela restreint la flexibilité du système. Par contre, l'automatisation des traitements dans le deuxième cas de l'exemple exige la formalisation des dictionnaires. Cette formalisation permettra de calculer un terme qui est sémantiquement "le plus proche" du terme reçu. Par conséquent, dans certains cas où un terme équivalent ne pourrait pas être trouvé dans le dictionnaire de l'acteur A_AMERICAIN, un calcul d'approximation pour trouver le terme le plus proche du terme reçu doit être envisagé.
En outre, dans le troisième cas de l'exemple la définition exacte du terme est déterminée par l'application de la règle. En effet,

- Par la définition 1. , A_EUROPEEN étant **EntrepriseEurop** implique que l'acteur A_EUROPEEN est associé à un **Européen**.
- Par la définition 2. , A_AMERICAIN étant **EntrepriseAmér** implique que l'acteur A_AMERICAIN est associé à un **Américain**.
- Par la définition 3. , PRODUIT-A est **Produit**.
- Par le cas de l'échange, l'acteur A_EUROPEEN *vend* le produit PRODUIT-A à l'acteur A_AMERICAIN.

Alors, la règle est appliquée et donc, les deux acteurs A_EUROPEEN et A_AMERICAIN partagent la compréhension du terme **ProdRésAuPolluant**.

Le troisième cas de l'exemple pose au moins deux problèmes supplémentaires. Le premier est l'hétérogénéité de la représentation de connaissances. Les définitions partagées évoquent un formalisme terminologique alors que les règles sont considérées comme un formalisme traditionnel pour la représentation de connaissances. Le second est la révision d'une base de connaissances pour s'adapter au contexte d'échange. Désormais, chaque référence à **ProdRésAuPolluant** dans le dictionnaire de A_AMERICAIN devrait être interprétée selon la nouvelle définition.

L'idée de l'utilisation de la Logique du Premier Ordre résultant de l'approche de FLBC [**Kim98 (45)**] pourrait être une solution au problème de la formalisation de la sémantique pour l'ebXML. Toutefois, les calculs basés sur la Logique du Premier Ordre sont en général indécidables. En se limitant aux connaissances non-modales, un fragment décidable de la Logique du Premier Ordre, appelé *Logique de Description*, peut être utilisé pour formaliser cette sémantique. De plus, le *mécanisme de contexte* [**ebX01a (26)**] venant de l'ebXML suggère une réponse au problème de transparence sémantique. Ce mécanisme est représenté dans l'ebXML par des *règles de contexte* qui sont considérées comme des règles de production spécifiques.

La formulation de la sémantique des formalismes utilisés dans les modèles de données à l'aide de la Logique de Description nous permet de traduire le problème de transparence sémantique dans le Commerce Electronique en celui de la transformation d'ontologies basées sur cette logique. Dans ce contexte, en supposant que toutes les ontologies du système dérivent d'une ontologie partagée de base, nous pouvons présenter deux instances du problème à étudier :

- Traduction précise ou approximative d'une ontologie dérivée en une autre. Ces ontologies utilisent des langages fondés sur la même sémantique formalisée par l'utilisation de la Logique de Description et les expressivités de ces langages peuvent être différentes.
- Harmonisation des ontologies dérivées en prenant en compte les règles de contexte.

Par conséquent, l'objet de cette thèse vise à étudier la Logique de Description dans le but, d'une part, de formaliser la sémantique des langages de modélisation utilisés dans ebXML, et d'autre part, d'établir un formalisme hybride dont la sémantique formelle permette de mettre en commun ces langages. Ce formalisme hybride sur lequel les services d'inférences sont développés doit être capable de répondre aux deux instances du problème mentionné.

Les objectifs de cette thèse sont les suivants :

(1) Identification d'un formalisme permettant la formalisation de la sémantique nécessaire à la construction des ontologies du Commerce Électronique .

(2) Proposition d'un formalisme hybride visant à atteindre la transparence sémantique entre les ontologies dérivées.

(3) Amélioration d'algorithmes existants ou développement de nouveaux algorithmes pour des services d'inférences inspirés des formalismes introduits.

(4) Réalisation de prototype mettant en œuvre les solutions proposées dans le cas spécifique d'une construction d'ontologies inspiré de bcXML[5].

Ce mémoire est structuré en trois parties.

La première partie est consacrée à la présentation du cadre théorique de la thèse. Cette partie comprend deux chapitres. Le premier chapitre rappelle les notions de

[5]Building and Construction eXtended Markup Language

base du domaine de la représentation de connaissances et présente la Logique de Description. Cette dernière est le formalisme choisi afin de représenter les connaissances commerciales et de développer les services d'inférence capables de répondre à certains scénarios d'échange de données dans le Commerce Électronique. Le deuxième chapitre définit et formule une variante du problème de transparence sémantique (PTS) rencontrée dans le Commerce Électronique. Le choix de placer cette formulation du problème dans ce chapitre, d'une part, libère de la plongée dans le domaine applicatif les lecteurs qui ne s'intéressent qu'au cadre théorique de la thèse. D'autre part, ce choix permet d'aller directement à la troisième partie de la thèse aux lecteurs qui cherchent seulement des solutions concrètes au problème de transformations d'ontologies. Ce chapitre introduit également les extensions existantes de la Logique de Description qui sont motivées par différentes applications. Ces extensions permettront d'une part de résoudre partiellement le problème formulé, d'autre part de se munir du fondement nécessaire pour les contributions théoriques de cette thèse.

La deuxième partie présente les contributions aux développements algorithmiques pour les deux instances du problème de transparence sémantique. Le premier chapitre de cette partie se concentre sur les deux inférences non-standard connues : le Plus Spécifique Subsumeur Commun (lcs) dans la Logique de Description $\mathcal{ALE}$ et l'approximation d'une description de concept $\mathcal{ALC}$ par une description de concept $\mathcal{ALE}$. Une solution à la première instance du problème formulé est l'application directe de ces inférences. Une amélioration des algorithmes pour ces inférences est le sujet principal traité dans ce chapitre. Le deuxième chapitre de cette partie introduit une nouvelle approche au problème de la révision d'une base de connaissance basée sur la Logique de Description. Le fruit de cette approche est la définition des opérations de révision pour la terminologie (TBox) qui utilise une Logique de Description autorisant le constructeur de restriction existentielle. Nous montrons que le problème du calcul pour les opérations de révision définies est décidable en proposant des algorithmes correspondant à chaque opération. Les règles de contexte présentées au-dessus sont représentées sous forme de *règles de révision* à l'aide des opérations de révision définies. Ensuite, le formalisme hybride combinant la Logique de Description $\mathcal{FLE}$ ou $\mathcal{ELN}^-$ avec les règles de révision est construit à partir des éléments obtenus. Nous exhibons également une condition de la terminaison de la procédure qui procède à l'extension d'une base de connaissances basée sur le formalisme hybride. Cette extension résulte de l'application de règles de révision. Le chapitre se termine par la présentation de la procédure qui détermine l'ordre d'application de règles tel que le nombre des règles appliquées soit maximal et la base de connaissance obtenue soit saturée.

La troisième partie commence par un chapitre qui présente les modèles d'échange de données existants dans le Commerce Électronique. Notamment, une analyse détaillée du modèle d'échange de données ebXML nous permet d'identifier la problématique du modèle. Cette dernière motive les travaux effectués dans cette thèse. Le chapitre suivant de cette partie est consacré à présenter le projet ONDIL qui est considéré comme un prototype de nos travaux.

Première partie

Approche Logique de la Représentation de Connaissances Hétérogènes et Logique de Description

CHAPITRE 1

Représentation de Connaissances et Logique de Description

Dans le domaine d'Intelligence Artificielle, un système nécessite un grand nombre de connaissances afin d'effectuer une inférence "intelligente". C'est pourquoi un tel système doit être muni d'un mécanisme permettant de représenter les connaissances et d'en tirer les conclusions. D'autre part, les normes actuelles pour les échanges de données dans le domaine du commerce électronique (ebXML, bcXML, *etc.*) se basent sur les *taxonomies* représentant les connaissances du domaine d'un acteur. Traditionnellement, un langage de conception, dont la sémantique n'est pas *formelle i.e* non-déclarative, est utilisé pour définir ces taxonomies. En conséquence, l'équivalence et la subsomption *sémantique* entre les termes définis dans différentes taxonomies ne peuvent pas être détectées. De plus, ces difficultés se multiplient lorsque la différence entre les taxonomies qui sont définies par différents acteurs, est déterminée par des *règles*. Ces dernières permettent à un acteur de modifier la définition d'un terme qui est défini par un autre acteur. Dans ce contexte, la Logique de Description montre un formalisme approprié pour cet objectif.

Ce chapitre commence par une présentation des notions de base du domaine de la représentation des connaissances et du raisonnement. Nous y montrons la nécessité d'une sémantique déclarative du langage de représentation. Ensuite, nous décrivons la Logique de Description comme un langage de représentation dérivé de la logique du premier ordre, et donc, muni d'une sémantique déclarative. En particulier, nous discutons des inférences développées résultant de la syntaxe particulière de la Logique de Description par rapport à la logique du premier ordre. Le chapitre se termine par une discussion sur les relations entre la Logique de Description et d'autres langages de représentation.

1.1. Connaissances et représentation de connaissances

Lorsque l'on discute du domaine de la représentation de connaissances, la première question qui se pose porte sur la différence entre les *données* stockées dans la base de données et les *connaissances*. Cette différence peut être caractérisée comme suit : les connaissances comportent l'abstraction et la généralisation de volumineuses données. La caractérisation de cette distinction peut changer en fonction de la manière d'interpréter l'information. Par exemple, une autre caractérisation peut se baser sur ce qui sont stockées dans la base de connaissances et dans la base de données.
En partant de cette idée sur les connaissances, une autre question qui se pose est celle de la représentation de ces connaissances. A ce sujet, *l'hypothèse de la représentation*

de connaissances, qui est largement acceptée par la communauté des chercheurs dans le domaine d'Intelligence Artificielle (I.A), stipule que :

- Si nous connaissons quelque chose, alors il y a un *objet* à notre connaissance *i.e* nous connaissons quelque chose *sur une entité particulière*.
- Cette connaissance peut être symboliquement codée pour que les symboles puissent être manipulés sans se référer aux objets.

Si l'on accepte cette l'hypothèse, cela signifie que l'on essaie de construire un système qui *ressemble fonctionnellement* aux connaissances humaines.

Afin d'avoir une base de connaissances (B.C), un système nécessite un *langage bien spécifié* , dit *un formalisme*, pour coder les connaissances dans la base. Un tel système est capable de fournir des services permettant d'inférer *les connaissances implicites* à partir des *connaissances explicites* stockées dans la base.
Depuis que le domaine de la représentation de la connaissance attire l'attention de la communauté de chercheurs, autour des années 70, les approches de la représentation de la connaissance peuvent être divisées en deux catégories : les formalismes basés sur la logique et les représentations non-logiques. Les représentations non-logiques utilisent des interfaces graphiques permettant de manipuler des connaissances représentées à l'aide de structures de données *ad hoc* (frames, réseaux sémantiques, *etc.*). Par ailleurs, ces représentations, comme par exemple les systèmes de production, dérivent de notions cognitives identifiées à partir d'expériences particulières [**BCM$^+$03 (8)**]. En général, les représentations non-logiques souffrent d'un manque de *caractérisation sémantique* précise. Une conséquence de cette lacune est que nous ne savons pas si les conclusions obtenues du système sont correctes et complètes.
Une des directions de recherche pour combler cette lacune des représentations non-logiques est de munir le formalisme de la représentation de connaissances d'une *sémantique formelle*. La logique du premier ordre est le premier candidat pour cette approche grâce à son expressivité.

1.2. Approche logique de la représentation de connaissances

Dans l'approche logique, le formalisme de la base de connaissances est non seulement la logique prédicate classique du premier ordre mais aussi les autres logiques, dites Logiques Non-Classiques : Logique Modale et Logique Nonmonotone. Les critiques de l'approche logique de la représentation de connaissances résultent souvent de l'égalisation de la logique et de la logique prédicate classique du premier ordre, car cette dernière ne permet pas de représenter des connaissances incomplètes, subjectives et temporelles. Dans le cadre de ce mémoire, nous n'étudions qu'une variante particulière de la logique du premier ordre. Il s'agit de la Logique de Description qui est le sujet de la Section 1.3.

1.2.1. La logique du premier ordre classique et sa sémantique. Dans ce paragraphe, nous abordons brièvement la logique du premier ordre. Une description suffisante de cette logique pour la représentation de connaissances peut être trouvée dans [**TGL$^+$88 (66)**].
En général, la logique du premier ordre est un langage formel comportant la *syntaxe* qui permet de distinguer les phrases logiques parmi les assemblages de sous-phrases, et la *sémantique* qui attribue une signification aux phrases logiques. Cette sémantique est souvent appelée *sémantique de Tarski* qui est un prototype d'une *sémantique déclarative*. En effet, chaque formule du calcul des prédicats se compose des expressions de base suivantes :

- Les *constantes* qui désignent des *noms spécifiques* d'objets.
- Les *variables* qui désignent des *noms génériques*
- Les *noms de prédicat* qui désignent les *règles d'assemblage* entre constantes et variables
- Les *noms de fonction* qui représentent les mêmes règles que les prédicats.

Il faut attribuer une valeur sémantique unique à chaque expression de base. Ceci permet d'enlever les ambiguïtés lexicales du monde réel. C'est pourquoi on a besoin d'une fonction qui applique les noms des objets du langage sur les entités du monde réel. Ces entitées sont elles-mêmes exprimées dans un langage appelé le *métalangage*. Dans ce cas, le métalangage est le français. Le concept fondamental de la sémantique est celui de *vrai dans le monde réel* ou *vrai dans un modèle*.
La méthode permettant de déterminer les valeurs sémantiques des formules logiques s'appuie sur l'interprétation d'une formule logique. Pour ce faire, chaque formule logique reçoit une *valeur de vérité*. Cependant, les composants d'une formule ne sont pas seulement des sous-formules mais aussi des *termes*. Un terme désigne intuitivement un *objet*. Une interprétation devra donc spécifier un ensemble d'objets non-vide, appelé *domaine d'interprétation*. Cette interprétation permet d'associer à tout composant d'une formule un objet dans le domaine d'interprétation ou dans l'ensemble $\{Vrai, Faux\}$.
On dit qu'une formule du calcul des prédicats A est *vraie pour une interprétation* $\mathcal{I}$ lorsque l'on a l'interprétation $\mathcal{I}$ qui associe à la formule A la valeur $Vrai$. On dit qu'une formule est *consistante* si elle est vraie pour au moins une interprétation, sinon elle est dite *inconsistante*. Contrairement à ce qui se passe pour le calcul des propositions, il n'existe pas d'algorithme général déterminant à quelle classe appartient une formule du calcul des prédicats. On peut démontrer l'indécidabilité de ce problème par réduire le *Problème de Correspondance de Poste*, qui est indécidable à celui-ci [**HU79 (42)**].

1.2.2. Sémantiques procédurale et déclarative. D'abord nous présentons des caractéristiques des deux points de vue opposés. Les *procéduralistes* affirment que le processeur des informations humaines est un appareil de programmes stockés et les connaissances du monde sont encastrées dans ces programmes. Par contre, les *déclarativistes* ne sont pas convaincus que les connaissances d'un domaine soient intimement liées aux procédures qui utilisent ces connaissances. Ils pensent que "l'intelligence" se

situe dans les deux bases suivantes : un ensemble général des procédures permettant de manipuler de toute sorte des faits et un ensemble des faits spécifiques décrivant le domaine en question. En d'autres termes, des procédures générales sont appliquées aux données spécifiées du domaine pour effectuer des déductions.
Quel point de vue est le plus avantageux ? A notre avis, il n'est pas très utile de chercher une réponse à cette question. Plutôt, nous examinons les mécanismes qui sont développés pour traiter ces représentations et indiquons les avantages que chaque approche peut offrir.

– Avantages de la sémantique déclarative

  (1) *Flexibilité et Économie.* Lorsqu'un fragment de connaissance est spécifié, il ne semble pas convaincant de spécifier impérativement chaque usage par avance. Considérons le fait suivant : "Tous mes amis côte-azuréens savent faire du vélo". Il peut servir à répondre à la question "Est-ce que Paul sait faire du vélo ?" en vérifiant que Paul est bien un ami côte-azuréen. Ce fait peut servir également à décider que John ne vient pas de Côte d'Azûr parce qu'il ne sait pas faire du vélo. Dans les représentations strictement procédurales *i.e* lorsque la sémantique du formalisme de ces représentation est procédurale, ce fait devrait être différemment représenté pour chaque déduction : répondre à une question ou à un nouveau fragment d'information ajouté. Toutefois, la logique du premier ordre peut fournir une représentation déclarative simple sous la forme d'une formule du calcul des prédicats :

  $$\forall(x)(\ \text{Azuréen}(x) \wedge \text{MesAmis}(x) \Rightarrow \text{FaireDuVélo}(x)\ )$$

  Les utilisations différentes de ce fait résulte d'un mécanisme de déduction général qui peut accéder à cette représentation. Si cette formule est ajoutée au système, nous ne devons pas anticiper comment la formule sera utilisée. Par conséquent, le programme est plus flexible lorsqu'il effectue les déductions.

  (2) *Compréhensibilité.* La simplicité de la représentation déclarative ci-dessus permet non seulement d'économiser de la mémoire mais aussi de faciliter la compréhension et la modification des connaissances du système. Si la base de connaissances comporte les faits indépendants, cette base peut être modifiée en ajoutant un nouveau fait et les implications de chaque fait se situent au contenu logique du fait. Ce qui n'est pas le cas pour les programmes.

– Avantages de la sémantique procédurale

  (1) *Modélisation procédurale.* Dans certains systèmes, par exemple la description des manipulations d'un robot, il est difficile de trouver une représentation déclarative pure. En revanche, il est très naturel de décrire les manipulations d'un robot par un programme.

(2) *Connaissances du second ordre.* Un composant important de nos connaissances est de connaître ce que l'on connaissait et ce que l'on peut connaître, appelées *méta-connaissances*. On a explicitement les faits qui disent comment utiliser d'autres faits pour raisonner. Cela doit être exprimé dans notre représentation. Dans la majorité des cas, on peut obtenir *directement* des méta-connaissances par la technique heuristique. Par exemple, "si vous ne voyez pas d'une raison *évidente* pour laquelle la route est impraticable, vous pouvez conduire". Le problème est évidemment caché dans le mot "évidente". Ce dernier est très loin des notions logiques : la vérité ou démontrabilité. Théoriquement, il est possible d'exprimer les connaissances du second ordre par une représentation déclarative, mais cela devient extrêmement difficile si la notion de contexte est prise en compte. Par contre, dans la représentation procédurale, on peut parler directement de particulières manières d'accéder aux faits.

1.2.3. Programmation logique. Brièvement, le principe général de la programmation logique est de représenter des propriétés algorithmiques d'une fonction sous forme d'un ensemble de clauses de Horn, et d'utiliser la *résolution* pour calculer des valeurs de cette fonction. La résolution permet de vérifier si un ensemble de clauses est inconsistant en engendrant des conséquences logiques de cet ensemble jusqu'à l'obtention de la clause vide (toujours faux). Par nature séquentielle, des algorithmes qui sont implémentés dans certains langages, par exemple Prolog, sont exécutés dans un ordre bien déterminé pour atteindre une réponse.

Depuis peu, la programmation logique est considérée comme un formalisme universel pour la représentation de connaissances. Toutefois, comme indiqué par son nom, les langages de la programmation logique sont des langages de *programmation*, ils ne sont donc pas appropriés lorsqu'ils sont utilisés comme langage de *représentation*. En effet, la majorité des langages de programmation logique, Prolog inclus, n'est pas muni d'une sémantique déclarative appropriée. Sous l'angle de la sémantique déclarative, l'ordre des clauses et des conjoncts dans un programme n'est pas pertinent. Cela n'implique pas qu'il soit impossible de résoudre un problème de représentation spécifique à l'aide du langage Prolog. Cependant, d'après la discussion ci-dessus, l'absence de la sémantique déclarative empêche de coder les connaissances indépendemment de la manière de traiter ces connaissances. Par conséquent, toutes les responsabilités des comportements du programme, par exemple la terminaison des processus d'inférence, appartiennent aux programmeurs et ne sont pas automatiquement assurées.

D'autre part, si l'on considère les langages de la programmation logique comme langages de représentation, il y a certains constructeurs facilement exprimés par des formalismes munis d'une sémantique déclarative, par exemple la Logique de Description, ne peuvent pas être exprimés par ces langages. Effectivement, la disjonction et la restriction existentielle qui permettent de représenter des connaissances spécifiées incomplètement ne sont pas exprimables par les langages de la programmation

logique courants. Il existe certes certaines extensions des langages de la programmation logique courants pour combler ces lacunes par ajout de la disjonction et de la négation classique, mais les solutions proposées ne sont pas complètes.

1.3. Logique de Description

Comme nous avons vu dans la Section 1.2, la logique prédicate du premier ordre (L.P.P.O) est en général indécidable. De plus, la syntaxe courante de cette logique ne supporte pas une représentation des connaissances structurée. Cette représentation doit permettre de regrouper *syntaxiquement* les informations qui sont *sémantiquement* liées. Motivé par ces exigences, on a cherché d'une part à munir les langages de représentation structurelle existants d'une sémantique déclarative comme celle de la L.P.P.O, d'autre part à identifier des fragments décidables de la L.P.P.O en réduisant l'expressivité de la L.P.P.O. Les Logiques de Description (L.D) - chaque logique diffère des autres par l'ensemble de constructeurs utilisés - résultent de ces directions de recherche.

1.3.1. Historique. Les formalismes *Frames* et *Réseaux Sémantiques* sont considérés comme les précurseurs de la L.D. Le formalisme *frames* est introduit par M. Minsky [**Min75 (55)**]. Dans ce formalisme, la structure de données *enregistrement* représente une situation et un objet. L'idée est de collecter toutes les informations nécessaires concernant une situation et de les mettre dans une place, appelée *frame*. Certains auteurs, par exemple Hayes [**Hay79 (41)**], ont critiqué l'absence d'une sémantique formelle dans ce formalisme et montré qu'une fois convenablement formalisé, ce formalisme est une variante syntaxique de la L.P.P.O.

Quant au formalisme *réseaux sémantique*, il est développé par M. Quillian pour représenter la sémantique du langage naturel [**Qui68 (62)**]. Ce formalisme a une représentation de graphe dont les nœuds représentent des concepts et des individus. Ces nœuds sont connectés par des arcs étiquetés. Il y a deux sortes d'arcs : les arcs de propriété qui affectent les propriétés à des concept ou à des individus et les arcs IS-A qui introduisent les relations hiérarchiques entre des concept ou entre des individus. Comme exemple, nous considérons un réseau sémantique très simple dans la Figure 1.3.1.

Une des interprétations possibles de ce réseau est que Produit_1 est une instance du concept ProdResAuBrt (ProduitRésisterAuBruit) qui est un sous-concept de Produit. Toute instance du concept ProdResAuBrt résiste au bruit. Les propriétés sont héritées via les arcs IS-A. Par exemple, Produit_1 doit résister au bruit. Puisque le formalisme des réseaux sémantiques n'est pas muni d'une sémantique formelle, la signification précise d'un réseau sémantique est décidée par les utilisateurs et des programmeurs. Dans l'exemple ci-dessus, le réseau n'indique pas clairement si une instance du concept ProdResAuBrt peut résister au feu, ou bien si les instances du concept ProdResAuBrt ne résistent qu'au bruit. Par conséquent, différents systèmes de représentation des connaissances basés sur le même réseau sémantique pourraient avoir interprétation différente.

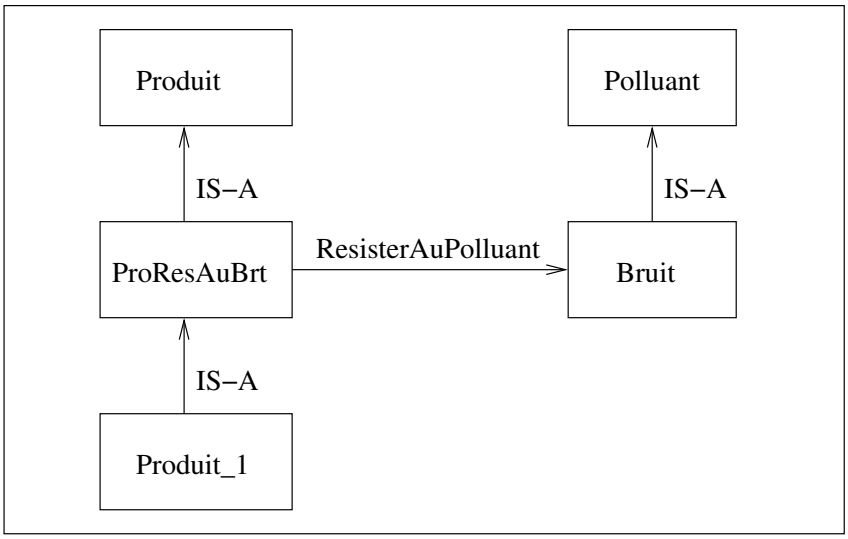

FIG. 1.3.1. Un réseau sémantique

Brachman dans [**Bra77 (13), Bra78 (14)**] a critiqué cet inconvénient des réseaux sémantiques et il a proposé une nouvelle représentation, appelée "réseaux d'héritage structurels", qui est équipée d'une sémantique formelle. Le nouveau formalisme avec cette sémantique formelle permet de capturer précisément la signification du formalisme qui est exprimée indépendemment des programmes attachés. Ce formalisme est implanté pour la première fois dans le système KL-ONE et est considéré comme la première L.D.

Les premiers systèmes qui sont les successeurs de KL-ONE utilisent des *algorithmes de subsomption structurelle* pour décider la subsomption entre des descriptions de concept. L'idée principale de ces algorithmes peut être décrite comme suit. Les descriptions de concept sont transformées en une forme normale dans laquelle les parties "comparables" entre les descriptions de concept de chaque description de concept sont regroupées. Alors, la subsomption entre les description de concept est récursivement vérifiée en comparant les formes normales correspondantes. De tels algorithmes de subsomption structurels pour les L.D expressives sont polynômiaux mais incomplets *i.e* il y a des relations de subsomptions qui ne sont pas détectées. Le problème réside dans la difficulté de définir une forme normale appropriée pour ces L.D expressives.

Les premiers travaux sur la puissance d'expression des L.D et la complexité de calcul ont montré que les L.D avec l'expressivité assez modeste, dont le problème de subsomption est déjà NP-difficile. Une issue de cette difficulté est de construire les systèmes moins expressifs mais avec les algorithmes corrects et complets *i.e* la correction et la complétude des systèmes sont assurées en sacrifiant l'expressivité.

Le travail de Schmidt-Schauß et Smolka [**SS91 (63)**] a ouvert une nouvelle direction de recherche sur les algorithmes de décision pour les L.D expressives. Effectivement, dans ce travail, les auteurs ont développé un algorithme correct et complet

basé sur la technique de *tableaux* pour le langage $\mathcal{ALC}$. Ce langage, qui est une logique de description, comporte la négation complète. Plus tard, cette technique a été étendue à d'autres L.D expressives. De plus, les systèmes L.D en s'appuyant sur les algorithmes de tableaux, par exemple FaCT, montrent que ces algorithmes sont acceptables en pratique malgré les résultats décourageants de la complexité.

Nous présentons maintenant les logiques de description étudiées dans ce mémoire. Il s'agit des logiques qui contiennent un sous-ensemble de l'ensemble des constructeurs suivants : conjonction ($\sqcap$), restriction universelle ($\forall r.C$), restriction existentielle ($\exists r.C$), disjonction ($\sqcup$), restrictions de cardinalité supérieure et inférieure ($\leq nr$, $\geq nr$) et négation primitive (ou complète).

C, D	$\rightarrow$	$\top \mid$	(concept universel ou top)
		$\bot \mid$	(concept vide ou bottom)
		$P \mid$	(nom de concept)
		$\neg P \mid$	(négation primitive)
		$\geq n\ r \mid$	(restriction de cardinalité inférieure)
		$\leq n\ r \mid$	(restriction de cardinalité supérieure)
		$C \sqcap D \mid$	(conjonction de concept)
		$C \sqcup D \mid$	(disjonction de concept)
		$\forall r.C \mid$	(restriction universelle)
		$\exists r.C \mid$	(restriction existentielle)
		$\neg C \mid$	(négation complète)

FIG. 1.3.2. Syntaxe des Descriptions de Concept

1.3.2. Syntaxe et sémantique. Traditionnellement, un système L.D comporte deux composants. Le premier est la base de connaissances (B.C) qui est encore divisée en deux blocs appelés *TBox* et *ABox*. Le deuxième est le moteur d'inférence qui implante les services d'inférence. Un TBox stocke les connaissances *conceptuelles* (terminologiques) du domaine d'application tandis qu'un ABox présente les connaissances *assertionnelle*s. Les concepts dans la B.C sont représentés par les *descriptions de concept*.

Description de concept. Les descriptions de concept d'un langage L.D sont définies récursivement à partir d'un ensemble des *noms de concept* N_C, d'un ensemble des *noms de rôle* N_R (rôle d'identité ϵ inclus) et l'ensemble des constructeurs que ce langage possède. Dans ce mémoire, nous désignons, s'il n'y a pas d'indication particulière, les éléments de N_C par les lettres A, B ; les éléments de N_R par les lettres r, s ; et les descriptions de concept par les lettres C, D.

A partir de ces notions, les descriptions de concept qui sont considérées dans ce mémoire se composent selon les règles de syntaxe décrites dans la Figure 1.3.2. La Figure 1.3.3 montre les constructeurs correspondant aux langages étudiés dans ce mémoire.

Constructeurs de concept	$\mathcal{EL}$	$\mathcal{ELN}$	$\mathcal{FLE}$	$\mathcal{ALE}$	$\mathcal{ALC}$
$\top$	×	×	×	×	×
$\bot$				×	×
$\neg A$				×	×
$(C \sqcap D\ )$	×	×	×	×	×
$(C \sqcup D\ )$					×
$\geq n\ r$		×			
$\leq n\ r$		×			
$\forall r.C$			×	×	×
$\exists r.C$	×	×	×	×	×

FIG. 1.3.3. Langages de DL

Afin de définir une sémantique formelle des descriptions de concept, nous considérons une *interprétation* $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ où $\Delta^{\mathcal{I}}$ est un ensemble non vide, appelé *domaine d'interprétation*, $.^{\mathcal{I}}$ est une *fonction d'interprétation*. Cette fonction associe chaque concept $A \in N_C$ à un ensemble $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ et associe chaque rôle $r \in N_R$ à un ensemble $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. La fonction d'interprétation est étendue aux descriptions de concept comme suit :

Syntaxe	Sémantique
$\top$	$\Delta^{\mathcal{I}}$
$\bot$	$\emptyset$
$C \sqcap D$	$\mathrm{C}^{\mathcal{I}} \cap D^{\mathcal{I}}$
$C \sqcup D$	$\mathrm{C}^{\mathcal{I}} \cup D^{\mathcal{I}}$
$\forall r.C,\ r \in N_R$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
$\exists r.C,\ r \in N_R$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
$\leq n\ r$	$\{x \in \Delta^{\mathcal{I}} \mid \mathrm{card}(\{x \mid (x,y) \in r^{\mathcal{I}}\}) \leq n\ \}$
$\geq n\ r$	$\{x \in \Delta^{\mathcal{I}} \mid \mathrm{card}(\{x \mid (x,y) \in r^{\mathcal{I}}\}) \geq n\ \}$

En utilisant les constructeurs présentés, nous pouvons décrire précisément la signification attendue du concept ProdResAuBrt dans la Figure 1.3.1 : "c'est un produit qui peut résister au moins au bruit" par la description de concept suivante :

$$\mathsf{ProdResAuBrt} \equiv \mathsf{Produit} \sqcap \exists \mathit{resister}.\mathsf{Bruit}$$

Terminologie : TBox. Les description de concept sont utilisés dans un TBox pour définir les concepts du domaine d'application. Nous pouvons introduire à un TBox les noms de concept correspondant aux descriptions de concept c'est-à-dire que les descriptions de concept sont baptisées dans un TBox. Étant donné un langage de logique de description L, nous avons besoin d'une définition exacte du TBox dans laquelle les notions abordées sont précisées.

Définition 1.3.1. Un TBox $\Delta_{\mathcal{T}}$ est un ensemble fini de *définitions de concept* sous forme $A := C$ où $A \in N_C$ et C est une L-description de concept. Un nom de concept $A \in N_C$ est appelé *nom défini* s'il apparaît du côté gauche d'une définition de concept, sinon il est appelé *nom primitif*. De plus, un nom défini doit apparaître une seule fois du côté gauche dans toutes les définitions de concept du $\Delta_{\mathcal{T}}$. La définition de concept C dans la définition $A := C$ est appelé *concept de définition* de A.

Bruit :=	Polluant $\sqcap \neg$Feu
ProdResAuPolluant :=	Produit $\sqcap \exists$*résister*.Polluant
ProdResAuBrt :=	Produit $\sqcap \exists$*résister*.Bruit
ProdResAuFeu :=	Produit $\sqcap \exists$*résister*.Feu
ProdPourBureau :=	ProdResAuBrt $\sqcap$ ProdResAuFeu

FIG. 1.3.4. Un exemple de TBox

Notons qu'une définition de concept $A := C$ est interprétée comme l'équivalence logique *i.e* cette définition fournit la *condition nécessaire et suffisante* pour classifier un individu dans le concept A. La puissance de cette sorte de déclaration est une caractéristique des systèmes basés sur la logique de description.

Dans la littérature, des axiomes d'inclusion sous forme $C \sqsubseteq D$, où C, D sont des descriptions de concept, sont également introduites à un TBox . Notons qu'une définition de concept $A := C$ est équivalente aux deux axiomes d'inclusion : $A \sqsubseteq C$ et $C \sqsubseteq A$. Toutefois, de tels TBox ne sont pas étudiés dans ce mémoire. La Figure 1.3.4 présente un exemple de TBox.

Un TBox $\Delta_{\mathcal{T}}$ est appelé *acyclique* s'il n'existe pas un nom de concept qui est *directement* ou *indirectement* défini via lui-même. Un TBox est dit *déplié* s'il n'existe pas de concept de définition qui contienne un nom défini. On peut obtenir le TBox déplié d'un TBox acyclique en remplaçant systématiquement les noms définis par leur concept de définition. Selon le travail dans [**Neb90b (59)**], la taille du TBox déplié obtenu peut être exponentielle en la taille de $\Delta_{\mathcal{T}}$.

Une interprétation $\mathcal{I}$ est un modèle d'un TBox $\Delta_{\mathcal{T}}$ si $A^{\mathcal{I}} = C^{\mathcal{I}}$ pour toute définition de concept $A := C$ dans $\Delta_{\mathcal{T}}$. La sémantique d'un TBox est défini par l'ensemble de ses modèles. Deux TBox sont équivalents s'ils ont les mêmes modèles.

Description de monde : ABox. Le deuxième composant de la B.C est le ABox. Contrairement au TBox qui restreint l'ensemble des mondes possibles, le ABox permet de décrire un état spécifique du monde en introduisant les individus et assertions de propriétés sur ces individus. Nous désignons les individus par les noms a, b, c. Les assertions sont représentées par les deux formes suivantes : $C(a)$, $r(b, c)$ où $C \in N_C$, $r \in N_R$. Un ABox $\Delta_{\mathcal{A}}$ est un ensemble fini de telles assertions. La figure 1.3.5 montre un exemple de ABox.

Afin de munir le ABox d'une sémantique, nous étendons les interprétations définies aux noms des individus. C'est-à-dire qu' une interprétation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ non

ProdResAuPolluant(FORMICA_1)	ProdResAuBrt(FORMICA_1)
Feu(INF_1000)	*résister*(FORMICA_1, INF_1000)

FIG. 1.3.5. Un exemple de Abox

seulement associe les noms de concepts et noms de rôles aux ensembles et aux relations, mais aussi associe un nom d'un individu a à un élément $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. Ces interprétations doivent satisfaire *l'hypothèse de nom unique* qui dit que si deux noms d'individus a et b sont différents alors $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. L'interprétation $\mathcal{I}$ satisfait l'assertion $C(a)$ et $r(a,b)$ si $a^{\mathcal{I}} \in C^{\mathcal{I}}$ et $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$. L'interprétation $\mathcal{I}$ satisfait le ABox $\Delta_{\mathcal{A}}$ si $\mathcal{I}$ satisfait toute assertion dans $\Delta_{\mathcal{A}}$, dans ce cas, $\mathcal{I}$ est un *modèle* de $\Delta_{\mathcal{A}}$. L'interprétation $\mathcal{I}$ est un modèle d'un ABox $\Delta_{\mathcal{A}}$ par rapport à un TBox $\Delta_{\mathcal{T}}$ si $\mathcal{I}$ est un modèle de $\Delta_{\mathcal{A}}$ et de $\Delta_{\mathcal{T}}$.

A partir du point de vue ci-dessus, une instance d'une base de données relationnelle qui a seulement les relations unaires et binaires, est un ABox. Toutefois, la sémantique de la base de données implique que les connaissances (données) dans la base sont complètes *i.e* si l'assertion $C(a)$ n'existe pas dans la base, on déduit que $C(a)$ est fausse. Une telle sémantique est appelée *sémantique d'un monde fermé*. De plus, le TBox qui impose les relations sémantiques entre les concepts et rôles dans le ABox, est comparable aux contraintes d'intégrité dans une base de données.

1.3.3. Services d'inférences standards. L'objectif d'un système de la représentation des connaissances est non seulement de stocker les définitions de concepts et les assertions mais aussi de fournir des services d'inférence qui permettent d'obtenir les connaissances qui ne sont pas représentées explicitement dans la base. Par exemple, l'assertion ProdPourBureau(FORMICA_1) peut être tirée à partir du TBox dans la Figure 1.3.4 et du ABox dans la Figure 1.3.5.

(1) *Subsomption.* Nous commençons par présenter une inférence fondamentale sur les concepts d'un TBox. Soient C, D des description de concept. D *subsume* C, écrit $C \sqsubseteq D$, par rapport à un TBox $\Delta_{\mathcal{T}}$ ssi $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ du TBox $\Delta_{\mathcal{T}}$. C, D sont *équivalentes*, écrit $C \equiv D$, par rapport à un TBox $\Delta_{\mathcal{T}}$ ssi $C \sqsubseteq D$ et $D \sqsubseteq C$. L'inférence de subsomption se sert de calculer la hiérarchie des concepts définis dans un TBox. Cette inférence nous permet de vérifier les relations entre les concepts. Si un concept se situe à une place inattendue dans la hiérarchie, cela pourrait montrer une erreur de conception. La Figure 1.3.6 présente la hiérarchie correspondante du TBox dans la Figure 1.3.4.

(2) *Satisfiabilité.* Soit C une description de concept. C est satisfiable (ou consistant) par rapport à un TBox $\Delta_{\mathcal{T}}$ ssi il existe un modèle $\mathcal{I}$ du TBox $\Delta_{\mathcal{T}}$ tel que $C^{\mathcal{I}} \neq \emptyset$.

Il y a une relation étroite entre les inférences de subsomption et de satisfiabilité. Effectivement, le problème de satisfiabilité peut être réduit au problème de subsomption car une description de concept C est *insatisfiable* ssi $C \sqsubseteq \bot$.

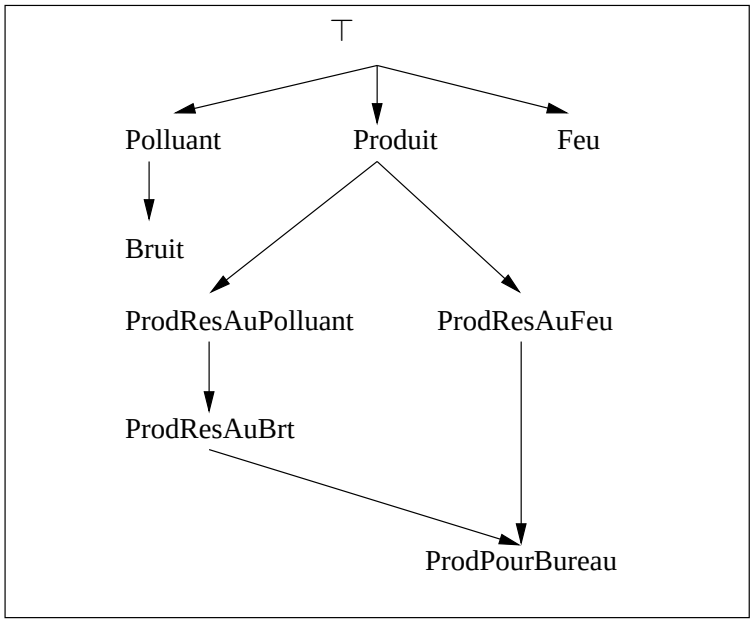

FIG. 1.3.6. Hiérarchie de subsomption

Inversement, pour les langages L.D qui comportent le constructeur de négation complète, par exemple $\mathcal{ALC}$, la subsomption peut être réduite à la satisfiabilité car $C \sqsubseteq D$ est équivalente à $C \sqcap \neg D \equiv \bot$.

(3) *Vérification d'instance*. Maintenant, nous introduisons une inférence importante sur le ABox. Soient $\Delta_{\mathcal{T}}$ un TBox et C une description de concept. Soient $\Delta_{\mathcal{A}}$ un ABox et a un individu dans cet ABox $\Delta_{\mathcal{A}}$. L'individu a est une *instance* de C par rapport au TBox $\Delta_{\mathcal{T}}$et au ABox $\Delta_{\mathcal{A}}$ ssi $a^{\mathcal{I}} \in C^{\mathcal{I}}$ pour tous les modèles $\mathcal{I}$ de $\Delta_{\mathcal{T}}$ et de $\Delta_{\mathcal{A}}$.

Le travail dans [**SS91 (63)**] a montré que le problème de subsomption pour le langage $\mathcal{ALC}$ est PSPACE-complet. Toutefois, le même problème pour le langage $\mathcal{ALE}$ est NP-complet. Ce résultat a été démontré dans [**DHL$^+$92 (22)**].

Par ailleurs, il y a une autre famille des inférences, appelée *inférence non-standard*, qui est également étudiée. Même si les inférences non-standards sont intéressées bien avant, les algorithmes corrects et complets pour ces inférences sont développés récemment [**BKM99 (5)**]. Certaines inférences non-standards parmi elles seront étudiées en détail dans le Chapitre 3.

1.3.4. Deux approches des algorithmes de décision. A partir de la technique utilisée dans le développement des algorithmes de décision pour les inférences standards nous pouvons les diviser en deux groupes. Le premier est appelé algorithmes de *subsomption structurelle* et le deuxième est appelé algorithmes de *tableaux*.

Approche structurelle. Cette approche résulte du point de vue des réseaux sémantiques dans lequel la description de concept est considérée comme un graphe orienté comportant les nœuds et les arcs. L'exécution des algorithmes de subsomption structurelle peut être divisée en deux étapes. Premièrement, les graphes correspondant aux descriptions de concept en question sont normalisés. La deuxième étape procède à une comparaison entre les graphes normalisés pour détecter les similitudes. Les algorithmes de subsomption structurelle sont souvent très efficaces (polynômiaux) mais incomplets. C'est-à-dire que la réponse résultant de ces algorithmes n'est que correcte si elle est positive. Par conséquent, le comportement des algorithmes dépend non seulement de la sémantique mais aussi d'autres facteurs que l'utilisateur doit prendre en compte. Toutefois, cette approche montre un avantage important lorsqu'elle est utilisée pour des *algorithmes de construction*. Ces derniers doivent calculer et retourner une description de concept d'un ensemble infini des descriptions de concept, contrairement aux algorithmes de décision. Actuellement, les inférences non-standard ne peuvent que se baser sur l'approche structurelle. Le détail de l'approche structurelle sera montré dans la Section 2.2.2 du Chapitre 2.

Approche des tableaux. Comme nous avons montré dans les sous-sections précédentes, le point de vue logique est utilisé pour définir la sémantique de la Logique de Description. Encore plus loin, ce point de vue permet également de développer les algorithmes qui peuvent être interprétés comme des déductions logiques. En général, ces algorithmes s'appuient sur la méthode spécialisée du calcul de tableaux pour la L.P.P.O, qui génère des modèles finis. Un grand avantage de cette approche est de fournir une technique de base sur laquelle les algorithmes de décision (satisfiabilité, subsomption) corrects et complets sont développés pour des langages L.D très expressifs. Ces langages peuvent comporter les constructeurs suivants : disjonction, négation complète, fermeture transitive de rôles, *etc.* On a obtenu les résultats les plus importants sur la complexité des inférences standards grâce à cette approche. La technique de base de l'approche des tableaux sera décrite dans la Section 2.2.3 du Chapitre 2.

1.4. Relations entre la logique de description et d'autres formalismes

1.4.1. Graphe conceptuel. Le formalisme de graphes conceptuels est introduit par Sowa [**Sow84 (64)**]. C'est un formalisme expressif qui permet de représenter graphiquement les connaissances d'un domaine d'application. Les graphes conceptuels sont munis d'un formalisme formel en les transformant en des formules de la logique du premier ordre. Certains services d'inférences pour ce formalisme ont été développés, comme par exemple la validation d'un graphe et la subsomption entre deux graphes. Puisque les graphes conceptuels peuvent exprimer toute la logique des prédicats du premier ordre, ces problèmes d'inférence sont indécidables pour les graphes conceptuels généraux. Cependant, on a identifié des fragments décidables de ce formalisme. Un fragment décidable important parmi eux est appelé *graphe conceptuel simple*.

Le travail dans [**BMT99b (7)**] a fait une comparaison entre le formalisme de graphes conceptuels et les logiques de description en les transformant en la logique du premier ordre. Il y a plusieurs différences entre les deux formalismes même s'ils peuvent être utilisés dans des applications similaires. Ces différences résultent des éléments suivants :

(1) Les graphes conceptuels sont transformés en des formules du premier ordre avec plusieurs variables libres, alors que les concepts et rôles de la L.D sont transformés en des formules avec une et deux variables libres.

(2) Puisque les L.D utilisent une syntaxe de variable libre, certaines identifications de variables exprimées par des cycles dans les graphes conceptuels ne sont pas exprimables par les L.D.

(3) Les L.D ne permettent que des relations unaires et binaires, alors que l'arité des relations dans les graphes conceptuels peut être supérieure à deux.

(4) La majorité des L.D utilisent la quantification universelle alors que les graphes conceptuels sont interprétés par les phrases qui contiennent la quantification existentielle.

Malgré les différences, les auteurs de [**BMT99b (7)**] ont identifié une L.D correspondante désignée par $\mathcal{ELIRO}_1$ qui permet la conjonction des concepts et la restriction existentielle $\mathcal{EL}$, le rôle inverse $\mathcal{I}$, la conjonction des rôles $\mathcal{R}$ et concept unaire one-of $\mathcal{O}_1$. Avec la L.D $\mathcal{ELIRO}_1$, les auteurs ont montré que si une description de concept est restreinte à contenir au plus un concept one-of dans chaque conjonction, l'arbre syntaxique de la description de concept peut être transformé en un graphe conceptuel simple équivalent qui est un arbre. Inversement, chaque graphe conceptuel simple qui est un arbre et ne contient que les relations binaires, peuvent être transformés en une description de concept équivalente.

La correspondance obtenue entre les graphes conceptuels simples sous forme d'un arbre et les $\mathcal{ELIRO}_1$-descriptions de concept permet de transférer les résultats polynômiaux du problème de subsomption entre ces graphes à la logique de description $\mathcal{ELIRO}_1$. Inversement, la correspondance permet également de déterminer un fragment plus expressif des graphes conceptuels correspondant à une Logique de Description expressive. Grâce à cela, on peut utiliser les algorithmes connus de cette Logique de Description pour décider la validité et la subsomption des graphes dans ce fragment.

1.4.2. Modèle de données orienté-objet. Dans cette sous-section, nous identifions une correspondance entre le modèle de données orienté-objet et la logique de description. C'est la raison pour laquelle nous limitons la discussion au composant structurel (statique) du modèle de données orienté-objet. Le modèle de données orienté-objet se base sur la notion d'*identification d'objet* au niveau extentionnel et sur la notion de *classe* au niveau intentionnel. La structure d'une classe est spécifiée par les mécanismes de *typage* et d'*héritage*.

Afin de présenter une comparaison appropriée, d'abord une formalisation du modèle de données orienté-objet est nécessaire. La formalisation du modèle utilisée ci-dessous est proposée par Abiteboul et Kanellakis [1989].

Un schéma orienté-objet est un ensemble fini de déclarations de classes qui impose les contraintes sur les instances des classes. Une déclaration d'une classe est sous forme :

$$\text{CLASS } C \text{ IS-A } C_1,...,C_k \text{ TYPE-IS } T,$$

où "IS-A" spécifie les inclusions entre l'ensemble des instances de la classe C et les ensembles des instances des classes $C_1,...,C_k$; "TYPE-IS" spécifie par le *type* T la structure des instances de la classe.

Le type T se compose selon la règle suivante :

$$\begin{array}{rcl} T & \rightarrow & C \mid \\ & & \text{UNION } T_1,...,T_k \text{ END} \mid \\ & & \text{SET-OF } T \mid \\ & & \text{RECORD } A_1 : T_1,...,A_k : T_k \text{ END.} \end{array}$$

Le schéma orienté-objet est muni d'une sémantique en spécifiant les caractéristiques d'un *état de base de données* instancié de ce schéma. La définition d'un état de base de données se base sur la notion d'*identifiant d'objet* et la notion de *valeur*. Soit $\mathcal{O}^{\mathcal{J}}$ un ensemble d'identifiants d'objet non vide. A partir de l'ensemble $\mathcal{O}^{\mathcal{J}}$, les valeurs complexes sont récursivement définies en regroupant des valeurs pour créer les ensembles et les enregistrements. Un état de base de données $\mathcal{J}$ d'un schéma est constitué par l'ensemble d'identifiants d'objet, une fonction $\pi^{\mathcal{J}}$ qui associe un sous-ensemble de $\mathcal{O}^{\mathcal{J}}$ à chaque classe et une fonction $\rho^{\mathcal{J}}$ qui associe une valeur à chaque élément de $\mathcal{O}^{\mathcal{J}}$.

Soit $\mathcal{V}_{\mathcal{J}}$ un ensemble fini de valeurs associées aux éléments de $\mathcal{O}^{\mathcal{J}}$ par la fonction $\rho^{\mathcal{J}}$. Une interprétation de *types* (ou expressions de type) dans un état de base de données $\mathcal{J}$ est défini via une *fonction d'interprétation* $.^{\mathcal{J}}$ qui associe un ensemble $T^{\mathcal{J}}$ de valeurs dans $\mathcal{V}_{\mathcal{J}}$ à chaque type T comme suit :

- Si T est une classe C, alors $T^{\mathcal{J}} = \pi^{\mathcal{J}}(C)$;
- Si T est un type d'union UNION $T_1,...,T_k$ END, alors $T^{\mathcal{J}} = T_1^{\mathcal{J}} \cup ... \cup T_k^{\mathcal{J}}$;
- Si T est un type d'enregistrement (d'ensemble), alors $T^{\mathcal{J}}$ est un ensemble des valeurs de 'enregistrement (d'ensemble) qui sont compatibles avec la structure de T.

Un état de base de données $\mathcal{J}$ d'un schéma orienté-objet $\mathcal{S}$ est *légal* par rapport à $\mathcal{S}$ si pour chaque déclaration :

$$\text{CLASS } C \text{ IS-A } C_1,...,C_k \text{ TYPE-IS } T,$$

dans $\mathcal{S}$, on a : $C^{\mathcal{J}} \subseteq C_i^{\mathcal{J}}$ pour tout $i \in \{1, ..., k\}$ et $\rho^{\mathcal{J}}(C^{\mathcal{J}}) \subseteq T^{\mathcal{J}}$.

Afin de capturer la sémantique du schéma de données orienté-objet présentée ci-dessus, on a besoin d'étendre la L.D en permettant de représenter la structure de type des classes. Plus précisément, on ajoute à la base de connaissances basée sur L.D les concepts et rôles particuliers : le concept AbstractClass désigne les instances

d'une classe ; le concept RecType désigne les valeurs d'un enregistrement ; le concept SetType désigne les valeurs d'un ensemble ; le rôle value modélise l'association entre les classes et les types et le rôle member permet de spécifier le type des éléments d'un ensemble. La transformation d'un schéma de données orienté-objet en une base de connaissances basée sur L.D est définie via une fonction Γ qui transforme chaque type (ou expression de type) en une description de concept comme suit :

Si C est une classe, $\Gamma(C)$ est un nom de concept (concept atomique) ;

$$\Gamma(\text{UNION } T_1,...,T_k \text{ END}) = \Gamma(T_1) \sqcup ... \sqcup \Gamma(T_k) ;$$
$$\Gamma(\text{SET-OF } T) = \mathsf{SetType} \sqcap \forall \mathsf{member}.\Gamma(T) ;$$
$$\Gamma(\text{RECORD } A_1 : T_1,...,A_k : T_k \text{ END}) = \mathsf{RecType} \sqcap \forall \Gamma(A_1).\Gamma(T_1) \sqcap (=1\ \Gamma(A_1)) \sqcap \ldots \sqcap \forall \Gamma(A_k).\Gamma(T_k) \sqcap (=1\ \Gamma(A_k)).$$

Alors, étant donné un schéma de données orienté-objet $\mathcal{S}$, nous pouvons obtenir la base de connaissances $\Gamma(\mathcal{S})$ en transformant chaque déclaration (CLASS C IS-A $C_1,...,C_k$ TYPE-IS T) $\in \mathcal{S}$ en une axiome d'inclusion comme suit :

$$\Gamma(C) \sqsubseteq \mathsf{AbstractClass} \sqcap \Gamma(C_1) \sqcap ... \sqcap \Gamma(C_k) \sqcap \forall \mathsf{value}.\Gamma(T)$$

Une description détaillée de cette transformation se trouve dans [**CLN99 (18)**].

1.5. Conclusion

Nous avons rappelé les notions de base du domaine représentation de connaissances et présenté le formalisme de la Logique de Description dans ce chapitre. Cette logique qui se munit d'une sémantique déclarative nous permet de formaliser les taxonomies existantes du domaine de commerce électronique pour obtenir les ontologies capables d'améliorer l'interopérabilité des échanges de données. Même si la complexité des algorithmes développés pour les Logiques de Description suffisamment expressives est élevée en général, les comportements de ces algorithmes sont acceptables dans la pratique. Finalement, les correspondances entre la Logique de Description et les autres formalismes concurrents sont identifiées. Cela nous permet à la fois de réutiliser les résultats obtenus d'un autre formalisme et de déterminer un formalisme approprié pour un domaine d'application.

CHAPITRE 2

Problème de Transparence Sémantique et Extensions Existantes de la Logique de Description

La prolifération de standards et d'initiatives différents dans le domaine du Commerce Électronique, qui ont pour objectif de faciliter l'échange de documents commerciaux, fait apparaître plusieurs systèmes de classification de produits et de services. Ainsi, la composition et l'interprétation des documents échangés peuvent se baser sur différents vocabulaires. Cela nécessite un mécanisme qui permet de traduire *sémantiquement* les termes d'un vocabulaire en ceux d'un autre vocabulaire. Ce problème est connu comme *problème de transparence sémantique* [**Kim00 (46)**]. La norme ebXML présente un des efforts afin de réunir les standards existants en introduisant un vocabulaire générique pour tout le domaine commercial. Cependant, dans cette norme le problème de la transparence sémantique persiste. Une solution issue d'une étude de l'ebXML (Chapitre 5) est d'introduire la Logique de Description dans cette norme pour munir les langages de représentation utilisés d'une sémantique formelle. Et pourtant, les Logiques de Description précurseuses ne sont pas suffisamment expressives pour qu'une partie importante de connaissances impliquées dans la norme soit capturée.

Par ailleurs, la Logique de Description résulte d'un compromis entre la complexité (décidabilité) et l'expressivité des langages de représentation. En effet, si le constructeur *fermeture transitive de rôle* n'est pas considéré, la Logique de Description est exprimable dans le fragment deux-variable L^2 de la L.P.P.O. dont la décidabilité est démontrée dans [**Mor75 (57)**]. En outre, dans certains domaines d'application, il existe des connaissances qui sont inexprimables dans les Logiques de Description précurseuses, comme par exemple, les systèmes comportant les règles de Horn. D'autre part, les services d'inférences développés pour ces L.D - inférences standards - ne répondent pas à tous les besoins de calculs. Ces exigences ouvrent deux directions de recherche. La première est d'y ajouter de nouveaux constructeurs ou composants en préservant la décidabilité des inférences connues. La seconde est de développer de nouveaux services d'inférences, dits *inférences non-standards*. Ces inférences se basent sur les *algorithmes de construction* qui calculent et retournent des descriptions de concept satisfaisant certaines conditions.

Ce chapitre débute par une formulation du problème de transparence sémantique qui est l'objet de cette thèse. Par la suite, les extensions existantes pour la L.D concernant le problème formulé sont investiguées. Une synthèse sur ces travaux et une identification des techniques extensibles concluent le chapitre.

2.1. Du Problème de Transparence Sémantique à la Transformation d'Ontologies

2.1.1. Notion d'ontologie. Les ontologies sont développées pour fournir à des sources d'information une sémantique traitable par la machine. Les acteurs différents, y inclus humain et logiciel, peuvent communiquer à travers les ontologies. A notre avis, la définition suivante caractérise le mieux l'essentiel d'une ontologie :

> "*Une ontologie est une spécification formelle, explicite d'une conceptualisation partagée*" [**Gru93 (38)**].

Une conceptualisation désigne un modèle abstrait d'un domaine d'application qui identifie les concepts pertinents de ce domaine. Une conceptualisation partagée reflète la notion selon laquelle une ontologie capture des connaissances acceptées par un groupe de personnes. Une spécification explicite signifie que le genre des concepts utilisés et les contraintes sur l'utilisation de ces concepts sont explicitement définis. Une spécification formelle implique que l'ontologie est traitable par la machine. En bref, une ontologie fournit un vocabulaire de termes et de relations pour modéliser les connaissances d'un domaine d'application.

Actuellement, il y a plusieurs langages basés sur la Logique de Description, comme par exemple DAML+OIL, OWL, qui sont employés pour représenter les ontologies d'un domaine d'application.

2.1.2. Problème de l'intégration des ontologies. Les sources d'information actuelles sont hétérogènes et distribuées. Afin d'établir un mécanisme efficace permettant de partager les informations, plusieurs problèmes techniques doivent être résolus. En effet, après avoir localisé la source d'information, l'accès aux données doit être fourni. Cela implique que chaque source déterminée est capable de communiquer avec le système qui interroge des informations. Cette communication entre de tels systèmes hétérogènes est établie à condition que la signification des informations échangées soit correctement interprétée à travers les systèmes en question. En d'autres termes, *l'interopérabilité sémantique* doit être assurée.

Selon le travail dans [**Goh97 (35)**], il y a trois causes principales suivantes qui provoquent l'hétérogénéité sémantique :

(1) Conflit de confusion : il se produit lorsque deux informations semblent avoir le même sens alors qu'elles sont réellement différentes dans des contextes différents.

(2) Conflit d'unité : il se produit lorsque les systèmes d'unité différents sont utilisés pour mesurer une valeur.

(3) Conflit de nom : il se produit lorsque les schémas de nom sont différents.

Les travaux dans [**Wac99 (69), WS01 (70)**] proposent une solution au problème en supposant que ces conflits soient causés par la différence de contexte entre les sources d'informations. Selon ces travaux, chaque ontologie est associée à des *patrons* (templates) qui représentent les informations contextuelles. Ainsi, l'interopérabilité sémantique entre les ontologies peut être atteinte à l'aide de l'application des règles de contexte qui transforment les patrons associés à une ontologie en ceux qui sont associés à une autre ontologie.

Normalement, l'intégration des ontologies consiste en étapes suivantes :

(1) Chercher et identifier la partie commune des ontologies.

(2) Annoter les sources d'informations différentes en construisant la hiérarchie de concept. Chaque concept de la hiérarchie est décrit en utilisant les concepts définis dans l'ontologie commune. Le résultat de cette étape est une ontologie qui est associée aux patrons de contexte.

(3) Traduire sémantiquement une ontologie en une autre à l'aide des patrons de contexte associés.

La réalisation de ces étapes se repose sur les hypothèses suivantes :

(1) L'interprétation d'un concept dans une ontologie est effectuée en se référant aux patrons de contexte. Ainsi, chaque conflit sémantique entre deux ontologies est considéré comme la conséquence de la référence à deux patrons de contexte différents.

(2) L'interopérabilité sémantique entre les ontologies en question est assurée par la transformation des patrons de contexte d'une ontologie en ceux d'une autre ontologie.

Or, cette approche rencontre au moins deux difficultés :

(1) La différence des langages qui servent à modéliser des ontologies. Cette différence se manifeste dans quatre aspects : syntaxe, représentation logique, sémantique des primitifs et expressivité du langage.

(2) La différence des modèles. C'est-à-dire que la différence dans la façon d'interpréter un domaine implique la différence entre des concepts et des relations spécifiés. Cette différence se manifeste non seulement au niveau de la spécification mais aussi au niveau de la conceptualisation, comme par exemple, la différence entre des définitions d'un concept. Il est très difficile d'isoler d'une ontologie les facteurs qui causent l'incompatibilité. Par conséquent, la transformation de contextes ne peut garantir l'interopérabilité.

Les conflits cités et donc, le problème d'interopérabilité sémantique peuvent être partiellement résolus en supposant que les sources d'informations des systèmes soient représentées comme des ontologies basées sur la Logique de Description (langage DAML+OIL, par exemple). En effet, les problèmes de synonymie et d'homonymie entre des termes peuvent se traduire en des problèmes de la détection des concepts

équivalents dans les ontologies en question. Cela implique que le conflit de nom peut être réglé par une inférence standard implémentée dans les ontologies.

2.1.3. Problème de Transparence Sémantique dans le Commerce Électronique. En prenant conscience de la difficulté présentée au paragraphe précédent, nous nous limitons notre champs de recherche à une version restreinte du problème de l'intégration des ontologies. Cette version, qui est connue comme problème de transparence sémantique dans le Commerce Électronique [**Kim00 (46)**], est inspirée du modèle de représentation de données dans la norme ebXML [**ebX01h (33)**]. Les détails de la norme ebXML seront décrits dans le Chapitre 5.

Dans ce modèle, on utilise une *ontologie de base* qui comporte les concepts les plus génériques du domaine commercial. Chaque sous-domaine possède une ontologie spécifique qui dérive de cette ontologie de base. Une *ontologie dérivée* permet de définir, soit un concept qui peut être une version spécialisée d'un concept dans l'ontologie de base, soit de nouveaux concepts spécifiques au sous-domaine correspondant. De plus, les ontologies dérivées peuvent utiliser des langages de Logique de Description dont les expressivités sont différentes. Cette hétérogénéité d'expressivité permet à chaque sous-domaine d'optimiser le compromis entre l'expressivité du langage de représentation et la complexité d'inférences.

Le modèle décrit ci-dessus peut répondre à un scénario d'échange de documents commerciaux dans lequel chaque acteur utilise l'ontologie dérivée correspondant à son sous-domaine pour, soit composer les documents à émettre, soit interpréter les documents reçus. L'ebXML suppose que les documents échangés soient des formulaires prédéfinis [**ebX01a (26)**], et donc, la composition et l'interprétation se traduisent en remplissage de ces documents et en explication des concepts remplis.

Par ailleurs, un des problèmes issu de cette conception est qu'il existe un concept qui correspond à plusieurs définitions réparties dans des ontologies dérivées. L'ebXML a proposé une méthode, appelée *mécanisme de contexte* [**ebX01g (32)**], qui permet de déterminer la définition la plus appropriée à partir d'informations contextuelles. Ces dernières sont représentées sous forme de règle, dites *règles de contexte* [**ebX01a (26)**]. L'antécédent de ces règles est une formule logique conjonctive dont la valeur est déterminée par les valeurs contextuelles. Le conséquent de ces règles introduit une nouvelle définition d'un concept correspondant à ces valeurs contextuelles. Notons que le mécanisme de contexte présenté dans ebXML est différent de la méthode de transformation de contexte par les règles de contexte, qui est proposée dans [**Wac99 (69)**]. En effet, les règles de contexte issues de l'ebXML effectuent certaines modifications de la définition d'un concept dans l'ontologie. Les opérations pour ces modifications sont déclenchées par des valeurs contextuelles. Cela signifie qu'afin d'atteindre l'interopérabilité sémantique dans l'ebXML, il y aura certaines répercussions au niveau de la conception.

Intuitivement, le problème cité du modèle de représentation de données dans la norme ebXML implique deux instances suivantes :

(1) Une ontologie dérivée est utilisée pour interpréter d'un document composé des concepts d'une autre ontologie dérivée en supposant que les deux langages sur lesquels ces ontologies se basent soient différents tels qu'un langage est plus expressif que l'autre. Si le langage de l'ontologie récepteur est plus expressif que celui d'émetteur, alors l'équivalence et la subsomption entre les concepts définis dans l'ontologie récepteur et les concepts à interpréter peuvent être détectées par l'inférence de subsomption. Sinon, on essaie de calculer la description de concept la plus proche de la définition du concept à interpréter selon une certaine mesure.

(2) Dans le cas où les informations contextuelles sont prises en compte, les règles de contexte sont déclenchées lorsque les conditions de ces règles sont remplies par les valeurs contextuelles. Les nouvelles définitions des concepts concernés sont obtenues des applications de règles de contexte. En d'autres termes, le conséquent d'une règle de contexte est une opération permettant de modifier la définition d'un concept. Cette modification reflète le partage de la compréhension des concepts concernés entre les acteurs, car une fois la transaction entre deux acteurs correspondant à deux ontologies dérivées est établie, chaque référence au concept modifié implique la nouvelle définition. D'autre part, si l'on considère une ontologie comme un TBox, les conjoncts constituant l'antécédent d'une règle de contexte sont les assertions dans le ABox correspondant à ce TBox. Par conséquent, la modification d'une définition dans le TBox peut provoquer une révision d'assertions dans le ABox [**Neb90a (58)**], et cette révision peut déclencher, à son tour, une autre règle de contexte.

Plus précisément, soient $\Delta_{\mathcal{T}}$ le TBox de base dans lequel tous les concepts, tous les rôles primitifs et génériques sont introduits, et L le langage de Logique de Description utilisé dans $\Delta_{\mathcal{T}}$. Soient $\Delta_{\mathcal{T}_1}$ le TBox d'un émetteur, $\Delta_{\mathcal{T}_2}$ le TBox d'un récepteur qui sont dérivés de $\Delta_{\mathcal{T}}$, et L_1, L_2 les langages utilisés respectivement dans $\Delta_{\mathcal{T}_1}, \Delta_{\mathcal{T}_2}$. Supposons que L_1, L_2 soient plus expressifs que L. La première instance du problème peut être reformulée comme suit :

Soit $A_e := C_e$ le concept reçu de l'émetteur, $[A_e := C_e] \in \Delta_{\mathcal{T}_1}$ où A_e est le nom de concept et C_e est le concept de définition.

(1) S'il existe un concept $[A := C] \in \Delta_{\mathcal{T}_2}$ tel que $C \equiv C_e$, alors le concept A_e est interprété comme le concept A dans $\Delta_{\mathcal{T}_2}$. Sinon,

(2) Si la description de concept C_e est exprimable par le langage L_2 dans $\Delta_{\mathcal{T}_2}$ et $A \neq A_e$ pour tout nom de concept A dans $\Delta_{\mathcal{T}_2}$, alors un nouveau concept $[A_e := C_e]$ est ajouté dans $\Delta_{\mathcal{T}_2}$. Sinon,

(3) S'il existe un nom de concept A dans $\Delta_{\mathcal{T}_2}$ tel que $A = A_e$, alors un nouveau concept $[A'_e := C_e]$ est ajouté dans $\Delta_{\mathcal{T}_2}$ et le concept A_e est interprété comme le concept ajouté A'_e dans $\Delta_{\mathcal{T}_2}$.

(4) Finalement, si la description de concept C_e est inexprimable par le langage L_2 dans $\Delta_{\mathcal{T}_2}$, alors un nouveau concept $[A'_e := C'_e]$ est ajouté dans $\Delta_{\mathcal{T}_2}$ où C'_e est la L_1-description de concept la plus spécifique par rapport à la

subsomption telle que $C_e \sqsubseteq C'_e$. Et le concept A_e est interprété comme le concept ajouté A'_e dans $\Delta_{\mathcal{T}_2}$.

Pour formuler la deuxième instance du problème de transparence sémantique, nous avons besoin de préciser certaines restrictions. Lorsque la transaction entre un émetteur et un récepteur est établie, les deux acteurs souhaitent partager la compréhension des connaissances. En d'autre termes, si deux concepts portant le même nom ont deux définitions différentes dans deux TBox différents, une révision de ces concepts est exigée pour que les deux acteurs partagent une seule définition ou se réfèrent aux deux définitions qui ne sont pas conflictuelles. Dans ce cas, la règle de contexte décide non seulement la condition de déclenchement de la révision mais aussi la procédure de cette révision.

Soient $\Delta_{\mathcal{T}}$, $\Delta_{\mathcal{A}}$ le TBox et le ABox respectivement. Soit $\Delta_{\mathcal{R}}$ l'ensemble de règles de contexte où chaque règle r est de la forme :

$$Ant(r) \Rightarrow Rev(Cons(r))$$

où $Ant(r)$ est une conjonction d'assertions du ABox, $Cons(r)$ est le concept à réviser par la règle r , Rev est une opération de révision du concept $Cons(r)$. La deuxième instance du problème peut être reformulée comme suit :

(1) Soit $R \subseteq \Delta_{\mathcal{R}}$ l'ensemble des règles initialement activées. Une règle r est activée si $Ant(r)$ est vérifiée. L'application de la règle r peut changer le TBox. Ainsi, nous désignons par $\Delta^1_{\mathcal{T}}$ le TBox suivant l'application de la règle r.

(2) Le changement du TBox peut provoquer une révision du ABox. Donc, nous désignons par $\Delta^1_{\mathcal{A}}$ le ABox suivant la révision du ABox.

(3) La révision du ABox peut changer l'ensemble des règles activées. Nous désignons par R_1 l'ensemble des règles activées suivant la révision du ABox.

(4) Supposons que le TBox ne change plus après n étapes où $n \geq 0$, et soit $\Delta^n_{\mathcal{T}}$ le TBox obtenu. En conséquence, le ABox ne change plus après n étapes et nous désignons également par $\Delta^n_{\mathcal{A}}$ le ABox obtenu.

La compréhension partagée entre les acteurs sera atteinte lorsque ce processus de transformation se termine.

Ces formulations impliquent plusieurs tâches intermédiaires à accomplir. Pour la première instance, une inférence standard, la subsomption, est utilisée afin de déterminer l'équivalence entre des descriptions de concept. Une inférence non-standard, appelée *approximation* [**BKT02a (15)**], est exigée pour calculer la L_2-description de concept qui est "la plus proche" d'une L_1-description de concept inexprimable dans le langage L_2. De plus, une autre inférence non-standard, appelée *réécriture* [**BKM00 (6)**], est envisagée pour simplifier les descriptions de concept obtenues de l'inférence d'approximation.

Pour la deuxième instance, tout d'abord un *formalisme hybride*, qui combine les trois composants $\Delta_{\mathcal{T}}$,$\Delta_{\mathcal{A}}$ et $\Delta_{\mathcal{R}}$, doit être défini pour munir ces composants d'une sémantique formelle commune basée sur la Logique de Description. Un problème important à traiter dans cette instance est la révision d'un TBox et le ABox correspondant. En général, le problème de la révision d'un ensemble de propositions, dit *révision de croyance*, dépasse le cadre de la Logique du Premier Ordre [**Gär92 (36)**]. Cependant, ce problème peut être investigué dans le cadre de la logique classique à condition que les postulats pour la révision proposés dans le canevas AGM [**AGM85 (1)**] se traduisent convenablement en ceux pour la révision d'un TBox [**Neb90a (58)**].

Il y a encore une tâche à accomplir dans la deuxième instance. Il s'agit de l'inférence d'extension de la base de connaissances, effectuée par l'application des règles de contexte. Cette inférence permet de calculer la série de transformation des composants $(\Delta^1_{\mathcal{T}},\Delta^1_{\mathcal{A}}),...,(\Delta^n_{\mathcal{T}},\Delta^n_{\mathcal{A}})$ résultant des applications de règles dans Δ_R. Notamment, la terminaison de la transformation des composants et l'unicité des composants finaux doivent être prises en compte dans cette inférence.

2.2. Extensions Envisagées de la Logique de Description

Dans cette section, nous présentons les extensions de la Logique de Description servant à compléter le modèle de données dans la norme ebXML. Même si ces extensions résultent de différentes motivations, elles fournissent des techniques de base visant à résoudre les deux instances formulées du problème. En effet, un méta-modèle des *Composants de Base* défini dans la norme ebXML [**ebX01f (31)**] est décrit en langage UMM[1] [**Cla00 (21)**] qui est un méta-langage du UML[2]. Ces Composants de Base forment un vocabulaire générique pour toute la communication du domaine commercial. La description de chaque composant de base, qui est instanciée de ce méta-modèle, peut être représentée par des diagrammes de classe UML. Concernant la formalisation de la sémantique des diagrammes de classe UML, le travail dans [**CCD$^+$02 (17)**] a proposé un langage de Logique de Description très expressif $\mathcal{DLR}$ capable de capturer la sémantique de ces diagrammes. Ce travail est la première extension introduite dans cette section. Par la suite, la technique de base pour l'approche structurelle abordée dans la Section 1.3.4 sera présentée en détail. Cette technique sert à développer des algorithmes de construction, connus comme inférences non-standard, non seulement pour la première instance mais aussi pour la deuxième instance du problème cité. Ainsi, la deuxième extension à présenter dans la section est le développement de ces inférences non-standard. L'introduction d'un travail traitant la combinaison des règles de Horn et une Logique de Description expressive, dit langage CARIN [**LR98 (53)**], terminera la section. Ce travail présentant un algorithme de décision sur ce formalisme hybride est un exemple d'utilisation de l'approche de tableaux abordée dans la Section 1.3.4.

[1]Unified Modelling Methodology
[2]Unified Modelling Language

2.2.1. Formalisation des diagrammes de classes UML. Les diagrammes UML sont connus comme un formalisme générique qui permet de modéliser les applications dans la plupart de domaines. Et pourtant, la sémantique de ce formalisme n'est pas décrite de façon déclarative. Cela empêche de développer des inférences qui permettent de détecter des propriétés formelles des diagrammes de classe UML. C'est le concepteur qui prend la responsabilité de détecter, par exemple, l'inconsistance et la redondance des diagrammes. Cette tâche dépassera les manipulations manuelles si les diagrammes à vérifier deviennent de plus en plus complexes. Pour cette raison, la recherche qui vise à munir ces diagrammes d'un mécanisme d'inférences automatique montre son importance. En sachant qu'un tel mécanisme d'inférences ne peut que se développer en basant sur une sémantique formelle, le premier travail pour cet objectif consiste donc à formaliser la sémantique des diagrammes de classe UML. Cette sémantique est capturée par une Logique de Description très expressive $\mathcal{DLR}$ proposée dans [**CCD$^+$02 (17)**].

La sémantique des diagrammes de classe UML est constituée des aspects suivants : l'attribut d'une classe ; l'association et l'agrégation des classes ; la généralisation et la spécialisation des classes ; et les contraintes sur un diagramme. Lorsque ces aspects sémantiques d'un diagramme de classe sont formalisés, les services d'inférence suivants sont envisageables :

(1) Vérifier la consistance d'une classe ou d'un diagramme de classe. Une classe est consistante si l'ensemble d'instances de cette classe n'est pas vide. En d'autres termes, cette classe peut être instanciée sans violer les contraintes imposées sur elle. Un diagramme de classes est consistent si chaque classe peut être instanciée sans violer les contraintes imposées sur ce diagramme.

(2) Vérifier la subsomption et l'équivalence entre des classes. Une classe est subsumée par une autre classe si l'ensemble d'instances de la classe subsumante inclut l'ensemble d'instances de la classe subsumée. Deux classes sont équivalentes si les deux ensembles d'instances des deux classes sont égales, lorsque les contraintes imposées sur le diagramme de classe sont satisfaites. La détermination de l'équivalence permet de diminuer la complexité du diagramme.

(3) Expliciter les conséquences logiques. Une propriété est une conséquence logique d'une autre propriété si cette propriété est vérifiée lorsque toutes les contraintes spécifiées dans le diagramme sont satisfaites. La détermination de la conséquence logique permet d'une part de diminuer la complexité du diagramme en effaçant les contraintes qui sont impliquées logiquement d'autres contraintes, d'autre part d'expliciter des propriétés implicites dans un diagramme. Cela améliore la lisibilité du diagramme.

Langage $\mathcal{DLR}$. Le langage $\mathcal{DLR}$ est connu comme une Logique de Description très expressive qui autorise non seulement les constructeurs du langage $\mathcal{ALC}$ décrit dans le Chapitre 1, mais aussi les constructeurs permettant de capturer les notions définies dans la base de données : la contrainte d'identification et la dépendance fonctionnelle. Pour cela, la relation $n - aire$ est introduite dans la langage $\mathcal{DLR}$.

Avec les nouveaux constructeurs, la syntaxe et la sémantique du langage sont décrites comme suit :

C, D	$\rightarrow$	$\top \mid$	(concept universel ou top)
		$A \mid$	(nom de concept)
		$\leq k[i]R$	(restriction de cardinalité)
		$C \sqcap D \mid$	(conjonction de concept)
		$\forall r.C \mid$	(restriction universelle)
		$\exists r.C \mid$	(restriction existentielle)
		$\neg C \mid$	(négation complète)
R	$\rightarrow$	$\top_n \mid$	(produit $n - aire$ de top)
		$P \mid$	(rôle primitif)
		$(i/n : C) \mid$	(restriction de cardinalité de rôle)
		$\neg R \mid$	(négation complète de rôle)
		$R_1 \sqcap R_2$	(conjonction de rôles)

FIG. 2.2.1. Syntaxe des $\mathcal{DLR}$-descriptions de Concept

Dans la Figure 2.2.1, on désigne par i un composant d'une relation *n-aire* et par k un nombre entier non-négatif. On ne considère que les concepts et le rôles bien typés *i.e* l'expression $R_1 \sqcap R_2$ est bien typé si les relations R_1, R_2 ont la même arité et $i \leq n$ lorsque l'on désigne par i un composant d'une relation *n-aire*. La sémantique de nouveaux constructeurs présentés dans la Figure 2.2.1 est décrite par le tableau suivant grâce à une interprétation $\mathcal{I}$:

Syntaxe	Sémantique
$\top_1$	$\top_1^{\mathcal{I}} = \Delta^{\mathcal{I}}$
$\top_n$	$\top_n^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^n$
P	$P^{\mathcal{I}} \subseteq \top_n^{\mathcal{I}}$
$(i/n : C)$	$\{t \in \top_n^{\mathcal{I}} \mid t[i] \in C^{\mathcal{I}}\}$
$(\neg R)$	$\top_n^{\mathcal{I}} \setminus R^{\mathcal{I}}$
$(R_1 \sqcap R_2)$	$R_1^{\mathcal{I}} \cap R_2^{\mathcal{I}}$
$(\leq k[i]R)$	$\{x \in \Delta^{\mathcal{I}} \mid card\{t \in R_1^{\mathcal{I}} \mid t[i] = x\} \leq k\}$

Notons que $\top_n$ n'est pas le produit cartésien $\top^n$. Il est possible donc que $\top_n^{\mathcal{I}} \subsetneq (\Delta^{\mathcal{I}})^n$. Pour cette raison, le constructeur "$\neg$" pour les relations est utilisé pour exprimer la différence entre des relations plutôt que le complément. Nous allons utiliser quelques abréviations suivantes : $\exists[i]R$ pour $\geq 1[i]R$ et $\forall[i]R$ pour $\neg\exists[i]\neg R$.

Maintenant, nous pouvons définir une base de connaissances basée sur le langage $\mathcal{DLR}$. Une B.C $\mathcal{DLR}$ comporte un ensemble fini des assertions d'inclusion suivantes :

$$R_1 \sqsubseteq R_2 \text{ et } C_1 \sqsubseteq C_2$$

où R_1 et R_2 ont la même arité.

En particulier, une B.C. $\mathcal{DLR}$ autorise également les assertions exprimant la contrainte d'identification et la dépendance fonctionnelle. D'abord, l'assertion d'identification sur un concept C est définie comme suit :

$$(\textbf{id}\ C\ [i_1]R_1,...,[i_h]R_h)$$

où chaque R_j est une relation et chaque i_j est un composant de R_j. L'assertion d'identification signifie que si a, b sont les instances du concept C telles que a est le composant i_j-ème d'un tuple t_j de R_j, b est le composant i_j-ème d'un tuple s_j de R_j pour $j \in \{1, ..., h\}$, et si tous les composants, sauf le composant i_j-ème, des deux tuples s_j, t_j sont égaux, alors a coïncide avec b.
L'assertion de dépendance fonctionnelle sur une relation R est écrite sous forme :

$$(\textbf{fd}\ R\ i_1,...,i_h \rightarrow j)$$

où $h \geq 2$ et on désigne par $i_1, ..., i_h, j$ des composants de R. Cette assertion signifie que si les composants i_1-ème, ..., i_h-ème de deux tuples sont égaux, alors les composants j-ème de ces deux tuples sont égaux.

A partir de la sémantique des constructeurs et la signification des assertions d'identification et de dépendance fonctionnelle, on peut définir normalement la sémantique d'une base de connaissance basée sur $\mathcal{DLR}$ grâce à une interprétation $\mathcal{I}$ [**CCD$^+$02 (17)**].

Par ailleurs, le travail dans [**CDL01 (19)**] a prouvé que l'inférence de satisfiabilité de la B.C et de concept dans le langage $\mathcal{DLR}$ est EXPTIME-complet. En particulier, si $h = 1$ dans l'assertion de dépendance fonctionnelle, alors l'inférence dans ce langage pourrait devenir indécidable [**CDL01 (19)**].

Formalisation des classes. Une classe UML se compose des trois parties : nom, attributs, opérations. La Figure 2.2.2 montre la représentation graphique d'une classe UML.

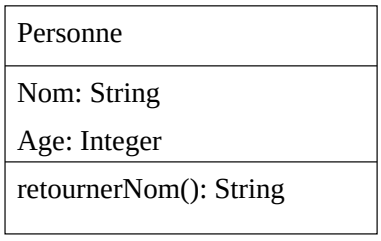

FIG. 2.2.2. Une classe UML

Un attribut a avec le type T, qui est défini dans la classe C, implique que chaque instance de la classe C associe à un ensemble d'instances de T. Chaque attribut est unique dans une classe mais deux classes peuvent avoir le même attribut. Une multiplicité $[i..j]$ pour un attribut a spécifie que a associe à chaque instance C au moins i instances de T et au plus j instances de T. Par ailleurs, puisque le code des méthodes

n'est pas spécifié dans le diagramme de classe, alors nous ne nous intéressons pas à la formalisation des méthodes ici. Toutefois, on peut formaliser la signature des méthodes qui est définie dans le diagramme de classe UML [**CDL01 (19)**].

A partir de cette signification, nous pouvons utiliser l'assertion $\mathcal{DLR}$ suivante pour capturer d'abord un attribut a avec le type T de la classe C :

$$C \sqsubseteq \forall[1](a \Rightarrow (2 : T))$$

c'est-dire-que pour chaque instance c de la classe C, tous les objets qui sont les premiers composants de la relation R sont les instances de la classe T.

Ensuite, l'assertion suivante capture la multiplicité $[i..j]$:

$$C \sqsubseteq (\geq i[1]a) \sqcap (\leq j[1]a)$$

c'est-dire-que chaque instance c de la classe C participe au moins i fois et au plus j fois à la relation a via le premier composant.

Formalisation des associations et des agrégations. Une association dans UML est une relation entre des instances des classes. Une association binaire A entre deux classes C_1 et C_2 est visualisée dans la Figure 2.2.3.

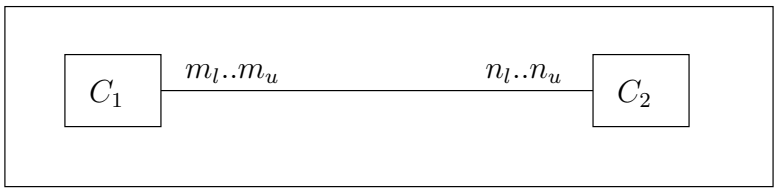

FIG. 2.2.3. Une association binaire dans UML

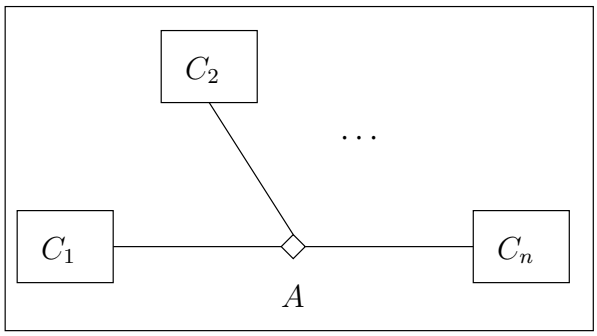

FIG. 2.2.4. Une association n-aire dans UML

Une association est unique dans un diagramme de classe. La multiplicité $n_l..n_u$ sur la relation binaire spécifie que chaque instance de la classe C_1 peut participer au moins n_l fois et au plus n_u fois à la relation A. Une association a souvent une classe associée décrivant les propriétés de la association. Notons que la formalisation d'une

association est différente de celle d'un attribut car l'association est unique dans un diagramme. Ce n'est pas le cas pour l'attribut.

Une agrégation dans UML est une *association binaire* entre des instances de deux classes. Cette relation spécifie que chaque instance d'une classe se compose d'un ensemble d'instances d'une autre classe. Dans un diagramme UML, les associations et les agrégations spécifient des relations entre des classes. Cependant, les agrégations sont binaires alors que les associations sont *n-aire*. De plus, les agrégations n'ont pas de classe associée alors que les associations en ont une. Par conséquent, on n'a besoin que de formaliser l'association.

L'assertions suivantes formalisent l'association entre n classes $C_1, ..., C_n$ sans classe associée et la multiplicité comme dans la Figure 2.2.3 :

$$A \sqsubseteq (1 : C_1) \sqcap ... \sqcap (n : C_n)$$

$$C_1 \sqsubseteq (\geq n_l[1]A) \sqcap (\leq n_u[1]A)$$

$$C_2 \sqsubseteq (\geq m_l[1]A) \sqcap (\leq m_u[1]A)$$

Pour formaliser l'association entre n classes $C_1, ..., C_n$ avec la classe associée (Figure 2.2.5), supposons qu'il y a n relations binaires r_1,...,r_n correspondant aux composant de l'association A. Cette association doit spécifier un concept A tel qu'il a *uniquement* les composants r_1,...,r_n de l'association A. De plus, ce concept A doit spécifier la classe à laquelle chaque composant appartient. Donc,

$$\begin{aligned} A \quad \sqsubseteq \quad & \exists[1]r_1 \sqcap (\leq 1[1]r_1) \sqcap \forall[1](r_1 \Rightarrow (2 : C_1)) \sqcap \\ & \vdots \\ & \exists[1]r_n \sqcap (\leq 1[1]r_n) \sqcap \forall[1](r_n \Rightarrow (2 : C_n)) \end{aligned}$$

où $\exists[1]r_i$ spécifie que le concept A doit avoir tous les composants r_1,...,r_n de l'association A ; $(\leq 1[1]r_i)$ spécifie que chaque composant prend une valeur simple ; et $\forall[1](r_n \Rightarrow (2 : C_n))$ spécifie la classe à laquelle chaque composant appartient.
Finalement, pour assurer que chaque instance de la classe A représente un tuple distingué dans le produit cartésien $C_1 \times ... \times C_n$, on a besoin d'une assertion d'identification :

$$(\textbf{id}\ C\ [1]r_1,...,[1]r_n)$$

Formalisation de la généralisation et de l'héritage. Dans UML, les instances de la classe fille héritent toutes les propriétés de la classe mère. De plus, les instances de la classe fille satisfont d'autres propriétés qui ne sont pas vérifiées dans la classe mère. Par conséquent, la généralisation dans UML peut être capturée par la subsomption dans le langage $\mathcal{DLR}$. Si C_2 est une généralisation de C_1 dans UML, on peut exprimer cette relation par l'assertion suivante :

$$C_1 \sqsubseteq C_2$$

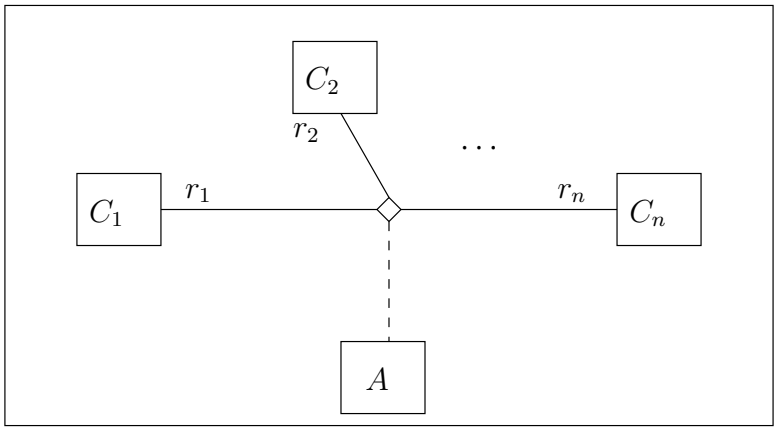

FIG. 2.2.5. Une association n-aire avec la classe associée dans UML

De plus, la contrainte disjointe entre les classes C_1,...,C_n peut être formalisée comme suit :

$$C_i \sqsubseteq \bigsqcap_{j=i+1}^{n} \neg C_j \text{ pour chaque } i \in \{1, ..., n\}$$

alors que la contrainte couvrante peut être exprimée comme suit :

$$C \sqsubseteq \bigsqcup_{j=1}^{n} C_j$$

Formalisation des contraintes. Dans UML, il est possible d'ajouter des informations à la classe en utilisant les contraintes. En général, une contrainte est utilisée pour exprimer informellement des informations qui ne peuvent pas être exprimées par d'autres constructions du diagramme UML. Par exemple, pour formaliser la notion *clé* qui est très utilisée dans la base de données. Le langage $\mathcal{DLR}$ peut servir à exprimer cette notion. En effet, si un ensemble des attributs a_1,...,a_n est une clé de la classe C, cela signifie qu'il n'existe pas de couple d'instances de C dont les attributs a_1,...,a_n prennent la même valeur. L'assertion suivante capture cette signification :

$$(\textbf{id } C \ [1]a_1,...,[1]a_n)$$

Exemple. Le TBox dans la Figure 1.3.4 peut se traduire en diagramme de classe UML dans la Figure 2.2.6.
Le diagramme de classe dans la Figure 2.2.6 peut être formalisé en langage $\mathcal{DLR}$ comme suit :

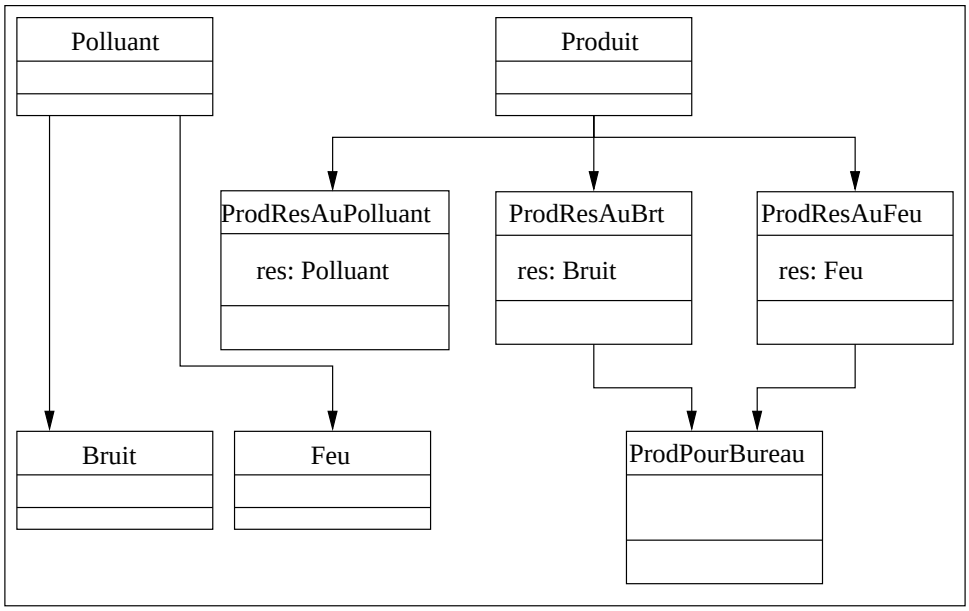

FIG. 2.2.6. Un diagramme de classe UML

Bruit	$\sqsubseteq$	Polluant
Feu	$\sqsubseteq$	Polluant
ProdResAuPolluant	$\sqsubseteq$	Produit $\sqcap \forall[1](\textit{résister} \Rightarrow(2 :\text{Polluant}))$
ProdResAuPolluant	$\sqsubseteq$	$\geq 1[1]\textit{résister}$
ProdResAuBrt	$\sqsubseteq$	Produit $\sqcap \forall[1](\textit{résister} \Rightarrow(2 :\text{Bruit}))$
ProdResAuBrt	$\sqsubseteq$	$\geq 1[1]\textit{résister}$
ProdResAuFeu	$\sqsubseteq$	Produit $\sqcap \forall[1](\textit{résister} \Rightarrow(2 :\text{Feu}))$
ProdResAuFeu	$\sqsubseteq$	$\geq 1[1]\textit{résister}$
ProdPourBureau	$:=$	ProdResAuBrt $\sqcap$ ProdResAuFeu

Maintenant, si on souhaite ajouter à la base de connaissance un nouveau concept ProdPourHabitation défini comme suit :

$$\text{ProdPourHabitation} := \text{ProdPourBureau} \sqcap \text{ProdResAuPolluant}$$

Par l'inférence de subsomption dans le langage $\mathcal{DLR}$, on peut conclure que le nouveau concept est équivalent au concept existant ProdPourBureau car le concept Polluant subsume le concept Bruit et donc, le concept ProdResAuPolluant subsume le concept ProdPourBureau.

2.2.2. Inférences non-standards . Les inférences s'appuyant sur les algorithmes de décision suscitent beaucoup d'intérêts dès que les premiers jours du développement des système basés sur la Logique de Description. Ces algorithmes sont investigués en profondeur au niveau de la capacité expressive du langage en question

et au niveau de la complexité. Le développement de la technique des tableaux permet aux études de ces algorithmes d'aboutir à des résultats remarquables. Les techniques d'optimisation résultant de la technique des tableaux assurent la réussite de plusieurs systèmes de la représentation de connaissances, comme par exemple FaCT[3] et RACER[4]. Et pourtant, au long du développement de ces systèmes, on se rend compte que des inférences s'appuyant sur les algorithmes de construction sont aussi nécessaires pour la construction et la maintenance de bases de connaissances larges. Une caractéristique de ces algorithmes est de permettre de *construire* une description de concept qui satisfasse certaines conditions au lieu de décider si une description de concept satisfait une propriété. En général, un algorithme de construction peut être transformé en un algorithme de décision à condition que l'espace d'objets à laquelle le résultat retourné par l'algorithme de construction appartient, soit fini [**Wil86 (72)**]. Cependant, l'ensemble des descriptions de concept envisageables d'une telle inférence dans un langage L.D suffisamment expressif est souvent infini ou très grand. Dans ce contexte, une nouvelle famille d'inférences est étudiée, appelée *inférence non-standard*. Dans cette section, nous nous concentrerons sur deux inférences non-standards : le plus spécifique commun subsumeur (*lcs*) et l'approximation d'une L_1-description de concept par une L_2-description de concept où L_2 est moins expressif que L_1 (*approx*).

Approche structurelle pour la subsomption dans le langage $\mathcal{ALE}$. La technique des tableaux s'est montrée avec le succès pour les inférences standards dans des langages très expressifs. Toutefois, cette technique est inapplicable aux inférences non-standards car on ne sait pas comment traduire le problème de la construction d'une description de concept en problème de la satisfiabilité d'une description de concept. La difficulté montre que l'approche issue de la sémantique pure n'est pas appropriée aux algorithmes de construction. Cela évoque une autre approche dans laquelle les formules, *i.e* descriptions de concepts, sont transformées en une représentation particulière basée sur la syntaxe de la description de concept. Cette représentation devrait permettre d'une part de décider les relations entre des descriptions de concept correspondantes, comme par exemple la subsomption, d'autre part de construire une description de concept à partir de certaines propriétés "sémantiques".

Dans la suite, nous présentons une telle représentation pour les $\mathcal{ALE}$-descriptions de concept. Ensuite, une procédure de décision pour la subsomption entre des $\mathcal{ALE}$-descriptions de concept, qui est un résultat essentiel pour établir le calcul du plus spécifique subsumeur commun de $\mathcal{ALE}$-descriptions de concept, est décrite.

Définition 2.2.1. (Arbre de description d'une $\mathcal{ALE}$-description de concept)
Un arbre de description d'une $\mathcal{ALE}$-description de concept $\mathcal{G} = (V, E, n_0, l)$ est un arbre défini comme suit :

- V est un ensemble fini des nœuds de $\mathcal{G}$;
- $E \subseteq V \times (N_R \cup \forall N_R) \times V$ est un ensemble fini des arcs étiquetés par des noms de rôles r ($\exists$-arcs) ou par $\forall r$ ($\forall$-arcs) ; $\forall N_R := \{\forall r \mid r \in N_R\}$;
- n_0 est la racine de $\mathcal{G}$;

[3]Fast Classification of Terminologies

[4]Renamed Abox and Concept Expression Reasoner

– l est une fonction d'étiquetage qui s'applique des nœuds dans V, dans les ensembles finis $\{P_1, ..., P_k\}$ où chaque P_i, $1 \leq i \leq k$, a une des formes suivantes : $P_i \in N_C$, $P_i = \neg P$ pour $P \in N_C$, ou $P_i = \bot$. L'étiquette vide correspond au top-concept.

Exemple 2.2.2. Soient C une $\mathcal{ALE}$-description de concept
$C := \exists r.(\forall r.\forall r.P_3^0) \sqcap \exists r.(\forall r.\forall r.P_3^1) \sqcap$
$\forall r.(\exists r.\forall r.P_2^0 \sqcap \exists r.\forall r.P_2^1 \sqcap \forall r.(\exists r.P_1^0 \sqcap \exists r.P_1^1))$
et l'arbre de description du C est illustré dans la Figure 2.2.7.

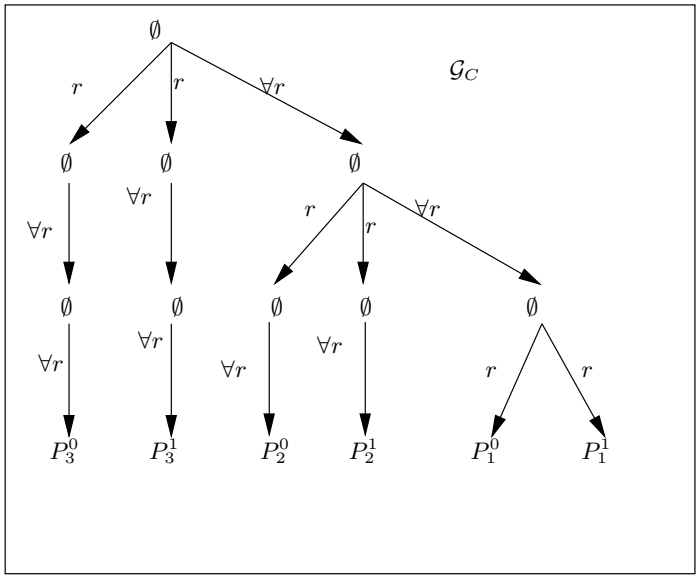

FIG. 2.2.7. Arbre de description

Chaque $\mathcal{ALE}$-description de concept peut être transformé en un arbre de description $\mathcal{G}_C = (V, E, n_0, l)$ comme suit. D'abord, la description de concept C est écrite sous forme :

$$C \equiv P_1 \sqcap ... \sqcap P_n \sqcap \exists r.C_1 \sqcap ... \sqcap \exists r_n.C_n \sqcap \forall s_1.D_1 \sqcap ... \sqcap \forall s_k.D_k$$

où $P_i \in N_C \cup \{\neg P | P \in N_C\} \cup \{\top, \bot\}$.
Ensuite, l'ensemble des concepts primitifs de la conjonction au top-level de C correspond à l'étiquette $l(v_0)$ de $\mathcal{G}_C$. Chaque restriction existentielle $\exists r_i.C_i$ dans cette conjonction correspond à un r_i-successeur qui est la racine de l'arbre correspondant au C_i. De même, chaque restriction universelle $\forall s_i.D_i$ dans cette conjonction correspond à un $\forall s_i$-successeur qui est la racine de l'arbre correspondant au D_i. Inversement, chaque $\mathcal{ALE}$-arbre de description $\mathcal{G} = (V, E, n_0, l)$ peut être transformé en la $\mathcal{ALE}$-description de concept $C_\mathcal{G}$ par récurrence en fonction de l'hauteur de $\mathcal{G}$. En effet, les concepts primitifs dans l'étiquette de v_0 correspondent aux concepts primitifs dans la conjonction au top-level de C. Chaque r-successeur ($\forall r$-successeur) v de v_0 correspond à une restriction existentielle $\exists r.C'$ (restriction universelle $\forall r.C'$) où C' est

une description de concept obtenue de la transformation du sous-arbre dont la racine est v. Notons qu'une étiquette vide est transformée en le concept top. Finalement, la sémantique d'arbre $\mathcal{G}$ est déterminée par la description de concept $C_{\mathcal{G}}$.

Grâce à la représentation d'arbre pour les $\mathcal{ALE}$-descriptions de concept, nous pouvons caractériser la subsomption entre deux descriptions de concept par un *homomorphisme* qui est défini comme suit.

Définition 2.2.3. (Homomorphisme entre deux arbres de $\mathcal{ALE}$-descriptions de concept)
Une application $\varphi : N_H \rightarrow N_G$ d'un $\mathcal{ALE}$-arbre de description $\mathcal{H} = (N_H, E_H, m_0, l_H)$ dans un $\mathcal{ALE}$-arbre de description $\mathcal{G} = (N_G, E_G, n_0, l_G)$ est appelée un homomorphisme ssi, les conditions suivantes sont satisfaites :

(1) $\varphi(m_0) = n_0$

(2) Pour tout $n \in N_H$, on a $l_H(n) \subseteq l_G(\varphi(n))$ ou $\bot \in l_G(\varphi(n))$;

(3) Pour tout $nrm \in E_H$, soit $\varphi(n)r\varphi(m) \in E_G$, soit $\varphi(n) = \varphi(m)$ et $\bot \in l_G(\varphi(n))$; et

(4) Pour tout $n\forall rm \in E_H$, soit $\varphi(n)\forall r\varphi(m) \in E_G$, soit $\varphi(n) = \varphi(m)$ et $\bot \in l_G(\varphi(n))$.

Cependant, l'utilisation *directe* d'un homomorphisme entre deux arbres de description ne nous permet pas d'obtenir une caractérisation de subsomption correcte et complète. C'est pourquoi les $\mathcal{ALE}$-descriptions de concept doivent être *normalisées* avant de les transformer en arbres de description. A partir de ces arbres de description obtenus des descriptions de concept normalisées, la subsomption peut être caractérisée *correctement* et *complètement* par l'existence d'un homomorphisme entre des arbres de description en question. La *forme normale* d'une $\mathcal{ALE}$-description de concept C est obtenue en s'appliquant exhaustivement les règles suivantes.

Définition 2.2.4. (Règles de normalisation d'une $\mathcal{ALE}$-description de concept)
Une $\mathcal{ALE}$- description de concept est appelée *normale* si les règles suivantes sont exhaustivement appliquées

(1) $\forall r.E \sqcap \forall r.F \rightarrow \forall r.(E \sqcap F)$

(2) $\forall r.E \sqcap \exists r.F \rightarrow \forall r.\ E \sqcap \exists r.(E \sqcap F)$

(3) $\forall r.\top \rightarrow \top$

(4) $E \sqcap \top \rightarrow E$

(5) $P \sqcap \neg P \rightarrow \bot$ pour chaque $P \in N_C$

(6) $\exists r.\bot \rightarrow \bot$

(7) $E \sqcap \bot \rightarrow \bot$

où E, F sont les $\mathcal{ALE}$-descriptions de concept et $r \in N_R$.

Puisque chaque règle de normalisation préserve l'équivalence, alors les $\mathcal{ALE}$-descriptions de concept obtenues de la normalisation sont équivalentes à celles originales. On désigne par $\mathcal{G}(C)$ l'arbre de description obtenu d'une description de concept C sans appliquer les règles de normalisation. Et pourtant, $C_{\mathcal{G}_C} \equiv C$ et il est possible que $C_{\mathcal{G}_C} \neq C$ où $\mathcal{G}_C$ est désigné pour l'arbre de description obtenu d'une description de concept C normalisée.

La normalisation d'une $\mathcal{ALE}$-description de concept peut accroire sa taille de façon exponentielle en fonction de la taille originale. En effet, la règle de normalisation 2. effectue les copies de sous-arbres qui accroît la taille de la description de concept à normaliser.

A partir des notions introduites, la subsomption dans le langage $\mathcal{ALE}$ peut être caractérisée par l'utilisation de l'homomorphisme comme suit :

Théorème 2.2.5. [**BKM99 (5)**] *Soient* C, D *des $\mathcal{ALE}$-descriptions de concept. Alors* $\mathrm{C} \sqsubseteq \mathrm{D}$ *si et seulement s'il existe un homomorphisme de $\mathcal{G}_D$ dans $\mathcal{G}_C$.*

D'après [**DHL$^+$92 (22)**], la subsomption dans le langage $\mathcal{ALE}$ est un problème NP-complet et la vérification de l'existence d'un homomorphisme entre les arbres de description est un problème polynômial. Cela explique l'accroissement exponentiel de la taille d'une $\mathcal{ALE}$-description de concept normalisée par rapport à la taille de celle originale.

Le plus spécifique subsumeur commun (lcs).

Définition 2.2.6. Soient C_1,...,C_k des descriptions de concept dans un langage de logique de description L. Une L-description de concept C est le subsumeur commun le plus spécifique de C_1,...,C_k (en abrégé $lcs(C_1,...,C_k)$) ssi

(1) $C_i \sqsubseteq C$ pour tout $i \in \{1, ..., k\}$, et

(2) C est une description de concept la plus spécifique qui satisfait cette propriété, *i.e* si C' est une description de concept telle que $C_i \sqsubseteq C'$ pour tout $i \in \{1, ..., k\}$, alors $C \sqsubseteq C'$.

Notons que pour les langages qui contiennent le constructeur de disjonction, par exemple $\mathcal{ALC}$, le plus spécifique subsumeur commun est trivialement calculé comme suit :

$$lcs(C_1,...,C_k) \equiv C_1 \sqcup ... \sqcup C_k$$

De plus, pour le langage $\mathcal{ALE}$, le plus spécifique subsumeur commun existe uniquement. En effet, supposons que $C_i \sqsubseteq C'$ pour tout $i \in \{1, ..., k\}$. Par définition de lcs, on obtient :

$$lcs(C_1,...,C_k) \sqsubseteq lcs(C_1,...,C_k) \sqcap C'$$

Par conséquent, $lcs(C_1,...,C_k) \equiv C'$.

Remarque 2.2.7. (Propriétés de *lcs*)
A partir de la Définition 2.2.6, on a :

$$lcs(C_1,...,C_k) \equiv lcs(C_k,\ lcs(...lcs(C_2,\ C_1)\ ...)$$

où C_1,...,C_k sont des L-descriptions de concept.

Cette propriété permet de partitioner le calcul du *lcs* de k descriptions de concept en des calculs *lcs* binaires. Cependant, dans la plupart des cas, les calculs du *lcs* binaire et du *lcs n-aire* appartiennent à des classes de complexité différentes.

Afin de calculer lcs dans le langage $\mathcal{ALE}$, on a besoin d'une notion, dite *produit* de deux arbres de descriptions.

Définition 2.2.8. Soient C, D des $\mathcal{ALE}$-descriptions de concept et $\mathcal{G}$= (V_G, E_G, v_0, l_G), $\mathcal{H}$= (V_H, E_H, w_0, l_H) les arbres de descriptions de C, D respectivement.

- Si l_G =$\{\bot\}$ (l_H =$\{\bot\}$), alors on définit le produit $\mathcal{G} \times \mathcal{H}$ en remplaçant chaque nœud w (ou nœud v) dans $\mathcal{H}$ (ou dans $\mathcal{G}$) par (v_0, w) (ou par (v, w_0)),
- Sinon, le produit est défini par récurrence sur la profondeur des arbres. Le nœud (v_0, w_0) étiqueté par $l_G(v_0) \cap l_H(w_0)$ est la racine de $\mathcal{G} \times \mathcal{H}$. Pour chaque r-successeur v de v_0 dans $\mathcal{G}$ et chaque r-successeur w de w_0 dans $\mathcal{H}$, on obtient un r-successeur (v, w) de (v_0, w_0) dans $\mathcal{G} \times \mathcal{H}$ qui est la racine du produit de $\mathcal{G}(v)$ et $\mathcal{H}(w)$ défini récursivement. Pour chaque $\forall r$-successeur v de v_0 dans $\mathcal{G}$ et chaque $\forall r$-successeur w de w_0 dans $\mathcal{H}$, on obtient un $\forall r$-successeur (v, w) de (v_0, w_0) dans $\mathcal{G} \times \mathcal{H}$ qui est la racine du produit de $\mathcal{G}(v)$ et $\mathcal{H}(w)$ défini récursivement.

Cette définition fournit une méthode directe pour calculer le plus spécifique le subsumeur commun des $\mathcal{ALE}$-descriptions de concept à partir de leur représentation des arbres de description. Le théorème suivant établit la correction de cette méthode.

Théorème 2.2.9. **[BKM99 (5)]** *Soient C, D des $\mathcal{ALE}$-descriptions de concept et $\mathcal{G}_C$, $\mathcal{G}_D$ les $\mathcal{ALE}$-arbres de description correspondants. Alors, le produit $\mathcal{G}_C \times \mathcal{H}_D$ est le plus spécifique le subsumeur commun de C et de D.*

Nous nous rendons compte que la taille du produit $\mathcal{G} \times \mathcal{H}$ de deux arbres $\mathcal{G}$ et $\mathcal{H}$ est le produit de la taille de $\mathcal{G}$ et de $\mathcal{H}$. D'autre part, la taille de l'arbre de description $\mathcal{G}_C$ d'une $\mathcal{ALE}$-description de concept C peut être exponentielle en fonction de la taille de C. Par conséquent, on obtient le résultat suivant :

Proposition 2.2.10. **[BKM99 (5)]** *La taille du lcs de deux $\mathcal{ALE}$-descriptions de concept C, D peut être exponentielle en fonction des tailles de C et de D.*

Approximation (approx). L'objectif de cette section est de présenter une inférence non-standard permettant de répondre à une tâche de la première instance du problème de transparence sémantique. Il s'agit d'approximer une L_1-description de concept par une autre L_2-description de concept. Brandt *et al.* **[BKT02a (15)]** ont proposé un algorithme pour le calcul de cette approximation où $L_1 = \mathcal{ALC}$ et $L_2 = \mathcal{ALE}$. Cet algorithme se base intimement sur le calcul de lcs.

Définition 2.2.11. Soit C une description de concept dans L_1 et D une description de concept dans un langage L_2 où L_1, L_2 sont des langages de logique de description. D est appelée L_2-approximation supérieure de C (en abrégé $D = approx_{L_2}(C)$) ssi

(1) $C \sqsubseteq D$

(2) Si $C \sqsubseteq D'$ et $D' \sqsubseteq D$, alors $D' \equiv D$ pour toute L_2-description de concept D'.

Quand L_2 est un sous-langage de L_1, alors $approx_{L_2}(C)$ est unique s'il existe. En effet, supposons que D' soit une L_2-description de concept qui vérifie la Définition 2.2.11. Puisque

$$C \sqsubseteq approx(C) \sqcap D' \sqsubseteq approx(C),$$

alors, par définition, on obtient : $approx(C) \sqcap D' \equiv approx(C) \equiv D'$.
C'est en particulier le cas quand L_2=$\mathcal{ALE}$ et L_1=$\mathcal{ALC}$.

Avant de calculer l'approximation d'une $\mathcal{ALC}$-description de concept C, cette dernière doit être normalisée. L'idée de cette normalisation est de transformer la $\mathcal{ALC}$-description de concept en une disjonction dont chaque disjonct est une conjonction. Cela permet de traduire la subsomption dont le subsumeur est une $\mathcal{ALE}$-description de concept C et le subsumé est une $\mathcal{ALC}$-description de concept D, en les subsomptions entre D et chaque disjonct au top-level de C' obtenue de C par la normalisation. La définition suivante fournit une définition formelle de la *forme normale* de la $\mathcal{ALC}$-description de concept.

Définition 2.2.12. Soit C une $\mathcal{ALC}$-description de concept.

(1) On désigne par $prim(C)$ l'ensemble de noms de concept qui apparaissent au top-level de C

(2) On désigne par $val(C)$ la conjonction de tous les C' qui apparaissent dans les restrictions universelles $\forall r.C'$ de C.

(3) On désigne par $ex(C)$ l'ensemble de tous les C' qui apparaissent dans les restrictions existentielles $\exists r.C'$ de C.

(4) La *forme normale* de C est définie comme suit :
Si $C \equiv \bot$ alors $C = \bot$; si $C \equiv \top$ alors $C = \top$; sinon, C est de la forme :
$C = C_1 \sqcup ... \sqcup C_n$
où $C_i = \sqcap_{A \in prim(C_i)} A \sqcap \bigsqcap_{C' \in ex(C_i)} \exists r.C' \sqcap \forall r.val(C_i)$, $\bot \sqsubset C_i$, et C', $val(C_i)$ sont sous la forme normale.

Notons que la normalisation d'une $\mathcal{ALC}$-description de concept C peut accroître la taille de C de façon exponentielle.

Grâce à la forme normale de la $\mathcal{ALC}$-description de concept, on peut caractériser la subsomption dont le subsumeur est une $\mathcal{ALE}$-description de concept et le subsumé est une $\mathcal{ALC}$-description de concept.

Théorème 2.2.13. [BKT02a (15)] *Soit D une $\mathcal{ALC}$-description de concept dans la forme normale spécifiée par la Définition 2.2.12, C une $\mathcal{ALE}$-description de concept. On a : $C \sqsubseteq D$ ssi :*

- *$C \equiv \bot$, ou*
- *$D \equiv \top$, ou*
- *Pour tout i , $1 \leq i \leq n$, les conditions suivantes doivent être simultanément satisfaites :*

  (1) $prim(D) \subseteq prim(C_i)$

  (2) *Pour chaque $D' \in ex(D)$, il existe un $C' \in ex(C_i)$ tel que $C' \sqcap val(C_i) \sqsubseteq D'$*

  (3) $val(C_i) \sqsubseteq val(D)$

Ce théorème établit un critère simple pour décider la subsomption dans un sous-ensemble des $\mathcal{ALC}$-descriptions de concept. Effectivement, ce théorème implique que la subsomption dont le subsumeur est une $\mathcal{ALE}$-description de concept et le subsumé est une $\mathcal{ALC}$-description de concept peut se traduire en la subsomption dans le langage $\mathcal{ALE}$. La conception et la correction de l'algorithme suivant résulte de ce théorème.

Algorithme 2.2.14. [BKT02a (15)]
Soit C une $\mathcal{ALC}$-description de concept dans la $\mathcal{ALC}$-forme normale :
$C = C_1 \sqcup ... \sqcup C_n$

(1) *Si $C \equiv \bot$ alors approx(C) := $\bot$;*

(2) *Si $C \equiv \top$ alors approx(C) := $\top$.*

(3) *Sinon, approx(C)*$\equiv$

$$\bigsqcap_{A \in \cap_{i=1}^{n} prim(C_i)} A \sqcap \bigsqcap_{(C'_1,...,C'_n) \in ex(C_1) \times ... \times ex(C_n)} \exists r.lcs\{approx(C'_i \sqcap val(C_i)) | \ 1 \leq i \leq n\} \sqcap$$
$$\forall r.lcs\{approx(val(C_i)) \mid 1 \leq i \leq n\}$$

Selon la remarque ci-dessus, la taille d'une $\mathcal{ALC}$-description de concept peut augmenter de façon exponentielle par la normalisation. En conséquence, le calcul de $approx(C)$ décrit dans l'Algorithme 2.2.14 peut engendrer un nombre double exponentiel des restrictions existentielles en fonction de la taille de C. Une des approches qui a pour objectif de diminuer la taille de $approx(C)$, sera étudiée dans le Chapitre 3.

Exemple 2.2.15. (Calcul d'Approx)

Supposons que dans une ontologie $\mathcal{T}_1$ basée sur le langage $\mathcal{ALC}$, les concepts Feu, Bruit, ProResAuPolluant soient définis comme suit :

Feu := Chaleur $\sqcap$ Polluant
Bruit := $\neg$Chaleur $\sqcap$ Polluant
ProResAuPolluant := Produit $\sqcap \exists resister.\top \sqcap (\forall resister.$Feu $\sqcup \forall resister.$Bruit$)$

D'après l'algorithme 2.2.14, la description de concept ProResAuPolluant peut être approximée par la $\mathcal{ALE}$-description de concept dans le langage suivante :

ProResAuPolluant$_{\mathcal{ALE}}$:= Produit $\sqcap \forall resister.$Polluant $\sqcap \exists resister.$Polluant

2.2.3. Langage CARIN. Une manière d'améliorer l'expressivité d'un système est de combiner plusieurs formalismes qui ne sont pas réciproquement exprimables. Pour ce faire, il est impératif de définir une sémantique commune pour le formalisme résultat en conservant les sémantiques de chaque formalisme constitutif. Cela permet d'assurer la consistance des comportements du système. Un formalisme résultant d'une combinaison de plusieurs formalismes dont la sémantique est définie de cette manière, est appelé *formalisme hybride*.

En sachant que le formalisme de règles de Horn et la Logique de Description sont orthogonaux, c'est-à-dire que l'un ne peut être exprimé par l'autre [**Bor94 (12)**], l'idée d'intégration du formalisme des règles à la Logique de Description pour augmenter l'expressivité du système a été réalisée dans un système basé sur le langage $\mathcal{AL}$-log [**DLN$^+$98b (25)**]. Le langage CARIN [**LR98 (53)**], qui est développé à l'Université de Paris Sud par *A. Y. Levy* et *M. C. Rousset* (1998), peut être considéré comme une extension du langage $\mathcal{AL}$-log en permettant aux descriptions de concepts et de *rôles* d'une Logique de Description d'apparaître en tant que prédicats dans les règles de Horn sans symbole de fonction.

Le langage CARIN se compose de trois composants. Le premier est une terminologie, basée sur la Logique de Description $\mathcal{ALCNR}$ ($\mathcal{R}$ signifie la permission de la conjonction de rôles primitifs), correspondant à un TBox. Le second est un ensemble de règles de Horn et le troisième est un ensemble de faits correspondant à un ABox étendu. Les concepts et rôles de la terminologie peuvent apparaître comme des prédicats dans les antécédents des règles de Horn et dans les faits. Les prédicats, qui n'apparaissent pas dans la terminologie, sont appelés *prédicats ordinaires*. Les prédicats ordinaires sont *n-aire*.

Composant de terminologie $\Delta_{\mathcal{T}}$. Le langage de la Logique de description le plus expressif de la famille CARIN est $\mathcal{ALCNR}$ où un rôle peut être défini comme une conjonction de rôles primitifs $P := R$ où P est un nom de rôle et R est une description de rôle. Une définition d'un concept est soit une déclaration de la forme suivante :

$A := C$ où A est un nom de concept et C est une description de concept,
soit une déclaration d'inclusion de la forme suivante :

$C \sqsubseteq D$ où C et D sont des descriptions de concept.

Dans CARIN, le nom d'un concept apparaît au plus une fois dans le côté gauche d'une définition de concept. La sémantique du composant est définie de façon normale *i.e.* elle se base sur une interprétation $(.^{\mathcal{I}}, \Delta)$ où $.^{\mathcal{I}}$ est une fonction d'interprétation et Δ est un ensemble non-vide, dit *domaine*.

Composant de Règles de Horn $\Delta_{\mathcal{R}}$. Le composant de règles de Horn, $\Delta_{\mathcal{H}}$, est un ensemble de règles sous la forme suivante :

$$p_1(\bar{X}_1) \wedge \ldots \wedge p_n(\bar{X}_n) \Rightarrow q(\bar{Y})$$

où $\bar{X}_i$ et $\bar{Y}$ sont des tuples de variables ou des constantes.
Une variable qui apparaît dans $\bar{Y}$ apparaît aussi dans $\bar{X}_1 \cup \ldots \cup \bar{X}_n$. Les prédicats $p_1, \ldots, p_n$ sont soit des noms de concept, soit des noms de rôle, soit des prédicats ordinaires qui n'apparaissent pas dans $\Delta_{\mathcal{T}}$. Le prédicat q doit être un prédicat ordinaire. Un prédicat ordinaire p dépend d'un prédicat ordinaire q si q apparaît dans l'antécédent d'une règle de Horn dont le conséquent est p. Un ensemble de règles est dit récursif, s'il existe un cycle de relation dépendante entre des prédicats ordinaires des règles.

Composant de faits $\Delta_{\mathcal{A}}$. Le composant des faits est un ensemble de faits sous la forme $p(\bar{a})$ où $\bar{a}$ est un tuple de constants et p est un concept, rôle ou prédicat ordinaire. Notons que le ABox correspondant au composant $\Delta_{\mathcal{T}}$ est inclus dans le composant des faits.

Sémantique de CARIN. La sémantique de CARIN est définie à partir de la sémantique de chaque composant *i.e.* une interprétation $\mathcal{I}$ est un modèle d'une base de connaissance basée sur le formalisme hybride $\Sigma = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ s'il est un modèle de chaque composant. Une interprétation $(.^{\mathcal{I}}, \Delta)$ affecte à chaque constante a dans la base Σ un objet $a^I \in \Delta^{\mathcal{I}}$, et à chaque prédicat ordinaire *n-aire* dans dans la base Σ, une relation *n-aire* sur le domaine $\Delta^{\mathcal{I}}$ non-vide.

Le modèle du composant de terminologie est normalement défini comme dans le Chapitre 1. Le modèle du composant de règles est défini comme suit :

Soient r une règle et une fonction α qui affecte à chaque variable de la règle r un objet dans le domaine $\Delta^{\mathcal{I}}$ et $\alpha(a) = a^{\mathcal{I}}$ pour tout $a \in \Delta$. Une interprétation $(.^{\mathcal{I}}, \Delta)$ avec la fonction α est un modèle de r si $\alpha(\bar{X}_i) \in p_i^I$ pour tout primitif de l'antécédent de r, alors $\alpha(\bar{Y}) \in q^{\mathcal{I}}$ où $q(\bar{Y})$ est le conséquent de r. Finalement, une interprétation $(.^{\mathcal{I}}, \Delta)$ est un modèle du fait $p(\bar{a})$ si $\bar{a}^{\mathcal{I}} \in p^{\mathcal{I}}$. Les modèles ci-dessus sont définis sous l'hypothèse des noms uniques qui stipule que si a et b sont des constantes dans la base Σ alors $a^I \neq b^I$.

Inférence dans CARIN. L'inférence générale de CARIN est la suivante : soit $\Sigma = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ une base de connaissances basée sur CARIN et $p(\bar{a})$ une requête primitive où p peut être un prédicat et $\bar{a}$ est un tuple de constantes, est-ce que $\Sigma \models p(\bar{a})$?

Dans un premier temps, on suppose que les règles dans $\Delta_{\mathcal{R}}$ soient non récursives. Avant de décrire l'algorithme pour l'inférence générale, nous présentons la technique des tableaux (Section 1.3.4) sur laquelle l'algorithme s'appuie. La technique des tableaux utilisée dans cet algorithme permet de calculer l'ensemble des *complétions* S à partir des assertions et des faits initiaux auxquels les *règles de propagations* s'appliquent. L'ensemble des assertions et des faits initiaux peut être considéré comme un ABox étendu initial (faits de prédicats ordinaires inclus) $\Delta_{\mathcal{A}} = \beta \cup \beta_r$ où β l'ensemble d'assertions de concepts et de rôles et β_r l'ensemble de faits de prédicats ordinaires dans la base Σ. Chaque complétion obtenue peut être considérée comme un ABox étendu complet.
On désigne par $\mathcal{V}$ l'ensemble des variables et des constantes dans la base Σ et par a, b les éléments de $\mathcal{V}$. De même, on désigne $\mathcal{W}$ l'ensemble des symboles de variables ajoutés pendant l'application des règles de propagation. On nomme u, v, x, y, z aux éléments de $\mathcal{W}$ et s, t aux éléments de $\mathcal{W} \cup \mathcal{V}$. Informellement, la construction de l'ensemble des complétions commence par un ABox initial S_β, construit à partir de $\beta \cup \Delta_{\mathcal{T}}$, qui représente tous les modèles de $\beta \cup \Delta_{\mathcal{T}}$. Ensuite, un ensemble des règles de propagation s'applique à S_β pour générer un ensemble de complétions. Chaque complétion est un ABox complété du ABox initial. Il y a certaines complétions qui contiennent des *clash* explicites (*i.e.* il existe un individu qui appartient à la fois à une classe et à son complément), et alors elles sont insatisfiables. Chaque complétion sans clash représente un sous-ensemble des modèles de $\beta \cup \Delta_{\mathcal{T}}$, ainsi elles fournissent une représentation finie de tous les modèles de $\beta \cup \Delta_{\mathcal{T}}$. Une propriété importante de ces complétions est de permettre de traduire la vérification que $p(\bar{a})$ est déduit d'une complétion sans clash en la vérification que $p(\bar{a})$ est satisfiable dans un *modèle canonique* de la complétion. Par conséquent, pour vérifier si $p(\bar{a})$ est déduit de $\beta \cup \Delta_{\mathcal{T}}$, il est suffisant de vérifier les modèles canoniques de chaque complétion sans clash.

Plus précisément, la construction de S comporte les étapes suivantes :

(1) Construire un ABox étendu initial S_β à partir de $\beta \cup \Delta_{\mathcal{T}}$ comme suit :
   (a) $S_\beta = \emptyset$
   (b) Si $C(a) \in \beta$, alors $S_\beta = S_\beta \cup C(a)$
   (c) Si $R(a, b) \in \beta$ et $R = P_1 \sqcap ... \sqcap P_m$, alors $S_\beta = S_\beta \cup \{P_1(a, b), ..., P_m(a, b)\}$
   (d) Si $C \sqsubseteq D \in \Delta_{\mathcal{T}}$, alors $S_\beta = S_\beta \cup \{\forall x : (\neg C \sqcup D)(x)\}$
   (e) Pour chaque couple d'individus $a, b \in \beta$, alors $S_\beta = S_\beta \cup \{a \neq b\}$
   (f) Pour chaque concept C qui apparaît dans $p(\bar{a})$ (la conclusion de l'inférence), alors $S_\beta = S_\beta \cup \{\forall x : (\neg C \sqcup C)(x)\}$

(2) Appliquer les règles de propagation suivantes au S_β pour obtenir un ensemble des complétions S. L'ensemble de S est initialisé par S_β. Pour chaque $s \in S$,

(a) La règle $\rightarrow_{\sqcap}$
Condition : s contient $(C_1 \sqcap C_2)(t)$, mais ne contient pas les deux $C_1(t)$ et $C_2(t)$.
Action : $s = s \cup \{C_1(t), C_2(t)\}$

(b) La règle $\rightarrow_{\sqcup}$
Condition : s contient $(C_1 \sqcup C_2)(t)$, mais ne contient ni $C_1(t)$ ni $C_2(t)$.
Action : $S = S \cup \{s_1, s_2\}$ où $s_1 = s \cup \{C_1(t)\}$, $s_2 = s \cup \{C_2(t)\}$

(c) La règle $\rightarrow_{\exists}$
Condition : s contient $\exists R.C(t)$ où $R = P_1 \sqcap ... \sqcap P_m$, mais ne contient aucun individu z tel que $C(z)$ et $R(t, z)$ se trouvent dans s. De plus, la variable t est bloquée (*cf.* [**LR98 (53)**])
Action : $s = s \cup \{C(y), P_1(t, y),...,P_m(t, y)\}$ où y est un individu qui n'apparaît pas dans s.

(d) La règle $\rightarrow_{\forall}$
Condition : s contient $\forall R.C(t)$ et $R(t, y)$, mais ne contient pas $C(y)$
Action : $s = s \cup \{C(y)\}$

(e) La règle $\rightarrow_{\geq}$
Condition : s contient $\geq nR(t)$ où $R = P_1 \sqcap ... \sqcap P_m$, mais il n'y a aucun individu $z_1,...,z_n$ tel que $R(t, z_i)$ $(1 \leq i \leq n)$ et $z_i \neq z_j$ $(1 \leq i < j \leq n)$, qui se trouvent dans s. De plus, la variable t est bloquée (*cf.* [**LR98 (53)**]).
Action : $s = s \cup \{(tPy_i),...,(tPy_m)\} \cup \{y_i \neq y_j \mid 1 \leq i < j \leq n\}$

(f) La règle $\rightarrow_{\leq}$
Condition : s contient $\leq nR(t)$ et contient les assertions xRy_i $1 \leq i \leq n+1$ et $y_i \neq y_j$ ne se trouve pas dans s pour un certain i, j , $1 \leq i < j \leq n+1$.
Action : $S = S \cup \{s_{i,j}\}$ où $s_{i,j}$ est construit comme suit. Pour chaque couple y_i, y_j tels que $1 \leq i < j \leq n+1$ et $y_i \neq y_j$ ne se trouve pas dan s, le ABox $s_{i,j}$ est obtenu de s en remplaçant chaque occurrence de y_i par y_j.

(g) La règle $\rightarrow_{\forall x}$
Condition : s contient $\forall t.C(t)$ et le variable y, mais ne contient pas $C(y)$
Action : $s = s \cup \{C(y)\}$

(3) Pour chaque complétion $s \in S$ sans *clash*, construire une *interprétation canonique* I_S. Une complétion contient un *clash* si elle contient :

(a) $\{\bot(s)\}$ ou

(b) $\{P(s), \neg P(s)\}$ ou

(c) $\{\leq nR(s)\} \cup \{sP_1t_i,...sP_mt_i \mid i \in \{1..n+1\}, R = P_1 \sqcap ... \sqcap P_m\} \cup \{t_i \neq t_j \mid 1 \leq i < j \leq n+1\}$

L'interprétation canonique I_s où s est une complétion sans clash est définie comme suit :

$\Delta^{\mathcal{I}_s} := \{\ t \mid t$ est un objet dans $s\}$; $\alpha^{\mathcal{I}_s}(t) = t$; $t \in A^{\mathcal{I}_s}$ ssi $(A(t) \in s$ pour chaque concept primitif A ; $(x, y) \in R^{\mathcal{I}_s}$ ssi $xRy \in S$ ou x est bloqué (*cf.* [**LR98 (53)**]).

Dans le pire des cas, la complexité de la construction de l'ensemble des complétions est double exponentielle en fonction de la taille de $\beta \cup \Delta_{\mathcal{T}}$ car d'une part le nombre maximal d'objets dans une complétion de S_β est double exponentiel en la taille de $\beta \cup \Delta_{\mathcal{T}}$, d'autre part la construction d'une complétion prend également un temps double exponentiel en la taille de $\beta \cup \Delta_{\mathcal{T}}$ [**LR98 (53)**].

En bref, l'algorithme pour l'inférence générale $\Sigma \models p(\bar{a})$ peut être exécuté en deux étapes (les détails de cet algorithme sont décrits dans [**LR98 (53)**])

(1) D'abord, on construit l'ensemble des assertions et des faits S_β. Ensuite, les règles de propagation s'appliquent à S_β pour obtenir l'ensemble des complétions S.

(2) Pour chaque complétion $s \in S$, et donc chaque interprétation canonique $\mathcal{I}_S$, on calcule les extensions des prédicats ordinaires en évaluant les règles de Horn. Cette tâche peut être effectuée par un algorithme d'inférence traditionnel (par exemple, inférence en arrière). Notons que l'on vérifie les assertions de concept $a^{\mathcal{I}_S} \in C^{\mathcal{I}_S}$ au lieu de vérifier $S \models C(a)$ et les assertions de rôle $(aRb)^{\mathcal{I}_S} \in R^{\mathcal{I}_S}$ au lieu de vérifier $S \models (aRb)$. Par contre, on utilise β_r pour vérifier les prédicats ordinaires.

Par conséquent, la requête $p(\bar{a})$ est déduite de la base $\Sigma = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ ssi $\bar{a}$ se trouve dans l'extension du prédicat p pour chaque complétion $s \in S$ sans clash. Le théorème suivant établit la correction de l'algorithme présenté :

Théorème 2.2.16. [**LR98 (53)**] *L'inférence $\beta \cup \Delta_{\mathcal{T}} \models p(\bar{a})$ est vérifiée si et seulement si $p(\bar{a})$ est satisfiable dans l'interprétation canonique de chaque complétion sans clash de S_β.*

Le théorème suivant est un résultat direct de l'algorithme.

Théorème 2.2.17. [**LR98 (53)**] *Soit $\Sigma = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ une base de connaissance basée sur le langage CARIN $\mathcal{ALCNR}$ dont les règles de Horn ne sont pas récursives. Soit $p(\bar{a})$ un prédicat primitif où p est un concept, rôle ou prédicat ordinaire. Le problème déterminant $\Sigma \models p(\bar{a})$ est décidable.*

Maintenant, on suppose que les règles dans $\Delta_{\mathcal{R}}$ soient récursives. Les travaux dans [**LR98 (53), LR96 (54)**] ont également traité le problème de l'inférence générale de CARIN dans le cas où les règles de Horn sont récursives. Un résultat important de ces travaux est de démontrer que si les règles de Horn sont récursives, ce problème est en général indécidable. Et pourtant les auteurs ont également identifié un sous-langage de $\mathcal{ALCNR}$ dans lequel le problème de l'inférence reste encore décidable.

2.3. Discussion et conclusion

Le langage $\mathcal{DLR}$ est une Logique de Description expressive capable de capturer la sémantique du diagramme de classe UML qui est le langage de représentation des composants de base dans l'ebXML. La représentation des Composants de Base de l'ebXML en $\mathcal{DLR}$ permet d'utiliser les inférences bien étudiées dans la Logique de Description pour détecter l'équivalence et la subsomption entre des Composants de Base. Cela peut éliminer les redondances, et donc diminuer la taille de l'ensemble des Composants de Base. Par ailleurs, le raisonnement dans $\mathcal{DLR}$ est en général EXPTIME-difficile [**BCD03 (9)**] et la formalisation des Composants de Base ne mobilise pas toute la puissance de l'expressivité du diagramme de classe UML. La question méritant d'être étudiée est la suivante : quel est le sous-langage de $\mathcal{DLR}$ tel qu'il est suffisamment expressif pour capturer les Composants de Base de l'ebXML ? La source de la complexité EXPTIME-difficile vient de la contrainte d'identification dans le langage $\mathcal{DLR}$ correspondant à l'association *n-aire* ($n \geq 2$) dans le diagramme de classe UML. Ainsi, l'aspect de la complexité devrait être pris en compte lors de la conception de ces Composants de Base pour que l'utilisation de l'association *n-aire* soit la plus limitée possible.

Quant aux inférences non-standards étudiées dans ce chapitre, l'approche structurelle se montre très utile. Même si le calcul lcs devient trivial pour les langages contenant la disjonction, le calcul d'approximation se montre toujours nécessaire dans le cas où la transformation d'une L_1-description de concept en une L_2-description de concept serait exigée. D'autre part, certaines nouvelles inférences non-standards, comme par exemple OUBLIER qui sera présentée dans le Chapitre 4, se base sur l'approche structurelle. C'est pourquoi la recherche portant sur une caractérisation de subsomption structurelle pour les langages expressifs (avec le constructeur de disjonction) est toujours motivée.

Par ailleurs, la complexité double exponentielle de l'algorithme pour l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$ est vraiment décourageante. Toutefois, cette complexité qui résulte de la considération dans le pire des cas, sera améliorée à l'aide d'une représentation particulière pour les descriptions de concept. Le chapitre 3 de ce mémoire sera consacré à cette étude.

En dehors de l'ajout d'un constructeur à une Logique de Description, le langage CARIN montre une autre manière d'augmenter l'expressivité d'une Logique de Description. Dans une base de connaissance basée sur ce langage, on a besoin de développer les inférences non seulement sur le TBox mais aussi sur le ABox afin de répondre à une requête, dite *requête conjonctive*. Pour de telles inférences, la technique des tableaux paraît avantageuse.

Cependant, le formalisme des règles de Horn dans le langage CARIN est loin d'être les règles de contexte qui sont décrites dans la Section 2.1.3. La première différence est que l'application des règles de Horn ne modifie ni TBox ni ABox car le conséquent de ces règles n'est qu'un prédicat ordinaire, alors que l'application des règles de contexte peut changer du TBox (partie intentionnelle de la base) et du ABox (partie extensionnelle de la base). De plus, contrairement aux règles de Horn non-récursives où la terminaison de l'application de ces règles et l'unicité de l'état final

de la base de connaissances sont automatiquement assurée, le conséquent des règles de contexte en tant que les *opérations de révision* doivent subir certaines restrictions. Ces problèmes cités sont l'objet du Chapitre 4 de ce mémoire.

Deuxième partie

Inférences non-standard et Règles de Révision

CHAPITRE 3

Inférences non-standard pour Transformation de Terminologies

3.1. Introduction

Comme présenté dans la Section 2.1.3, la Logique de Description peut être utilisée pour formaliser et concevoir les ontologies d'un domaine d'application. Afin de réduire la taille des ontologies, celles-ci sont organisées comme une hiérarchie dont la racine est une ontologie de base partagée et les feuilles sont des ontologies dérivées de l'ontologie de base. Chaque ontologie dérivée correspond à un sous-domaine d'application qui spécifie les activités d'un acteur. Les ontologies dérivées partagent les concepts, les rôles primitifs et les concepts de base introduits dans l'ontologie de base. Si toutes les ontologies du système utilisent le même langage L.D pour exprimer les descriptions de concept, alors cette organisation des ontologies supporte sans difficulté majeure l'échange de documents de façon transparente sémantique entre les acteurs. C'est-à-dire qu'un document composé des concepts d'une ontologie dérivée est interprétable par un autre acteur qui utilise une autre ontologie dérivée. Cependant, dans certains cas, les expressivités de Logiques de Description utilisées dans les ontologies dérivées sont différentes *i.e.* les langages L.D. utilisés sont différents. C'est pourquoi le problème qui se pose est la traduction d'une description concept d'une ontologie exprimée par un langage D.L L_1 en une description de concept d'une autre ontologie exprimée par un langage D.L L_2 qui est moins expressif que L_1. Ce problème est connu comme l'approximation d'une Logique Description L_1 par une Logique de Description L_2, c'est-à-dire le calcul du subsumeur minimal en L_2 d'une description de concept en L_1. Un algorithme pour ce calcul a été présenté dans la Section 2.2.2. Cet algorithme permet de calculer l'approximation où L_1= $\mathcal{ALC}$ et L_2= $\mathcal{ALE}$.
La complexité de l'Algorithme *2.2.14* vient des deux sources suivantes. Premièrement, la normalisation d'une $\mathcal{ALC}$-description de concept selon la Définition 2.2.12 peut générer une description de concept dont la taille est exponentielle en fonction de la taille de la description de concept d'entrée [**BKT02a (15)**]. Deuxièmement, l'interaction entre la restriction universelle et existentielle via la conjonction, c'est-à-dire la normalisation d'une $\mathcal{ALE}$-description de concept par les règles dans la Définition 2.2.4, engendre également une description de concept dont la taille est exponentielle en fonction de la taille de la description de concept d'entrée [**BKM99 (5)**]. La complexité résultant de l'interaction de ces deux sources peut être doublement exponentielle en fonction de la taille d'entrée. Par un exemple présenté à la fin de ce chapitre, nous allons montrer que dans le pire des cas la taille d'une description

de concept obtenu du calcul d'approximation peut s'élever à un nombre doublement exponentiel en taille de la $\mathcal{ALC}$ -description de concept d'entrée.
Cependant, si les descriptions de concept sont transformées en arbre de description selon la Définition 2.2.1, on pourrait obtenir une représentation compacte pour les descriptions de concept résultant du calcul du Plus Spécifique Subsumeur Commun (*lcs*) (Définition 2.2.6) dans la Logique de Description $\mathcal{ALE}$. Comme montré dans [**BT02 (4)**], la taille exponentielle de *lcs* est inévitable si l'on utilise la représentation ordinaire pour exprimer les descriptions de concept. En effet, la taille de descriptions de concept peut accroire de façon exponentielle à cause de la normalisation par les règles dans la Définition 2.2.4 avant le calcul du *lcs*. Cela nous permet d'espérer que s'il y a une représentation compacte des descriptions de concept dans laquelle l'accroissement exponentiel causé par la normalisation, est réduit, alors la complexité du calcul de *lcs* peut être améliorée par un algorithme qui nécessite seulement une espace polynômiale. Cela sera complété par une autre procédure de calcul en espace polynômiale qui permet de décider la subsomption basée directement sur cette représentation compacte des descriptions de concept.

Ce chapitre commence par présenter une représentation compacte pour les $\mathcal{ALE}$-description de concept. Cette représentation résulte d'une méthode de normalisation pour $\mathcal{ALE}$-descriptions de concept dans laquelle l'interaction entre les restrictions existentielles et universelles est compressée. L'idée de cette compression réside en introduction de ϵ-arcs dans le $\mathcal{ALE}$-arbre de description. L'arbre obtenu, qui est appelé ϵ-arbre de description, nous permet de traduire la duplication d'une partie d'un arbre de description en l'ajout d'un ϵ-arc. Cette duplication est générée par l'application des règles de normalisation dans la Définition 2.2.4. Par la suite, nous allons montrer que la représentation compacte pour le *lcs* peut être obtenue du calcul de produit entre les ϵ-arbres de description. Un algorithme en espace polynômiale pour le calcul de *lcs* sera également présenté.

Le deuxième élément constitutif de ce chapitre est d'introduire un exemple qui montre malheureusement qu'un algorithme exponentiel pour le calcul de l'approximation d'une $\mathcal{ALC}$-description de concept par une $\mathcal{ALE}$-description de concept n'existe pas si la représentation des descriptions de concept ordinaire est utilisée. Cependant, la complexité du calcul d'approximation est améliorée naturellement par la nouvelle méthode de calcul *lcs*.

3.2. ϵ-arbre de description pour $\mathcal{ALE}$-description de concept

Comme mentionné dans [**BT02 (4)**], l'application de la règle de normalisation 2. (comme spécifiée dans la Définition 2.2.4) à une $\mathcal{ALE}$-description de concept peut accroire exponentiellement la taille de celle-ci. Dans cette section, nous proposons une structure de donnée spécifique pour les $\mathcal{ALE}$-arbres de description en y ajoutant d'arcs vides, nommé ϵ-arc, qui correspondent au rôle d'identité. L'introduction de ϵ-arcs permet d'éviter la duplication d'une partie d'un arbre de description. Ainsi, une application de la règle de normalisation 2. peut être interprétée comme un ajout de ϵ-arc.

Nous supposons que toute $\mathcal{ALE}$-description de concept considérée dans cette section est normalisée par toutes les règles de normalisation décrites dans la Définition 2.2.4 à l'exception de la règle 2. De plus, pour des raisons de simplicité, supposons que l'ensemble de rôles comporte uniquement un élément *i.e* $N_R = \{r\}$. Tous les résultats de ce chapitre peuvent s'appliquer à un ensemble de rôles arbitraire N_R.

3.2.1. ϵ-arbre de description. On désigne par $\mathcal{G}^{\epsilon}_C = (V_G, E_G \cup E^{\epsilon}_G, v_G, l_G)$ le ϵ-arbre de description du concept C où C est normalisé par les règles de normalisation à l'exception de la règle 2., E^{ϵ}_G est l'ensemble des ϵ-arcs que nous allons définir dans cette section. Normalement, on désigne également par $v^k \in V_G$ un nœud au niveau k de l'arbre $\mathcal{G}^{\epsilon}_C$. Néanmoins, pour simplifier l'écriture, on désigne par u_i et par w_i les nœuds aux niveaux $(k-1)$ et k respectivement.

Définition 3.2.1. (voisinage)
Soit C une $\mathcal{ALE}$-description de concept et son arbre de description $\mathcal{G}'_C = (V_{G'}, E_{G'}, v^0_{G'}, l_{G'})$ défini dans la Définition 2.2.1 où C est normalisé par les règles de normalisation à l'exception de la règle 2.

(1) Au niveau 0 de $\mathcal{G}'_C$, on désigne par $N^0 = \{v^0_{G'}\}$ le voisinage unique.

(2) Pour chaque niveau $k > 0$ de $\mathcal{G}'_C$, l'ensemble des voisinages au niveau k, N^k, est récursivement défini comme suit :
Pour chaque $n^{k-1} \in N^{k-1}$ où $n^{k-1} = \{u_1, ..., u_m\}$, on obtient :

   (a) $\{w_1, ..., w_m\} \in N^k$ s'il existe les arcs : $u_1 \forall r w_1, ..., u_m \forall r w_m$

   (b) $\{w_1, ..., w_m, w_{m+1}\} \in N^k$ s'il existe les arcs $u_1 \forall r w_1, ..., u_m \forall r w_m$ et $u_i r w_{m+1}$, $u_i \in n^{k-1}$

Remarque 3.2.2. A partir de la Définition 3.2.1, on peut obtenir plusieurs voisinages $n^k_i, n^k_j \in N^k$ qui contiennent le même ensemble de nœuds. Et pourtant, si n^k_i, n^k_j sont construits d'un $(k-1)$-voisinage n^{k-1}, alors $n^k_i \neq n^k_j$.

Définition 3.2.3. (ϵ-normalisation 1)
Soit C une $\mathcal{ALE}$-description de concept et son arbre de description $\mathcal{G}'_C = (V_{G'}, E_{G'}, v^0_{G'}, l_{G'})$ défini dans la Définition 2.2.1 où C est normalisé par les règles de normalisation à l'exception de la règle 2. Pour chaque niveau $k \geq 0$ de l'arbre $\mathcal{G}'_C$, pour chaque $n^k \in N^k$ et pour chaque couple des nœuds $w_i, w_j \in n^k$:

(1) S'il existe les arcs $u_i \forall r w_i, u_j \forall r w_j$, $u_i \epsilon u_j$ et il n'existe pas de ϵ-arc $(w_j \epsilon w_i)$, le ϵ-arc $(w_i \epsilon w_j)$ est ajouté.

(2) S'il existe les arcs $u_i r w_i, u_j \forall r w_j$, le ϵ-arc $(w_i \epsilon w_j)$ est ajouté.

La Définition 3.2.3 fournit une procédure pour la ϵ-normalisation. Cette définition exige la détermination des voisinages. Cela prend au plus un temps exponentiel en fonction de la taille de C. Une définition équivalente - mais moins complexe - pour ϵ-arbre de description est proposée comme suit :

Définition 3.2.4. (ϵ-normalisation 2)
Soit C une $\mathcal{ALE}$-description de concept et son arbre de description
${\mathcal{G}'}_C = (V_{G'}, E_{G'}, v^0_{G'}, l_{G'})$ défini dans la Définition 2.2.1 où C est normalisé par les règles de normalisation à l'exception de la règle 2. .

(1) Pour chaque nœud $v \in V_{G'}$, s'il existe les arcs vrv'' et $v\forall rv'$, un ϵ-arc $(v''\epsilon v)$ est ajouté.

(2) Pour chaque niveau $k > 0$ de l'arbre ${\mathcal{G}'}_C$, pour chaque couple des nœuds $w_i, w_j \in n^k$ à ce niveau tels qu'il existe les arcs $u_i\forall rw_i$, $u_j\forall rw_j$ et $u_i\epsilon u_j$. S'il existe le ϵ-arc $(u_i\epsilon u_j)$ et il n'existe pas de ϵ-arc $(w_j\epsilon w_i)$, alors le ϵ-arc $(w_i\epsilon w_j)$ est ajouté .

(3) Pour chaque niveau $k > 0$ de l'arbre ${\mathcal{G}'}_C$, pour chaque couple des nœuds $w_i, w_j \in n^k$ à ce niveau tels qu'il existe les arcs u_irw_i, $u_j\forall rw_j$. S'il existe le ϵ-arc $(u_i\epsilon u_j)$ ou $(u_j\epsilon u_i)$, alors le ϵ-arc $(w_i\epsilon w_j)$ est ajouté.

La Définition 3.2.4 fournit une procédure polynômiale en fonction de la taille de C pour la ϵ-normalisation. Cependant, la correction de cette procédure sera assurée si l'équivalence entre les deux définitions est établie.

Proposition 3.2.5. *Les règles de ϵ-normalisation introduites dans la Définition 3.2.3 sont équivalentes à celles introduites dans* 3.2.4.

Démonstration. Nous montrons que les ensembles des ϵ-arcs ajoutés à l'arbre de description par les deux définitions sont identiques. Ce n'est pas difficile de vérifier que les deux ensembles des ϵ-arcs sont identiques au niveau k=1. Au niveau $k > 1$, soit $w_1\epsilon w_2$ un ϵ-arc ajouté à l'arbre de description par Définition 3.2.3. Donc, w_1, w_2 doivent appartenir à un k-voisinage. Supposons qu'il existe u_1, u_2 tels que $u_1\forall rw_1$, $u_2\forall rw_2$ ou u_1rw_1, $u_2\forall rw_2$. Selon la Définition 3.2.1, il existe un (k-1)-voisinage $n^{k-1} \in N^{k-1}$ tel que $u_1, u_2 \in n^{k-1}$ et $u_1\epsilon u_2$ ou $u_2\epsilon u_1$ ajoutés par la Définition 3.2.3 (u_1, u_2 peuvent être identiques). Par hypothèse de récurrence, soit $u_1 \equiv u_2$, ou $u_1\epsilon u_2$ ou $u_2\epsilon u_1$ est ajouté par la Définition *3.2.4*. Donc, $w_1\epsilon w_2$ est également ajouté par la Définition *3.2.4*.
Inversement, soit $w_1\epsilon w_2$ un ϵ-arc ajouté à l'arbre de description par la Définition *3.2.4*. Supposons qu'il existe u_1, u_2 tels que $u_1\forall rw_1$, $u_2\forall rw_2$ ou u_1rw_1, $u_2\forall rw_2$ (u_1, u_2 peuvent être identiques). On a : $u_1\epsilon u_2$ ou $u_2\epsilon u_1$ ajouté par la Définition *3.2.4*. Par hypothèse de récurrence, $u_1\epsilon u_2$ ou $u_2\epsilon u_1$ sont ajoutés par la Définition 3.2.3. Donc, il existe un (k-1)-voisinage $n^{k-1} \in N^{k-1}$ tel que $u_1, u_2 \in n^{k-1}$. D'après la Définition 3.2.1, w_1, w_2 appartiennent à un k-voisinage. Par conséquent, $w_1\epsilon w_2$ est également ajouté par la Définition 3.2.3. □

Remarque 3.2.6. La ϵ-normalisation selon la Définition *3.2.4* ne modifie pas le nombre des nœuds de l'arbre. De plus, l'ajout des ϵ-arcs prend un temps polynômial en fonction du nombre des nœuds de l'arbre.

La définition du ϵ- arbre de description simple résulte directement de la ϵ-normalisation.

Définition 3.2.7. (ϵ-arbre de description simple)

Un ϵ-arbre de description simple est généré par la Définition *3.2.4 i.e* un $\mathcal{ALE}$-arbre de description obtenu en y ajoutant les ϵ-arcs selon les règles dans la Définition *3.2.4.*

Maintenant, nous montrons que la ϵ-normalisation peut remplacer l'application de la règle de normalisation 2. C'est-à-dire que nous devons fournir une procédure qui transforme un ϵ-arbre de description $\mathcal{G}_C^\epsilon$ en l'arbre de description $\mathcal{G}_C$ où la description de concept C est normalisée par toutes les règles dans la Définition 2.2.4. Notons qu'une application de la règle de normalisation 2. peut amener à de nouvelles applications de la règle de normalisation 1. Pour cette raison, la procédure de la transformation doit prendre en compte l'application de la règle 1. générée par l'application de la règle 2. Cela est impliqué dans la notion du voisinage. Les autres règles de normalisation sont également appliquées lorsque l'on calcule les étiquettes des nœuds de l'arbre résultat.

Algorithme 3.2.8.
Entrée *: ϵ-arbre de description $\mathcal{G}_C^\epsilon = (V_G, E_G \cup E_G^\epsilon, v_0, l_G)$.*
Sortie *: $\mathcal{ALE}$-arbre de description $\mathbf{B}(\mathcal{G}_C^\epsilon)$.*

(1) *Au niveau 0, un voisinage unique 0-voisinage $\{v_0\}$, qui est la racine de $\mathbf{B}(\mathcal{G}_C^\epsilon)$, est créé et $l(\{v_0\}) = l(v_0)$.*

(2) *Pour chaque $(k-1)$-voisinage $n^{k-1} = \{v_1, ..., v_m\}$ de $\mathcal{G}_C^\epsilon$ correspondant au $(k-1)$-nœud de $\mathcal{G}$, les k-nœuds et les arcs de $\mathbf{B}(\mathcal{G}_C^\epsilon)$ sont créés à partir un k-voisinage du $\mathcal{G}_C^\epsilon$ comme suit :*

   (a) *un k-voisinage $n^k = \{v'_j \mid$ pour tout arc $(v_i \forall r v'_j)$ où $v_i \in n^{k-1}\}$ et un $\forall r$-arc correspondant $n^{k-1} \forall r n^k$.*
   Si $\bot \in \bigcup_{v_i \in n^{k-1}}^m \{l(v_i)\}$ ou il existe $P \in N_C$ tel que $P, \neg P \in \bigcup_{v_i \in n^{k-1}}^m \{l(v_i)\}$, alors $l(n^k) := \{\bot\}$. Sinon,
   $l(n^k) = \bigcup_{v'_j \in n^k}^m \{l(v'_j)\}$.

   (b) *les k-voisinages $m_j^k = n^k \cup \{v'_j\}$ pour tout arc $(v_i r v'_j)$ où $v_i \in n^{k-1}$, n^k est déterminé dans l'étape (a), et les r-arcs correspondants : $n^{k-1} r m_j^k$*
   Si $\bot \in \bigcup_{v_i \in n^{k-1}}^m \{l(v_i)\}$ ou il existe $P \in N_C$ tel que $P, \neg P \in \bigcup_{v_i \in n^{k-1}}^m \{l(v_i)\}$, alors $l(m_j^k) := \{\bot\}$. Sinon,
   $l(m_j^k) = \bigcup_{v'_j \in m_j^k}^m \{l(v'_j)\}$.

(3) *Appliquer les règles de normalisation 4. 6. et 7. à l'arbre $\mathbf{B}(\mathcal{G}_C^\epsilon)$ obtenu dans l'étape 2. :*

   (a) *$l(v) \to l(v) \setminus \{\top\}$ où $v \in V_{B(G_C^\epsilon)}$ (règle 4)*

   (b) *$v' r v \to v$ si $l(v) = \{\bot\}$, v est une feuille et $v' r v \in E_{B(G_C^\epsilon)}$ (règle 6)*

   (c) *$\mathcal{G}'(v) \to \bot$ où $\bot \in l(v)$ et $\mathcal{G}'(v)$ est un sous-arbre de $\mathbf{B}(\mathcal{G}_C^\epsilon)$ (règle 7)*

Note 3.2.9. Il n'est pas nécessaire d'appliquer la règle 3. dans l'algorithme car cette règle n'a besoin que d'être appliquée une seule fois.

Exemple 3.2.10. (ϵ-normalisation)
Soit $C := \exists r.(\forall r.\forall r.P_3^0) \sqcap \exists r.(\forall r.\forall r.P_3^1) \sqcap$
$\forall r.(\exists r.\forall r.P_2^0 \sqcap \exists r.\forall r.P_2^1 \sqcap \forall r.(\exists r.P_1^0 \sqcap \exists r.P_1^1))$
La Figure 3.2.1 illustre la représentation graphique de l'Exemple 3.2.10.

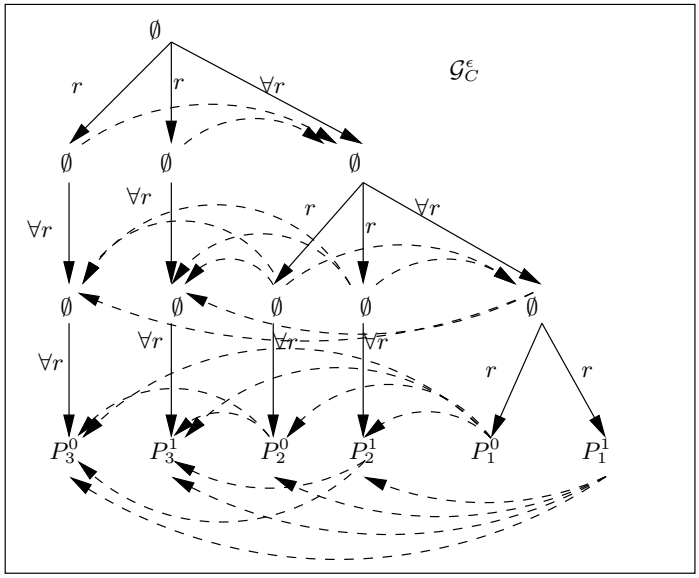

FIG. 3.2.1. ϵ-arbre de description

Nous avons besoin de la proposition suivante pour établir l'équivalence entre l'arbre de description obtenu de la normalisation par les règles dans la Définition 2.2.4 et l'arbre obtenu de l'Algorithme *3.2.8.*

Proposition 3.2.11. *Soit C une $\mathcal{ALE}$-description de concept et $\mathcal{G}_C^\epsilon$ son ϵ-arbre de description. L'arbre $\boldsymbol{B}(\mathcal{G}_C^\epsilon)$ est identique à l'arbre de description $\mathcal{G}_C$ obtenu de la normalisation par les règles dans la Définition 2.2.4.*

DÉMONSTRATION. On désigne par $|\mathcal{G}|$ la profondeur de l'arbre $\mathcal{G}_C^\epsilon$. Rappelons que toutes les règles de normalisation à l'exception de la règle 2 doivent être appliquées à la description de concept C avant la ϵ-normalisation. De plus, on n'a besoin qu'une fois d'appliquer les règles 1. et 2. dans la normalisation d'une $\mathcal{ALE}$-description de concept. Donc, la proposition sera montrée si nous démontrons que l'arbre obtenu de l'application des règles 1, 2, 5 suivant l'application de toutes les règles à l'exception de la règle 2. est identique à l'arbre obtenu de l'Algorithme *3.2.8* sans l'étape 3.
Nous le montrons par récurrence sur le niveau k de $\mathcal{G}_C$. Notons que $|\mathcal{G}_C| = |\mathcal{G}_C^\epsilon|$.
Niveau $k = 0$.
Puisque $\mathcal{G}_C^\epsilon$ n'a qu'un 0-voisinage $\{v_0\}$ et l'arbre de description $\mathcal{G}_C$ a seulement un nœud v_0 au niveau 0, alors le nœud unique au niveau 0 de $\mathbf{B}(\mathcal{G}_C^\epsilon)$ et de $\mathcal{G}_C$ possède l'étiquette $l(v_0)$.
Niveau $k > 0$.

On désigne par $\{u_1, ..., u_m\}$ chaque nœud au niveau k de l'arbre de description $\mathcal{G}_C$ s'il y a une unification des sous-arbres, correspondant à la règle 1., ou une copie des sous-arbres correspondant à la règle 2., et les racines de ces sous-arbres sont les nœuds $u_1, ..., u_m$. De plus, on a : $l\{u_1, ..., u_m\} = \{l(u_1) \cup ... \cup l(u_m)\}$. En effet, chaque unification de deux sous-arbres $(u_i \forall r\mathcal{G}(w_p))$ et $(u_i \forall r\mathcal{G}(w_q))$ effectuée par la règle 1 crée le sous-arbre $(u_i \forall r\mathcal{G}(\{w_p, w_q\}))$. De même, chaque copie du sous-arbre $(u_i \forall r\mathcal{G}(w_p))$ au sous-arbre $(u_i r w_q)$ effectuée par la règle 2 crée $(u_i r\mathcal{G}(\{w_p, w_q\})$. Plusieurs occurrences d'un nœud u_i dans différents nœuds de l'arbre de description normalisé résultent de cette copie des sous-arbres.
Soit $\{u'_1, ..., u'_n\}$ k-voisinage de l'arbre $\mathcal{G}$ qui est créé par l'opérateur **B** correspond à un k- nœud $\{u_1, ..., u_m\}$ de l'arbre de description. Par hypothèse de récurrence, on a : $m = n$ et $u_1 \equiv u'_1$, ..., $u_m \equiv u'_m$.
Supposons que $\{v_1, ..., v_m\} \neq \perp$. Dans ce cas, par l'étape 2.a de l'Algorithme *3.2.8,* un $\forall r$-arc et un $(k+1)$-voisinage sont créés et cet $\forall r$-arc correspond exactement au $\forall r$-arc et son nœud de destination créé par l'application de la règle de normalisation 1. De la même manière, par l'étape 2.b de l'Algorithme *3.2.8,* les r-arcs et $(k+1)$-voisinages sont créés et ces arcs correspondent exactement aux r-arcs et leur nœuds de destination créés par la règle de normalisation 2. Donc, les étiquettes des $(k+1)$-voisinages obtenus de l'Algorithme *3.2.8* sont équivalentes aux étiquettes des nœuds de destination obtenus de la normalisation.
Supposons que $\{v_1, ..., v_m\} = \perp$. Dans ce cas, il n'existe pas d'arc sortant du nœud $\{v_1, ..., v_m\}$ de l'arbre de description $\mathcal{G}_C$; et il n'existe pas d'arc sortant du nœud $\{u'_1, ..., u'_n\}$ de l'arbre obtenu de l'Algorithme *3.2.8.*
Par conséquent, l'arbre $\mathbf{B}(\mathcal{G}^{\epsilon}_C)$ et l'arbre de description $\mathcal{G}_C$ coïncident à tout niveau $k \leq |\mathcal{G}|$. □

Remarque 3.2.12. (complexité de l'Algorithme 3.2.8)
L'Algorithme 3.2.8 prend un temps exponentiel et un espace polynômiale en fonction de la taille de la description de concept d'entrée. En effet, cet algorithme exige au plus une mémoire pour stocker une serie des voisinages n^1,...,n^k où k est la profondeur de l'arbre résultat. Cette serie des voisinages correspond à un chemin de la racine vers une feuille de l'arbre résultat. De plus, avec ce chemin, les voisinages de chaque nœud au long du chemin sont également stockés, et le nombre des voisinages de chaque nœud est polynômial en fonction de k. Par conséquent, l'Algorithme 3.2.8 nécessite une espace polynômiale en fonction de k.

3.3. Produit de ϵ-arbres de description

3.3.1. Produit de ϵ-arbres de description. Pour simplifier la définition du produit de deux ϵ-arbres de description et faciliter le calcul, on a besoin d'ajouter aux ϵ-arbres de description quelques informations supplémentaires. En effet, la sémantique d'un ϵ-arbre de description n'est pas changée si l'on y ajoute des ϵ-arcs, appelé ϵ-cycle, dont les deux extrémités sont identiques. Les ϵ-cycles ajoutés permettent à chaque ϵ-arc du ϵ-arbre de description de réapparaître dans le produit. En outre, on ajoute également à chaque nœud une valeur, appelé *drapeau*, initialisée par son nœud

prédécesseur. On désigne par $f(v)$ la valeur de drapeau du nœud v. Le drapeau d'un nœud v nous permet de déterminer le prédécesseur de v dans le cas où il n'existerait pas d'arc arrivant normal. (un arc normal n'est pas un ϵ-arc).

Définition 3.3.1. Soient C ,D des $\mathcal{ALE}$-descriptions de concept et $\mathcal{G}^{\epsilon}_{C}$= $(V_G, E_G \cup E^{\epsilon}_{G}, v_0, l_G)$, $\mathcal{H}^{\epsilon}_{D} = (V_H, E_H \cup E^{\epsilon}_{H}, w_0, l_H)$ les ϵ-arbres de description avec les ϵ-cycles, de C et de D respectivement.

- Si $l_G = \{\bot\}$ ($l_H = \{\bot\}$) alors on définit $\mathcal{G}^{\epsilon} \times \mathcal{H}^{\epsilon}$ par remplaçant chaque nœud w (v) dans $\mathcal{H}^{\epsilon}$ ($\mathcal{G}^{\epsilon}$) par (v_0, w) ((v, w_0)),
- Sinon, le produit est défini par récurrence sur la profondeur des arbres. Le nœud (v_0, w_0) étiqueté par $l_G(v_0) \cap l_H(w_0)$ est la racine de $\mathcal{G}^{\epsilon} \times \mathcal{H}^{\epsilon}$.
  - Pour chaque $\forall r$-successeur v de v_0 dans $\mathcal{G}^{\epsilon}$ et chaque $\forall r$-successeur w de w_0 dans $\mathcal{H}^{\epsilon}$, on obtient un $\forall r$-successeur (v, w) de (v_0, w_0) dans $\mathcal{G}^{\epsilon} \times \mathcal{H}^{\epsilon}$ qui est la racine du produit du sous-arbre $\mathcal{G}^{\epsilon}(v)$ et du sous-arbre $\mathcal{H}^{\epsilon}(w)$, qui est défini récursivement. Le drapeau de (v, w), $f(v, w)$, est affecté par $(f(v), f(w))$.
  - Pour chaque r-successeur v de v_0 dans $\mathcal{G}^{\epsilon}$ et chaque r-successeur w de w_0 dans $\mathcal{H}^{\epsilon}$, on obtient un r-successeur (v, w) de (v_0, w_0) dans $\mathcal{G}^{\epsilon} \times \mathcal{H}^{\epsilon}$ qui est la racine du produit du sous-arbre $\mathcal{G}^{\epsilon}(v)$ et du sous-arbre $\mathcal{H}^{\epsilon}(w)$, qui est défini récursivement. Le drapeau de (v, w), $f(v, w)$, est affecté par $(f(v), f(w))$.
  - Pour chaque ϵ-successeur v de v_0 dans $\mathcal{G}^{\epsilon}$ et w de w_0 dans $\mathcal{H}^{\epsilon}$, on obtient un ϵ-successeur (v, w) de (v_0, w_0) dans $\mathcal{G}^{\epsilon} \times \mathcal{H}^{\epsilon}$ qui est la racine du produit du sous-arbre $\mathcal{G}^{\epsilon}(v)$ et du sous-arbre $\mathcal{H}^{\epsilon}(w)$, qui est défini récursivement. Le drapeau de (v, w), $f(v, w)$, est affecté par $(f(v), f(w))$.

Note 3.3.2. L'évaluation des expressions d'étiquette pour les nœuds de l'arbre de produit ne doit pas être exécutée dans le calcul du produit. Il faut que cette évaluation soit effectué lorsque l'on se réfère à l'arbre de description correspondant, par exemple, dans la procédure de subsomption ou dans la transformation du ϵ-arbre en l'arbre de description. La raison pour cela est que nous n'avons pas suffisamment l'information pour évaluer indépendemment les expressions d'étiquette. L'évaluation indépendante des étiquettes des nœuds d'un ϵ-arbre de description peut amener à un résultat incorrect.

Par exemple, supposons qu'un arbre de produit possède les nœuds v_1, v_2 et les arcs où $l(v_1) = \{\neg P\}$, $l(v_2) = \{P\} \cap \{Q\}$, $(v_1 \epsilon v_2)$. Si l'expression $l(v_1) = \{P\} \cap \{Q\}$ est évaluée avant d'exécuter l'Algorithme 3.2.8, on obtiendra $l(\{v_1, v_2\}) = \{\emptyset\} \cup \{\neg P\} = \{\neg P\}$. Sinon, on a : $l(\{v_1, v_2\}) = eval(\{\{P\} \cap \{Q\}\}, \{\neg P\}) = \{\neg P, Q\}$.

Exemple 3.3.3.

$D_1 := \exists r.(A \sqcap \forall r.C) \sqcap \exists r.(B \sqcap \forall r.B) \sqcap \forall r.(C \sqcap \exists r.A)$, and
$D_2 := \exists r.(A \sqcap B \sqcap C \sqcap \exists r.(A \sqcap B \sqcap C))$
où $A, B, C \in N_C$, $r \in N_R$

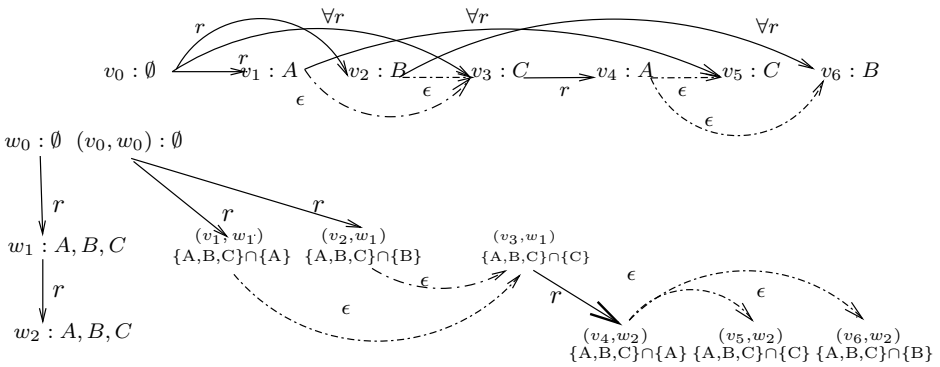

FIG. 3.3.1. Produit $\mathcal{G}^{\epsilon}_{D_1} \times \mathcal{G}^{\epsilon}_{D_2}$

L'arbre de description du produit des $\mathcal{G}^{\epsilon}_{D_1}$, $\mathcal{G}^{\epsilon}_{D_2}$ est décrit dans la Figure 3.3.1.

Nous nous rendons compte qu'un ϵ-arbre de description simple ne contient que les ϵ-arcs qui possèdent la propriété suivante : les deux nœuds reliés par un ϵ-arc sont les destinations de deux arcs normaux (qui ne sont pas des ϵ-arcs). Cependant, lorsque l'on calcule le produit de deux ϵ-arbres de description simple, par exemple selon la Définition 2.2.8, le ϵ-arbre de description obtenu peut contenir un ϵ-arc tel qu'il n'existe pas d'arc normal dont une extrémité est la destination de cet ϵ-arc. C'est pourquoi on a besoin d'une définition du ϵ-arbre de description plus générale.

D'autre part, la Définition 3.2.1 (voisinage) nécessite également une modification pour que tout nœud fasse partie d'un voisinage. Ainsi, la nouvelle définition du voisinage suppose que chaque nœud v d'un ϵ-arbre de description possède une valeur indiquant un nœud, appelé *drapeau* , qui est initialisée par le nœud prédécesseur v' où $(v'ev)$, $e \neq \epsilon$.

Définition 3.3.4. (voisinage étendu)

(1) Un voisinage est un voisinage étendu.

(2) S'il existe un ϵ-arc $(w\epsilon w')$ et le drapeau de w' est un nœud u' où $w \in n^k$, $u \in n^{k-1}$, le voisinage n^k est défini du voisinage n^{k-1}, alors $w' \in n^k$.

Nous avons également la définition suivante pour le ϵ-arbre de description étendu.

Définition 3.3.5. (ϵ-arbre de description étendu)

(1) Un ϵ-arbre de description est un ϵ-arbre de description étendu.

(2) Tout ϵ-arbre de description contient des voisinages étendus est un ϵ-arbre de description étendu.

Avec la nouvelle définition pour le ϵ-arbre de description, on a besoin d'un algorithme qui permet de transformer un ϵ-arbre de description étendu vers l'arbre de description. A l'exception du traitement des voisinages étendus et l'évaluation des expressions d'étiquette, l' algorithme suivant n'est pas différent de l'Algorithme *3.2.8.*

Algorithme 3.3.6.
Entrée : ϵ-arbre de description étendu $\mathcal{G}^\epsilon = (V_G, E_G \cup E_G^\epsilon, v_0, l_G)$.
Sortie : $\mathcal{ALE}$-arbre de description $\mathbf{B}(\mathcal{G}_C^\epsilon)$.

(1) *Au niveau 0, un 0-voisinage $\{v_0, v_{0,1}, ..., v_{0,m}\}$, qui est la racine de $\mathbf{B}(\mathcal{G}_C^\epsilon)$, est créé où $(v_0 \epsilon v_{0,1})$,...,$(v_0 \epsilon v_{0,m})$ et*
$l(\{v_0, v_{0,1}, ..., v_{0,m}\}) = eval(l(v_0), l(v_{0,1}), ..., l(v_{0,m}))$.

(2) *Pour chaque $(k-1)$-voisinage $n^{k-1} = \{v_1, ..., v_m\}$ de $\mathcal{G}_C^\epsilon$ correspondant au $(k-1)$-nœud de $\mathbf{B}(\mathcal{G}_C^\epsilon)$ où $\bot \notin \bigcup_{v_i \in n^{k-1}}^m \{l(v_i)\}$, les k-nœuds et les arcs de $\mathcal{G}$ sont créés d'un k-voisinage du $\mathcal{G}_C^\epsilon$ comme suit :*

   (a) *Un k-voisinage $n^k = \{v'_j \mid$ pour tout arc $(v_i \forall r v'_j)$ où $v_i \in n^{k-1}\}$ et un $\forall r$-arc correspondant $n^{k-1} \forall r n^k$ et,*
   $l(n^k) = eval_{v'_j \in n^k}\{l(v'_j)\}$.

   (b) *Pour chaque arc $(v_i r v'_j)$ où $v_i \in n^{k-1}$ et chaque ensemble $V^\epsilon = \{v'_{j,i} \mid$ pour tout ϵ-arc $(v'_j \epsilon v'_{j,i})$ où $f(v'_{j,i}) \in n^{k-1}\}$, un k-voisinage $m_j^k = n^k \cup \{v'_j\} \cup V^\epsilon$, n^k est déterminé dans l'étape (a) , et les r-arcs correspondants : $n^{k-1} r m_j^k$,*
   $l(m_j^k) = eval_{v' \in m_j^k}\{l(v')\}$.

(3) *Appliquer les règles de normalisation 4. 6. et 7. à l'arbre $\mathbf{B}(\mathcal{G}_C^\epsilon)$ obtenu dans l'étape 2. :*

   (a) *$l(v) \to l(v) \setminus \{\top\}$ où $v \in V_{B(G_C^\epsilon)}$ (règle 4)*

   (b) *$v' r v \to v$ si $l(v) = \{\bot\}$, v est une feuille et $v' r v \in E_{B(G_C^\epsilon)}$ (règle 6)*

   (c) *$\mathcal{G}'_C(v) \to \bot$ où $\bot \in l(v)$ et $\mathcal{G}'(v)$ est un sous-arbre de $\mathbf{B}(\mathcal{G}_C^\epsilon)$ (règle 7)*

Note : la fonction eval *est calculée comme suit :*
Supposons que n^k soit un k-voisinage $\{v_1, ..., v_m\}$ et $l(v_1), ..., l(v_m)$ soient les expressions d'étiquette composées de conjonctions, disjonctions et noms de concept $Q_{i,j}$ où $Q_{i,j} = P$ ou $Q_{i,j} = \neg P$, $P \in N_C$ (plus exactement, si v_i est un nœud du produit, alors $l(v_i)$ est une conjonction d'ensembles d'étiquettes) .

(1) *On définit : $\{l(v_1), ..., l(v_m)\} := \bigcup_i^m l(v_i) =$*
$(Q_{1,1}, Q_{2,1}, ..., Q_{m,1}) \cap ... \cap (Q_{1,m_1}, Q_{2,m_2}, ..., Q_{m,m_m})$

(2) *On désigne $S_{(k_1,...,k_m)} := \{Q_{1,k_1}, Q_{2,k_2}, ..., Q_{m,k_m}\}$ où $k_1 \in \{1..m_1\}, ..., k_m \in \{1..m_m\}$. Donc,*
$\text{eval}(l(v_1), ..., l(v_m)) := \bigcap S_{(k_1,...,k_m)}$ pour tout $S_{(k_1,...,k_m)}$ où $\bot \notin S_{(k_1,...,k_m)}$ et il n'existe pas $P \in N_C$ tel que $P \in S_{(k_1,...,k_m)}$ et $\neg P \in S_{(k_1,...,k_m)}$. Sinon, $eval(l(v_1), ..., l(v_m)) := \{\bot\}$.

3.3.2. Le Plus Spécifique Subsumeur Commun et Produit de ϵ-arbres de description. Maintenant, nous avons préparé le nécessaire pour formuler et démontrer le résultat le plus important de la section. Ce résultat établit la correspondance entre le produit d'arbres de description et celui de ϵ-arbres de description. Cela

implique que le calcul de produit d'arbres de description ou le plus spécifique subsumeur commun peut être se traduire en calcul de produit de ϵ-arbres de description.

Selon la Définition 3.3.1, le produit de deux ϵ-arbres de description peut être un ϵ-arbre de description étendu. Notons que dans la formulation du théorème suivant, l'opérateur $\mathbf{B}$ du côté gauche est appliquée à un ϵ-arbre de description étendu alors que l'opérateurs $\mathbf{B}$ du côté droit sont appliqués aux ϵ-arbres de description simples.

Théorème 3.3.7. *Soient $\mathcal{G}^\epsilon$, $\mathcal{H}^\epsilon$ des ϵ-arbres de description. On a :*

$$\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon) \equiv \mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$$

DÉMONSTRATION. On désigne par $|\mathcal{G}^\epsilon|$ la profondeur de l'arbre $\mathcal{G}^\epsilon$. Supposons que $|\mathcal{G}^\epsilon| \leq |\mathcal{H}^\epsilon|$. Nous montrons le théorème par récurrence sur le niveau de l'arbre $\mathcal{G}^\epsilon$.

Niveau $k = 0$.
Au niveau 0, puisque le produit $\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon$ a un voisinage $\{(v_0, w_0)\}$ unique sans ϵ-arc, $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$ a seulement un nœud $\{(v_0, w_0)\}$. De même, puisque $\mathcal{G}^\epsilon$ a seulement un nœud v_0 sans ϵ-arc et $\mathcal{H}^\epsilon$ a seulement un nœud w_0 sans ϵ-arc au niveau 0, alors $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$ a seulement un nœud $\{(v_0, w_0)\}$.
Si $l_G(v_0) = \emptyset$ alors $l_{G^\epsilon \times H^\epsilon}(v_0, w_0) = \emptyset$, $l_{B(G^\epsilon \times H^\epsilon)}(v_0, w_0) = \emptyset$. De même, on a : $l_{B(G^\epsilon)} = \emptyset$, $l_{B(G^\epsilon) \times B(H^\epsilon)}(v_0, w_0) = \emptyset$.
Supposons que $l_G(v_0) = \{P_1, ..., P_m, \neg Q_1, ..., \neg Q_n\} \neq \emptyset$. Si $l_G(v_0) = \{\bot\}$ alors $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon) \equiv \mathbf{B}(\mathcal{H}^\epsilon)$ et $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon) \equiv \mathbf{B}(\mathcal{H}^\epsilon)$.
Si $l_{G^\epsilon}(v_0) \neq \{\bot\}$, alors $l_{B(G^\epsilon \times H^\epsilon)}(v_0, w_0) = l_{B(G^\epsilon) \times B(H^\epsilon)}(v_0, w_0) = eval(l_{G^\epsilon}(v_0), l_{H^\epsilon}(w_0)) = (l_{G^\epsilon}(v_0) \cap l_{H^\epsilon}(w_0))$.
Par conséquent, dans tous les cas, on obtient : $l_{B(G^\epsilon \times H^\epsilon)}(v_0, w_0) = l_{B(G^\epsilon) \times B(H^\epsilon)}(v_0, w_0)$.

Niveau $k > 0$.
Soient $m^k = \{u_1, ..., u_m\}$ un k-voisinage de $\mathcal{G}^\epsilon$ et $n^k = \{w_1, ..., w_n\}$ un k-voisinage de $\mathcal{H}^\epsilon$. Par l'Algorithme 3.2.8, on obtient de l'arbre $\mathcal{G}^\epsilon$ les $(k{+}1)$-voisinages et les arcs suivants qui relient le k-voisinage m^k à ces $(k{+}1)$-voisinages :

(1) Un $(k{+}1)$-voisinage $m_0^{k+1} = \{v_j |$ pour tout arc $(u_i \forall r v_j)$, $u_i \in m^k\}$ et un $\forall r$-arc $(m^k \forall r m_0^{k+1})$.

(2) Pour chaque r-arc $(u_i r v_j)$, $j = 1..m_1$, un $(k{+}1)$-voisinage $m_j^{k+1} = m_0^{k+1} \cup \{v_j\}$ est créé où $u_i \in m^k$, et un r-arc $(m^k r m_j^{k+1})$ est également créé.

De même, on obtient également de l'arbre $\mathcal{H}^\epsilon$ les $(k{+}1)$-voisinages et les arcs suivants qui relient le k-voisinage n^k à ces $(k{+}1)$-voisinages :

(1) Un $(k{+}1)$-voisinage $n_0^{k+1} = \{y_j |$ pour tout arc $(w_i \forall r y_j)$, $w_i \in n^k\}$ et un $\forall r$-arc $(n^k \forall r n_0^{k+1})$.

(2) Pour chaque r-arc $(w_i r y_j)$, $j = 1..n_1$, un $(k{+}1)$-voisinage $n_j^{k+1} = n_0^{k+1} \cup \{y_j\}$ est créé où $w_i \in n^k$, et un r-arc $(n^k r n_j^{k+1})$ est également créé.

Par la définition de produit 3.3.1, les nœuds et arcs suivants sont construits pour le produit $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$:

(1) Un $\forall r$-arc $(m^k, n^k)\forall r(m_0^{k+1}, n_0^{k+1})$ où $(m^k \forall r m_0^{k+1})$, $(n^k \forall r n_0^{k+1})$, $l(m^k, n^k) = l(m^k) \cap l(n^k)$, $l(m_0^{k+1}, n_0^{k+1}) = l(m_0^{k+1}) \cap l(n_0^{k+1})$ et $l(m^k)$, $l(n^k)$, $l(m_0^{k+1})$, $l(n_0^{k+1})$ sont calculés comme dans l'Algorithme 3.2.8.

(2) Les r-arcs $(m^k, n^k)r(m_i^{k+1}, n_j^{k+1})$ où $(m^k r m_i^{k+1})$, $(n^k r n_j^{k+1})$, $i \in \{1, ..., m_1\}$, $j \in \{1, ..., n_1\}$, $l(m^k, n^k) = l(m^k) \cap l(n^k)$, $l(m_i^{k+1}, n_j^{k+1}) = l(m_i^{k+1}) \cap l(n_j^{k+1})$ et $l(m^k)$, $l(n^k)$, $l(m_i^{k+1})$, $l(n_j^{k+1})$ sont calculés comme dans l'Algorithme 3.2.8.

D'autre part, par l'Algorithme 3.3.6, à partir du k-voisinage $p^k = \{(u_1, w_1), ..., (u_m, w_n)\}$ de l'arbre $\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon$, on obtient les $(k+1)$-voisinages et les arcs comme suit :

(1) Un $(k+1)$-voisinage $p_0^{k+1} = \{(v_i, y_j)|$ pour tout arc $(u_i, w_j)\forall r(v_i, y_j)$ où $(u_i, w_j) \in p^k$, $(u_i \forall r v_i)$, $(w_j \forall r y_j)\}$ et un $\forall r$-arc $(p^k \forall r p_0^{k+1})$ où $l(p^k) = l(u_1, w_1) \cup ... \cup (u_m, w_n)$ et $l(p_0^{k+1}) = \bigcup_{(v_i, y_j) \in p_0^{k+1}} l(v_i, y_j)$.

(2) Pour chaque r-arc $(u_i, w_j)r(v_i, y_j)$, $j = 1..r_1$ où $(u_i, w_j) \in p^k$, et chaque l'ensemble $V^\epsilon = \{(v_l, y_{l'}) \mid$ pour tout ϵ-arc $(u_i, w_j)\epsilon(v_l, y_{l'})$ où $f(v_l, y_{l'}) \in p^k\}$, un $(k+1)$-voisinage $q_{i,j}^{k+1} = p_0^{k+1} \cup \{(v_i, y_j)\} \cup V^\epsilon$ est créé où $(u_i r v_i)$, $(w_j r y_j)$, et un r-arc $(p^k r q_{i,j}^{k+1})$ est créé où $l(p^k) = l(u_1, w_1) \cup ... \cup (u_m, w_n)$ et $l(q_{i,j}^{k+1}) = l(p_0^{k+1}) \cup l(v_i, y_j) \cup \bigcup_{(v_l, y_{l'}) \in V^\epsilon} l(v_l, y_{l'})$.

Par hypothèse de récurrence, au niveau k, pour chaque nœud (m^k, n^k) du produit $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$, on peut trouver un nœud (un k-voisinage) $p^k = \{(u_1, w_1), ..., (u_m, w_n)\}$ de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$ tel que leur étiquette et arcs sont équivalents. C'est-à-dire qu'il existe un isomorphisme φ entre la $k-$partie de l'arbre du $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$ et la $k-$partie de l'arbre du $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$ (la $k-$partie d'un arbre est la partie de la racine au niveau k) . Ainsi, $\varphi(m^k, n^k) = p^k$ où $m^k = \{u_1, ..., u_m\}$ est un k-voisinage de $\mathcal{G}^\epsilon$, $n^k = \{w_1, ..., w_n\}$ est un k-voisinage de $\mathcal{H}^\epsilon$, $p^k = \{(u_1, w_1), ..., (u_m, w_n)\}$ et il existe les ϵ-arcs qui relient les nœuds $(u_1, w_1), ..., (u_m, w_n)$ ensemble.
Maintenant, nous montrons que l'isomorphisme φ entre les deux k-parties des arbres peut être étendu pour obtenir l'isomorphisme φ' entre la $(k+1)$-partie de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$ et la $(k+1)$-partie de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$. Pour cela, il faut montrer que chaque $\forall r$-arc $(m^k, n^k)\forall r(m_0^{k+1}, n_0^{k+1})$ de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$ correspond uniquement à un $\forall r$-arc $(p^k \forall r p_0^{k+1})$ de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$ et que chaque r-arc $(m^k, n^k)r(m_i^{k+1}, n_j^{k+1})$ de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$ correspond uniquement à un r-arc $(p^k r q_{i,j}^{k+1})$ de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$, où $l(m_0^{k+1}, n_0^{k+1}) \equiv l(p_0^{k+1})$ et $l(m_i^{k+1}, n_j^{k+1}) \equiv l(q_{i,j}^{k+1})$.
En effet,

(1) Sur l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$, on a le $\forall r$-arc unique $(m^k, n^k)\forall r(m_0^{k+1}, n_0^{k+1})$ où $l(m_0^{k+1}, n_0^{k+1}) = l(m_0^{k+1}) \cap l(n_0^{k+1}) = \bigcup \{l(v_i) \mid$ pour tout arc $(u_i \forall r v_i)$, $u_i \in m^k\} \cap \bigcup \{l(y_j) \mid$ pour tout arc $(w_j \forall r y_j)$, $w_j \in n^k\}$.
D'autre part, sur l'arbre $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$, on a le $\forall r$-arc unique $(p^k \forall r p_0^{k+1})$ où $l(p_0^{k+1}) = \bigcup\{\ l(v_i, y_j) = l(v_i) \cap l(y_j) \mid$ pour tout arc $(u_i, w_j)\forall r(v_i, y_j)$ où $(u_i, w_j) \in p^k$, $(u_i \forall r v_i)$, $(w_j \forall r y_j)\}$.

En outre, si $(u_i, w_j) \in p^k$, $(u_i \forall r v_i)$, $(w_j \forall r y_j)$, alors par l'hypothèse de récurrence, on a : $u_i \in m^k$, $w_j \in n^k$, et donc les $\forall r$-arcs $(u_i \forall r v_i)$, $(w_j \forall r y_j)$. Par conséquent, on obtient : $l(m_0^{k+1}, n_0^{k+1}) \equiv l(p_0^{k+1})$ et l'isomorphisme est étendu comme suit : $\varphi'(m_0^{k+1}, n_0^{k+1}) = p_0^{k+1}$.

(2) Sur l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$, on a un r-arc $(m^k, n^k) r (m_i^{k+1}, n_j^{k+1})$ où $l(m_i^{k+1}, n_j^{k+1}) = l(m_i^{k+1}) \cap l(n_j^{k+1}) = (l(m_0^{k+1}) \cup l(v_i)\}) \cap (l(n_0^{k+1}) \cup l(y_j)\})$; $(u_i r v_i)$, $u_i \in m^k$, $(w_j r y_j)$, $w_j \in n^k$ et $l(m_0^{k+1})$, $l(n_0^{k+1})$ sont calculés dans 1. D'autre part, sur l'arbre $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$, on a également un r-arc $(p^k r q_{i,j}^{k+1})$ où $l(q_{i,j}^{k+1}) = l(p_0^{k+1}) \cup l(v_i, y_j) \cup \bigcup_{(v_l, y_{l'}) \in V^\epsilon} \{l(v_l, y_{l'})\}$, $V^\epsilon = \{(v_l, y_{l'}) |$ pour tout $(v_i, y_j) \epsilon (v_l, y_{l'})$ où $f(v_l, y_{l'}) \in p^k\}$, $(u_i, w_j) r (v_i, y_j)$, $(u_i, w_j) \in p^k$ et $l(p_0^{k+1})$ est calculé dans 1.
En outre, si $(u_i, w_j) \in p^k$, $(u_i r v_i)$, $(w_j r y_j)$, alors par l'hypothèse de récurrence, on a : $u_i \in m^k$, $w_j \in n^k$, et donc $(u_i r v_i)$, $(w_j r y_j)$. De plus, selon 1., on a : $l(m_0^{k+1}) \cap l(n_0^{k+1}) = l(p_0^{k+1})$.
Il nous reste à montrer que :

(a) Sur l'arbre $(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$, pour chaque r-arc $(u_i, w_j) r (v_i, y_j)$ où $(u_i, w_j) \in p^k$ et pour tout $v_h \in m_0^{k+1}$ et $y_h \in n_0^{k+1}$, on a : $(v_i, y_j) \epsilon (v_i, y_h) \in V^\epsilon$, $(v_i, y_j) \epsilon (v_h, y_j) \in V^\epsilon$. En effet, sur l'arbre $\mathcal{G}^\epsilon$, on a le r-arc, les $\forall r$-arcs et ϵ-arcs suivants : $(u_i r v_i)$, $(u_h \forall r v_h)$, $(u_i \epsilon u_h)$, $(v_i \epsilon v_h)$, $u_h \in m^k$, $v_h \in m_0^{k+1}$. Sur l'arbre $\mathcal{H}^\epsilon$, on a le r-arc, les $\forall r$-arcs et ϵ-arcs suivants : $(w_j r y_j)$, $(w_h \forall r y_h)$, $(w_j \epsilon w_h)$, $(y_j \epsilon y_h)$, $w_h \in n^k$, $y_h \in n_0^{k+1}$. Par la Définition de Produit 3.3.1, on obtient les ϵ-arcs suivants sur l'arbre $(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$: $(v_i, y_j) \epsilon (v_i, y_h)$, $(v_i, y_j) \epsilon (v_h, y_j)$, où $v_h \in m_0^{k+1}$, $y_h \in n_0^{k+1}$ et $f(v_i, y_h) = (f(v_i), f(y_h)) = (u_i, w_h)$, $f(v_h, y_j) = (f(v_h), f(y_j)) = (u_h, w_j)$. Puisque, par l'hypothèse de récurrence, $\varphi'(m^k, n^k) = p^k$, $w_h \in n^k$ et $u_h \in m^k$, alors $f(v_i, y_h) = (u_i, w_h) \in p^k$ et $f(v_h, y_j) = (u_h, w_j) \in p^k$ sur l'arbre $(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$. Donc, $(v_j, y_j) \epsilon (v_j, y_h)$, $(v_j, y_j) \epsilon (v_h, y_j) \in V^\epsilon$.

(b) Inversement, s'il existe un ϵ-arc : $(v_i, y_j) \epsilon (v_i, y_h) \in V^\epsilon$ où $(u_i, w_j) r (v_i, y_j)$ et $(u_i, w_j) \in p^k$, alors $y_h \in n_0^{k+1}$ sur l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$. En effet, puisque $(v_i, y_j) \epsilon (v_i, y_h) \in V^\epsilon$ et $\mathcal{G}^\epsilon$, $\mathcal{H}^\epsilon$ sont les ϵ-arbres de description, alors on a également : $(u_i, w_j) \epsilon (u_i, w_h)$ ou $(u_i, w_h) \epsilon (u_i, w_j)$, $(u_i, w_j), (u_i, w_h) \in p^k$. Puisque, sur l'arbre $\mathcal{G}^\epsilon$ il y a le r-arc $(w_j r y_j)$ et ϵ-arc $(y_j \epsilon y_h)$, alors il faut un $\forall r$-arc $(w_h \forall r y_h)$. De plus, par l'hypothèse de récurrence, $\varphi'(m^k, n^k) = p^k$ et $(u_i, w_h) \in p^k$, alors $w_h \in n^k$. Cela implique que $y_h \in n_0^{k+1}$. Le même argument pour l'arc $(v_i, y_j) \epsilon (v_h, y_j) \in V^\epsilon$, on obtient également $v_h \in m_0^{k+1}$ sur l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$.

A partir de l'affirmation ci-dessus, on obtient :
$(l(m_0^{k+1}) \cup l(v_j)\}) \cap (l(n_0^{k+1}) \cup l(y_j)\}) \equiv l(p_0^{k+1}) \cup l(v_j, y_j) \cup \bigcup_{(v_l, y_l) \in V^\epsilon} \{l(v_l, y_l)\}$.
En conséquence, on peut étendre l'isomorphisme comme suit : pour chaque $(k+1)$-voisinage (m_i^{k+1}, n_j^{k+1}) de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon) \times \mathbf{B}(\mathcal{H}^\epsilon)$ correspond uniquement au $(k+1)$-voisinage q_j^{k+1} de l'arbre $\mathbf{B}(\mathcal{G}^\epsilon \times \mathcal{H}^\epsilon)$ *i.e* $\varphi'(m_i^{k+1}, n_j^{k+1}) = q_j^{k+1}$.

□

Exemple 3.3.8. A partir du ϵ-arbre de description étendu décrit dans la Figure 3.3.1, on applique l'Algorithme 3.3.6 à cet arbre comme suit :
Puisque $f(v_1, w_1) = f(v_2, w_1) = f(v_3, w_1) = (v_0, w_0)$ et $(v_1, w_1)\epsilon(v_2, w_1)$, $(v_2, w_1)\epsilon(v_3, w_1)$, alors au niveau 1, on a les deux 1-voisinages étendus suivants : $[(v_1, w_1), (v_2, w_1)]$ et $[(v_1, w_1), (v_3, w_1)]$. Ces voisinages correspondent aux deux arcs suivants $[(v_0, w_0)]r[(v_1, w_1), (v_2, w_1)]$ et $[(v_0, w_0)]r[(v_1, w_1), (v_3, w_1)]$.
Pour 1-voisinage $[(v_2, w_1), (v_3, w_1)]$, au niveau 2 on a : $f(v_4, w_2) = (v_3, w_1) \in \{(v_2, w_1), (v_3, w_1)\}$, $f(v_6, w_2) = (v_2, w_1) \in \{(v_2, w_1), (v_3, w_1)\}$ et $(v_4, w_2)\epsilon(v_6, w_2)$, un 2-voisinage $[(v_4, w_2), (v_6, w_2)]$ qui est créé correspond au arc $[(v_2, w_1), (v_3, w_1)]r[(v_4, w_2), (v_6, w_2)]$.
De même, pour 1-voisinage $[(v_1, w_1), (v_3, w_1)]$, au niveau 2 on a $f(v_4, w_2) = (v_3, w_1) \in \{(v_1, w_1), (v_3, w_1)\}$, $f(v_5, w_2) = (v_1, w_1) \in \{(v_1, w_1), (v_3, w_1)\}$ et $(v_5, w_2)\epsilon(v_6, w_2)$, un 2-voisinage $[(v_5, w_2), (v_6, w_2)]$ qui est créé correspond au arc $[(v_1, w_1), (v_3, w_1)]r[(v_5, w_2), (v_6, w_2)]$. Finalement, chaque nœud du produit a une expression d'étiquette qui est une conjonction de disjonctions de noms de concept.

Dans l'exemple suivant qui est extrait de **[BT02 (4)]**, la taille du *lcs* (irréductible) peut être exponentielle en fonction de la taille des descriptions de concept d'entrée dans la représentation ordinaire. Cependant, la taille de cette *lcs* est polynômiale dans la représentation compacte.

Exemple 3.3.9. (extrait du **[BT02 (4)]**)
Soient

$$\begin{aligned} C_3 := \quad & \exists r.(\forall r.\forall r.P_3^0) \sqcap \exists r.(\forall r.\forall r.P_3^1) \sqcap \\ & \forall r.(\exists r.\forall r.P_2^0 \sqcap \exists r.\forall r.P_2^1 \sqcap \forall r.(\exists r.P_1^0 \sqcap \exists r.P_1^1)) \\ D_3 := \quad & \exists r.\exists r.\exists r.(P_1^0 \sqcap P_1^1 \sqcap P_2^0 \sqcap P_2^1 \sqcap P_3^0 \sqcap P_3^1) \end{aligned}$$

La Figure 3.3.2 illustre la représentation graphique de l'Exemple 3.3.9.

Remarque 3.3.10. (Complexité)

(1) La taille du produit de deux ϵ-arbres de description qui est calculé comme dans la Définition *3.3.1,* est au plus un produit des tailles de ces deux ϵ-arbres de description. En effet, la ϵ-normalisation d'un arbre de description (sans application de la règle 2.) n'accroît pas le nombre des nœuds de l'arbre. De plus, le nombre des ϵ-arcs ajoutés est borné par n^2 où n est le nombre de nœuds de l'arbre. D'autre part, soient m, n les nombres des nœuds de deux arbres $\mathcal{G}^\epsilon$ et $\mathcal{H}^\epsilon$. Le produit calculé par la Définition *3.3.1* permet d'obtenir un arbre dont le nombre de nœuds est borné par $m.n$ et le nombre des arcs du produit est également borné par $(m.n)^2$.

(2) La complexité de calcul en temps de l'opérateur **B** pour un ϵ-arbre de description étendu est au plus exponentielle en fonction de la taille de l'arbre et cet opérateur nécessite une espace polynômiale. En effet, afin de déterminer un voisinage étendu, on a besoin de suivre les ϵ-arcs et de vérifier

les drapeaux des nœuds de destination. Puisque le nombre de nœuds au niveau k est borné par le nombre total de nœuds n et le nombre de ϵ-arcs est borné par n^2, alors le traçace des ϵ-arcs et la vérification des drapeaux ne modifient pas la complexité de calcul en temps de l'opérateur **B**. En outre, l'Algorithme 3.3.6 (l'algorithme de **B** pour le ϵ-arbre de description étendu) nécessite seulement une mémoire pour stocker un chemin de la racine vers une feuille de l'arbre résultat et chaque nœud au long de ce chemin est un voisinage. Par conséquent, cet algorithme est PSPACE en fonction de la taille n.

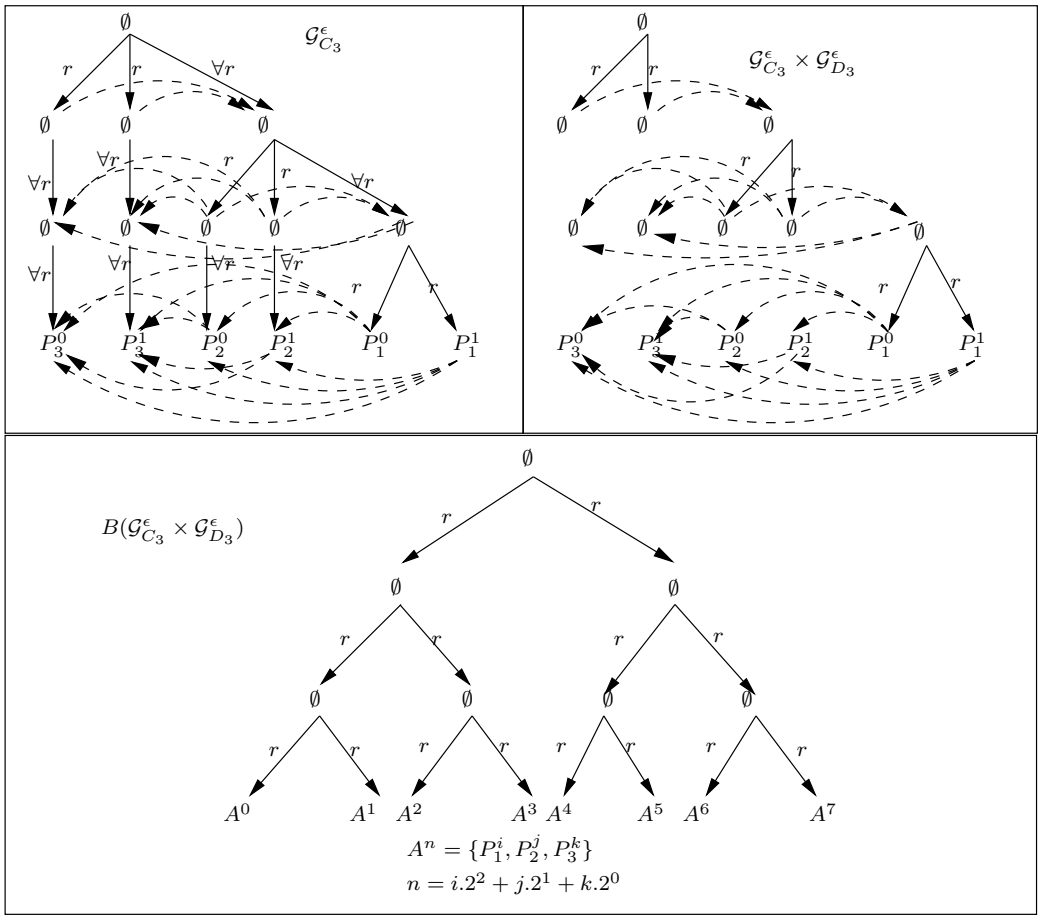

FIG. 3.3.2. Produit $\mathcal{G}^{\epsilon}_{C_3} \times \mathcal{G}^{\epsilon}_{D_3}$ et sa représentation ordinaire obtenue par l'Algorithme 3.3.6

La Proposition *3.2.11* et le Théorème 3.3.7 nous permettent de représenter le Plus Spécifique Subsumeur Commun (*lcs*) de deux $\mathcal{ALE}$-descriptions de concept C, D comme un ϵ-arbre de description étendu dont la taille est un produit des tailles des

ϵ-arbres de description correspondant à C et à D. Pour obtenir la représentation ordinaire du *lcs*, on applique l'opérateur **B** au ϵ-arbre de description étendu.

Ce résultat nous permet de calculer le ϵ-arbre du Plus Spécifique Subsumeur Commun de deux $\mathcal{ALE}$-descriptions de concept en temps polynômial. Cela implique que l'on peut calculer le Plus Spécifique Subsumeur Commun de deux $\mathcal{ALE}$-descriptions de concept sans normalisation par la règle 2. dans la Définition 2.2.4. Comme indiqué dans [**BKM99 (5)**] et [**BT02 (4)**], l'application de cette règle accroît de façon exponentielle la taille du Plus Spécifique Subsumeur Commun de deux $\mathcal{ALE}$-descriptions de concept.

3.4. Sur le Calcul de l'Approximation $\mathcal{ALC}$-$\mathcal{ALE}$

Cette section a pour objectif d'éclaircir la relation entre le calcul du Plus Spécifique Subsumeur Commun (*lcs*) dans le langage $\mathcal{ALE}$ et celui de l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$. A partir de cela, nous proposons une nouvelle procédure de calcul pour l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$. Le deuxième élément constitutif de la section est l'introduction de l'exemple qui montre dans le pire des cas l'inexistence d'un algorithme exponentiel en fonction de la taille d'entrée pour le calcul de l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$. Cet exemple montre également qu'un algorithme *doublement exponentiel* pour l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$ est optimal dans le sens où il n'existe pas d'algorithme (pour calculer cette approximation) appartenant à une classe de complexité inférieure si la représentation des descriptions de concept actuelle est utilisée (y inclus la représentation compacte présentée dans la Section 3.2).

3.4.1. Le Plus Spécifique Subsumeur Commun et Approximation $\mathcal{ALC}$-$\mathcal{ALE}$.

Effectivement, le calcul du $lcs(C_1, C_2)$ considéré comme un produit des arbres de description $\mathcal{G}_{C_1} \times \mathcal{G}_{C_2}$, qui est présenté dans la Définition 2.2.8, préserve un conjonct commun des deux descriptions de concept C_1 et C_2 si ces descriptions de concept sont normalisées par les règles de normalisation dans la Définition 2.2.4. Ce conjonct commun correspond à la partie commune des deux arbres $\mathcal{G}_{C_1}, \mathcal{G}_{C_2}$. D'autre part, l'approximation d'une disjonction peut être calculé comme le Plus Spécifique Subsumeur Commun des approximations de chaque disjonct. Ces affirmations sont formulées et montrées dans la proposition suivante.

Proposition 3.4.1.
Soit C une $\mathcal{ALC}$-description de concept où $C := C_1 \sqcup C_2$ and $\bot \sqsubset C_1, C_2$.

(1) *Si $A = A_1 \sqcap C$ et $B = B_1 \sqcap C$ sont les $\mathcal{ALE}$-descriptions de concept normalisées par les règles de normalisation dans la Définition 2.2.4, on a :*

$$lcs(A, B) \equiv C \sqcap lcs(A_1, B_1)$$

(2) *Si $C = C_1 \sqcup C_2$ où C est une $\mathcal{ALC}$-description de concept et $\bot \sqsubset C_1, C_2$, alors*

$$approx_{\mathcal{ALE}} C \equiv lcs\{approx_{\mathcal{ALE}}(C_1),\ approx_{\mathcal{ALE}}(C_2)\}$$

Démonstration.

(1) Selon la Définition de Produit 2.2.8, l'arbre de description $\mathcal{G}_A$ est une concaténation exacte des deux arbres de description $\mathcal{G}_{A_1}$ et $\mathcal{G}_C$ car A et B sont les $\mathcal{ALE}$-descriptions de concept normalisée. De même, l'arbre de description $\mathcal{G}_B$ est une concaténation exacte des deux arbres de description $\mathcal{G}_{B_1}$ and $\mathcal{G}_C$. A partir de la construction du produit pour $lcs(A, B)$ dans la Définition 2.2.8, on obtient :

$lcs(A, B) = lcs(C, C) \sqcap lcs(C, B_1) \sqcap lcs(C, A_1) \sqcap lcs(A_1, B_1) =$
$C \sqcap lcs(C, B_1) \sqcap lcs(C, A_1) \sqcap lcs(A_1, B_1)$ (*).

D'autre part, il est évident que : $C \sqcap lcs(A_1, B_1) \sqsubseteq lcs(C, A_1)$ et
$C \sqcap lcs(A_1, B_1) \sqsubseteq lcs(C, B_1)$. Donc,
$C \sqcap lcs(A_1, B_1) \sqsubseteq lcs(C, A_1) \sqcap lcs(C, B_1)$(**).

A partir de (*) et de (**), on obtient :
$lcs(A, B) = C \sqcap lcs(C, B_1) \sqcap lcs(C, A_1) \sqcap lcs(A_1, B_1) = C \sqcap lcs(A_1, B_1)$.

(2) Supposons que la description de concept C soit écrite comme suit : $C = C_1 \sqcup C_2$ où $\bot \sqsubset C_1, C_2$. D'abord, on a :
$C = C_1 \sqcup C_2 \sqsubseteq lcs(approx_{\mathcal{ALE}}(C_1), approx_{\mathcal{ALE}}(C_2))$.
Supposons qu'il existe une $\mathcal{ALE}$-description de concept D telle que
$C = C_1 \sqcup C_2 \sqsubseteq D \sqsubseteq lcs(approx_{\mathcal{ALE}}(C_1), approx_{\mathcal{ALE}}(C_2))$.

Si $approx_{\mathcal{ALE}}(C_1) \sqcup approx_{\mathcal{ALE}}(C_2) \sqsubseteq D$ alors,
$D \equiv lcs(approx_{\mathcal{ALE}}(C_1), approx_{\mathcal{ALE}}(C_2))$
car $approx_{\mathcal{ALE}}(C_1) \sqcup approx_{\mathcal{ALE}}(C_2) \sqsubseteq lcs(approx_{\mathcal{ALE}}(C_1), approx_{\mathcal{ALE}}(C_2))$ et $lcs(approx_{\mathcal{ALE}}(C_1), approx_{\mathcal{ALE}}(C_2))$ est une $\mathcal{ALE}$-description de concept.

Si $approx_{\mathcal{ALE}}(C_1) \sqcup approx_{\mathcal{ALE}}(C_2) \not\sqsubseteq D$, il existe une interprétation $(\Delta, .^{\mathcal{I}})$ et un individu d tel que $d^{\mathcal{I}} \in (approx_{\mathcal{ALE}}(C_1) \sqcup approx_{\mathcal{ALE}}(C_2))^{\mathcal{I}}$ et $d^{\mathcal{I}} \notin D^{\mathcal{I}}$. Notons que $d^{\mathcal{I}}$ existe car C_1 et C_2 sont consistants. Donc, on a les deux possibilités suivantes :

- si $d^{\mathcal{I}} \in (approx_{\mathcal{ALE}}(C_1))^{\mathcal{I}}$ et $d^{\mathcal{I}} \notin D^{\mathcal{I}}$ alors $C_1 \sqsubseteq D \sqcap approx_{\mathcal{ALE}}(C_1) \sqsubset approx_{\mathcal{ALE}}(C_1)$, qui contredit la définition d'approximation car $(D \sqcap approx_{\mathcal{ALE}}(C_1))$ est une $\mathcal{ALE}$-description de concept.
- si $d^{\mathcal{I}} \in (approx_{\mathcal{ALE}}(C_2))^{\mathcal{I}}$ et $d^{\mathcal{I}} \notin D^{\mathcal{I}}$ alors $C_2 \sqsubseteq D \sqcap approx_{\mathcal{ALE}}(C_2) \sqsubset approx_{\mathcal{ALE}}(C_2)$, qui contredit la définition d'approximation car $(D \sqcap approx_{\mathcal{ALE}}(C_2))$ est une $\mathcal{ALE}$-description de concept.

□

Un algorithme pour le calcul de l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$ résulte directement de la Proposition 3.4.1. Un avantage de cet algorithme par rapport à l'Algorithme *2.2.14* est qu'il peut fournir une implémentation plus simple. Pour des raisons de simplicité, on écrit $approx(C)$ à la place de $approx_{\mathcal{ALE}}(C)$.

Algorithme 3.4.2. (Approximation simple)
Entrée : C est une $\mathcal{ALC}$-description de concept dans la $\mathcal{ALC}$-forme normale :
$C = C_1 \sqcup ... \sqcup C_n$
Sortie : $approx(C)$

(1) *Si* $C \equiv \bot$ *alors, approx(C)* $:= \bot$;
(2) *Si* $C \equiv \top$ *alors, approx(C)* $:= \top$.
(3) *Si* $n = 1$, $approx(C) \equiv$
$\bigsqcap_{A \in prim(C_1)} A \sqcap \bigsqcap_{C' \in ex(C_1)} \exists r.approx(C' \sqcap val(C_1)) \sqcap \forall r.approx(val(C_1))$
(4) *Sinon,* $approx(C) \equiv lcs\{approx(C_1),..., approx(C_n)\}$

L'Algorithme 3.4.2 ainsi que l'Algorithme *2.2.14* est dans le pire des cas doublement exponentiel en fonction de la taille d'entrée. Une question laissée ouverte par les auteurs du travail [**BKT02a (15)**] concerne l'existence d'un *algorithme exponentiel* pour le calcul de l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$.

Dans la première tentative afin de trouver une réponse à la question, nous avons espéré que l'on peut diminuer la complexité du calcul de l'approximation en baissant la taille des *lcs* car cette approximation est calculée via *lcs*. Cependant, par la suite, nous nous rendons compte que la classe de complexité du calcul de l'approximation ne change pas dans le pire des cas malgré la taille polynômiale du *lcs* de deux descriptions de concept. En effet, d'une part le calcul de l'approximation exige des calculs de *lcs* qui s'appliquent à un nombre exponentiel de descriptions de concept. D'autre part, le calcul de *lcs* n'est plus polynômial lorsque le nombre de descriptions de concept, auquel ce calcul s'applique, s'élève à $n > 2$.

Pour cette raison, nous nous penchons à la réponse négative à cette question. Dans la deuxième tentative, nous construisons un exemple, *i.e* une $\mathcal{ALC}$-description de concept C, dont la taille de l'approximation sous forme irréductible est doublement exponentielle en fonction de la taille de C. Par conséquent, dans le pire des cas, la taille doublement exponentielle de l'approximation est inévitable malgré la nouvelle représentation des $\mathcal{ALE}$-descriptions de concept.

Le reste de la section est consacré à la construction de cet exemple et aux calculs qui montrent ses propriétés attendues.

3.4.2. Un exemple pour une $\mathcal{ALC}$-$\mathcal{ALE}$ approximation doublement exponentielle. Selon l'Algorithme *2.2.14,* l'accroissement exponentiel résultant de l'interaction entre des restrictions existentielles et universelles peut apparaître au premier niveau de l'arbre de description. Simultanément, un autre accroissement exponentiel peut être produit par la $\mathcal{ALC}$-normalisation dès le premier niveau. C'est pourquoi la $\mathcal{ALC}$-description de concept pour l'exemple pourrait être construite de telle sorte que les *lcs* obtenus du calcul de l'approximation au second niveau soient irréductibles.

Soient,

$A_k^1 = \exists r.(P_k^1 \sqcap \bigsqcap_{i=1..n, i \neq k}(P_i^1 \sqcap P_i^2) \sqcap Q_1 \sqcap Q_2)$

$A_k^2 = \exists r.(P_k^2 \sqcap \bigsqcap_{i=1..n, i \neq k}(P_i^1 \sqcap P_i^2) \sqcap Q_1 \sqcap Q_2)$

pour $k \in \{1, ..., n\}$ et,

$B_1 = \exists r.(Q_1 \sqcap \bigsqcap_{i=1..n}(P_i^1 \sqcap P_i^2))$

$B_2 = \exists r.(Q_2 \sqcap \bigsqcap_{i=1..n}(P_i^1 \sqcap P_i^2))$

où $r \in N_R$ et P_k^i, Q_j sont les noms de concept (N_C) pour $i, j \in \{1, 2\}$, $k \in \{1, ..., n\}$.

Maintenant, la $\mathcal{ALC}$-description de concept C est définie comme suit :

$C := \exists r.B_1 \sqcap \exists r.B_2 \sqcap \bigsqcap_{i=1}^{n}(\forall r.A_i^1 \sqcup \forall r.A_i^2)$

Il faut montrer que la taille de $approx_{ALE}(C)$ n'est pas inférieure à 2^{2^n} et il n'existe pas de $\mathcal{ALE}$-description de concept C', qui est équivalent à $approx_{ALE}(C)$, dont la taille est inférieure à 2^{2^n}.

Un cas particulier : $n = 2$. Pour faciliter la lecture de la démonstration dans le cas général, on calcule d'abord $approx_{ALE}(C)$ dans le cas $n = 2$. Effectivement, C peut être écrite en forme normale comme suit :

$$\begin{aligned} C \equiv \quad & (\exists r.(B_1 \sqcap A_1^1 \sqcap A_2^1) \sqcap \exists r.(B_2 \sqcap A_1^1 \sqcap A_2^1) \sqcap \forall r.(A_1^1 \sqcap A_2^1)) \sqcup \\ & (\exists r.(B_1 \sqcap A_1^1 \sqcap A_2^2) \sqcap \exists r.(B_2 \sqcap A_1^1 \sqcap A_2^2) \sqcap \forall r.(A_1^1 \sqcap A_2^2)) \sqcup \\ & (\exists r.(B_1 \sqcap A_1^2 \sqcap A_2^1) \sqcap \exists r.(B_2 \sqcap A_1^2 \sqcap A_2^1) \sqcap \forall r.(A_1^2 \sqcap A_2^1)) \sqcup \\ & (\exists r.(B_1 \sqcap A_1^2 \sqcap A_2^2) \sqcap \exists r.(B_2 \sqcap A_1^2 \sqcap A_2^2) \sqcap \forall r.(A_1^2 \sqcap A_2^2)) \end{aligned}$$

Selon l'Algorithme *2.2.14*, on obtient 2^{2^2} restrictions existentielles
$\{C(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4}) \mid (l_1, l_2, l_3, l_4) \in \{1, 2\} \times \{1, 2\} \times \{1, 2\} \times \{1, 2\}\}$,
et une restriction universelle. Par exemple, une restriction existentielle parmi elles est la suivante :

$C(B_2, B_1, B_1, B_1) = \exists r.lcs\{B_2 \sqcap A_1^1 \sqcap A_2^1,\ B_1 \sqcap A_1^1 \sqcap A_2^2,\ B_1 \sqcap A_1^2 \sqcap A_2^1,\ B_1 \sqcap A_1^2 \sqcap A_2^2\}$

A priori, le calcul de l'expression *lcs* retourne un ensemble $E(B_2, B_1, B_1, B_1)$ comportant les 3^4 restrictions existentielles. Chacune soit subsume au moins un élément d'un des deux groupes suivants :

$EX_{(I)}(B_2, B_1, B_1, B_1) = \{\exists r.(Q_1 \sqcap Q_2 \sqcap P_1^1 \sqcap P_1^2),\ \exists r.(Q_1 \sqcap Q_2 \sqcap P_2^1 \sqcap P_2^2)\}$

$EX_{(II)}(B_2, B_1, B_1, B_1) = \{\exists r.(P_1^1 \sqcap P_1^2 \sqcap P_2^1 \sqcap P_2^2),\ \exists r.(P_1^1 \sqcap P_1^2 \sqcap P_2^1 \sqcap P_2^2)\}$

soit, est *subsumée* au moins par un élément du groupe suivant :

$EX_{(III)}(B_2, B_1, B_1, B_1) = \{\exists r.(Q_2 \sqcap P_1^2 \sqcap P_2^2),\ \exists r.(Q_1 \sqcap P_1^2 \sqcap P_2^1),$

$\exists r.(Q_1 \sqcap P_1^1 \sqcap P_2^2)$, $\exists r.(Q_1 \sqcap P_1^1 \sqcap P_2^1)\}$

Il est évident que :
$$E(B_2, B_1, B_1, B_1) = \begin{array}{l} EX_{(I)}(B_2, B_1, B_1, B_1) \cup EX_{(II)}(B_2, B_1, B_1, B_1) \cup \\ EX_{(III)}(B_2, B_1, B_1, B_1) \end{array} .$$

Maintenant, soit $C(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4}) \in \{C(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4}) \mid (l_1, l_2, l_3, l_4) \in \{1,2\} \times \{1,2\} \times \{1,2\} \times \{1,2\}\}$ où $(l_1, l_2, l_3, l_4) \neq (2,1,1,1)$, $(l_1, l_2, l_3, l_4) \neq (1,1,1,1)$, $(l_1, l_2, l_3, l_4) \neq (2,2,2,2)$. Sans perte de généralité, supposons que $l_1 = 1$, $l_2 = 2$.

A partir de la définition du groupe $EX_{(III)}(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4})$ ci-dessus et $l_1 = 1$, on choisit $e'' \in EX_{(III)}(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4})$ et $e'' = \exists r.(Q_1 \sqcap P_1^2 \sqcap P_2^2)$. Nous montrons que :

(1) $e' \not\sqsubseteq e''$ pour tout $e' \in EX_{(I)}(B_2, B_1, B_1, B_1)$. En effet, $\{Q_1, P_1^2, P_2^2\} \not\subseteq \{Q_1, Q_2\} \cup \bigcup_{l=1, l\neq h}^{2} \{P_l^1, P_l^2\}$ for $h \in \{1..2\}$.

(2) $e' \not\sqsubseteq e''$ pour tout $e' \in EX_{(II)}(B_2, B_1, B_1, B_1)$. En effet, $\{Q_1, P_1^2, P_2^2\} \not\subseteq \bigcup_{l=1}^{2} \{P_l^1, P_l^2\}$.

(3) $e' \not\sqsubseteq e''$ pour tout $e' \in EX_{(III)}(B_2, B_1, B_1, B_1)$. En effet, pour tout e' telle que $e' \notin EX_{(I)}(B_2, B_1, B_1, B_1)$ et soit $e' \sqsubseteq \exists r.(Q_1 \sqcap P_1^2 \sqcap P_2^1)$, soit $e' \sqsubseteq \exists r.(Q_1 \sqcap P_1^1 \sqcap P_2^2)$, soit $e' \sqsubseteq \exists r.(Q_1 \sqcap P_1^1 \sqcap P_2^1)$, on a : $e' \not\sqsubseteq e''$. En outre, pour tout e' telle que $e' \notin EX_{(II)}(B_2, B_1, B_1, B_1)$ et $e' \sqsubseteq \exists r.(Q_2 \sqcap P_1^2 \sqcap P_2^2)$, on a également : $e' \not\sqsubseteq e''$.

(4) $e' \not\sqsubseteq e''$ pour tout $e' \in E(B_2, B_2, B_2, B_2)$. C'est évident.

(5) $e' \not\sqsubseteq e''$ pour tout $e' \in E(B_1, B_1, B_1, B_1)$. On peut choisir $f'' \in EX_{(III)}(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4})$ et $f'' = \exists r.(Q_2 \sqcap P_1^2 \sqcap P_2^1)$ car $l_2 = 2$. On a : $e' \not\sqsubseteq f''$ pour tout $e' \in E(B_1, B_1, B_1, B_1)$.

De même, nous pouvons montrer qu'il existe une restriction existentielle $e_1 \in EX_{(III)}(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4})$ telle que $f' \not\sqsubseteq e_1$ pour tout $f' \in E(B_2, B_1, B_1, B_1) \cup E(B_2, B_2, B_2, B_2)$ et qu'il existe une restriction existentielle $e_2 \in EX_{(III)}(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4})$ telle que $f'' \not\sqsubseteq e_2$ pour tout $f'' \in E(B_1, B_1, B_1, B_1)$.

En conséquence, nous avons montré que pour tout $e, f \in \{C(B_{l_1}, B_{l_2}, B_{l_3}, B_{l_4}) \mid (l_1, l_2, l_3, l_4) \in \{1,2\} \times \{1,2\} \times \{1,2\} \times \{1,2\}\}$, on a : $e \not\sqsubseteq f$ et $f \not\sqsubseteq e$.

Le cas général. Le point essentiel du calcul de la $approx(C)$ dans le cas général réside, comme dans le cas $n = 2$, en division de l'ensemble E en les groupes $E_{(I)}$, $E_{(II)}$ et $E_{(III)}$. En effet, C peut être écrite en forme normale :

$C \equiv C_1 \sqcup ... \sqcup C_m$ où
$C_i = (\exists r.B_1 \sqcap \exists r.B_2 \sqcap \forall r.val(C_i))$ et $val(C_i) = A_1^{i_2} \sqcap ... \sqcap A_n^{i_n}$,
$(i_1, ..., i_n) \in (\{1,2\} \times ... \times \{1,2\})$

Selon l'Algorithme *2.2.14*, $approx_{ALE}(C)$ est calculée comme suit :

$$approx_{ALE}(C) \equiv \bigsqcap_{(B'_1,...,B'_m)\in(\{B_1,B_2\}\times...\times\{B_1,B_2\})} \exists r.\mathrm{lcs}\{(B'_i \sqcap val(C_i))|1 \leq i \leq m\} \sqcap \forall r.\mathrm{lcs}\{val(C_i) \mid 1 \leq i \leq m\}$$

Notons que dans l'approximation ci-dessus, pour chaque $(B'_1, ..., B'_m) \in (\{B_1, B_2\} \times ... \times \{B_1, B_2\})$, les termes B'_i et $val(C_i)$ pour $1 \leq i \leq m$ contiennent seulement des restrictions existentielles. On désigne par $E(B'_1, ..., B'_m)$ l'ensemble de restrictions existentielles obtenues du calcul de $lcs\{(B'_i \sqcap val(C_i))|1 \leq i \leq m\}$ pour chaque $(B'_1, ..., B'_m)$. L'ensemble $E(B'_1, ..., B'_m)$ peut être divisé en les trois groupes suivants :

(I) $E_1(B'_1, ..., B'_m)$ se compose des éléments qui subsument les restrictions existentielles suivantes :
$\exists r.lcs(ex(A^1_k), ex(A^2_k)) \equiv \exists r.(Q_1 \sqcap Q_2 \sqcap \bigsqcap^n_{l=1,l\neq k}(P^1_l \sqcap P^2_l))$ pour $k \in \{1..n\}$.

Il est évident que $\exists r.lcs(ex(A^1_k), ex(A^2_k)) \in E(B'_1, ..., B'_m)$ pour tout $k \in \{1..n\}$. On désigne par $EX_{(I)}(B'_1, ..., B'_m)$ l'ensemble : $EX_{(I)}(B'_1, ..., B'_m) := \{Q_1 \sqcap Q_2 \sqcap \bigsqcap^n_{l=1,l\neq k}(P^1_l \sqcap P^2_l))|1 \leq k \leq n\}$

(II) $E_2(B'_1, ..., B'_m)$ se compose des éléments qui subsument les restrictions existentielles suivantes :
$\exists r.lcs(ex(B_i), ex(B_j)) \equiv \exists r.(\bigsqcap^n_{k=1}(P^1_k \sqcap P^2_k))$ s'il existe B'_i, B'_j qui appartiennent à $(B'_1, ..., B'_m)$ et $B'_i \neq B'_j$, ou $\exists r.lcs(ex(B_i), ex(B_i)) \equiv \exists r.(Q_i \sqcap \bigsqcap^n_{k=1}(P^1_k \sqcap P^2_k))$ pour $i \in \{1, 2\}$ si $B'_1 = ... = B'_m$. On a également :$\exists r.lcs(ex(B_i), ex(B_j)) \in E$ et $\exists r.lcs(ex(B_i), ex(B_i)) \in E$ pour $i \in \{1, 2\}$.
On désigne par $EX_{(II)}(B'_1, ..., B'_m)$ l'ensemble

$EX_{(II)}(B'_1, ..., B'_m) := \{\bigsqcap^n_{k=1}(P^1_k \sqcap P^2_k)\}$,

et par $EX^i_{(II)}(B'_1, ..., B'_m)$ l'ensemble

$EX^i_{(II)}(B'_1, ..., B'_m) = \{(Q_i \sqcap \bigsqcap^n_{k=1}(P^1_k \sqcap P^2_k)\}$ pour $i \in \{1, 2\}$.

(III) D'abord, on considère les restrictions existentielles dont les expressions sous ces restrictions existentielles sont les *lcs* qui s'appliquent à un $ex(B'_k)$ et $ex(A^{i_1}_1)$, $ex(A^{i_2}_2)$,..., $ex(A^{i_n}_n)$ pour un certain $k, 1 \leq k \leq m$ et $(i_1, ..., i_n) \in (\{1, 2\} \times ... \times \{1, 2\})$. Pour déterminer la relation entre k et $(i_1, ..., i_n)$, on utilise la propriété suivante : pour chaque $(i_1, ..., i_n) \in (\{1, 2\}, ..., \{1, 2\})$ il existe uniquement $k \in \{1, ..., m\}$ tel que $val(C_k)$ ne contient pas A^l_1 pour tout $l \in \{i_1, ..., i_n\}$. Ainsi, on désigne $\bar{I}(k) = (\bar{i}_1, ..., \bar{i}_n)$ où $val(C_k) = (A^{i_1}_1 \sqcap A^{i_2}_2 \sqcap ... \sqcap A^{i_n}_n)$ et $\bar{i}_h \neq i_h$ pour tout $h = 1..n$, et $A^{\bar{I}(k)} = \{A^{\bar{i}_1}_1, ..., A^{\bar{i}_n}_n\}$. Donc,

$E_3(B'_1, ..., B'_m)$ se compose des éléments *subsumés* par les restrictions existentielles suivantes :
$\exists r.lcs\{ex(B'_k), ex(A^{\bar{i}_1}_1), ex(A^{\bar{i}_2}_1), ..., ex(A^{\bar{i}_n}_n)\}$ où $B'_k \in \{B'_1, ..., B'_m\}$, $B'_k \in ex(C_k)$, $A^{\bar{I}(k)} = \{A^{\bar{i}_1}_1, ..., A^{\bar{i}_n}_n\}$. De plus , on a :

$\exists r.lcs\{ex(B'_k), ex(A_1^{\bar{i}_1}), ex(A_1^{\bar{i}_2}), ..., ex(A_1^{\bar{i}_n})\} \equiv \exists r.(Q_{i_k} \sqcap P_1^{\bar{i}_1} \sqcap P_1^{\bar{i}_2} \sqcap ... \sqcap P_1^{\bar{i}_n})$ où $B'_k \in \{B'_1, ..., B'_m\}$, $B'_k = \exists r.(Q_{i_k} \sqcap \bigsqcap_{i=1..n}(P_i^1 \sqcap P_i^2))$ et $P^{\bar{I}(k)} = \{P_1^{\bar{i}_1}, ..., P_n^{\bar{i}_n}\}$.

On désigne par $EX_{(III)}(B'_1, ..., B'_m)$ l'ensemble
$EX_{(III)}(B'_1, ..., B'_m) := \{Q_{i_k} \sqcap P_1^{\bar{i}_1} \sqcap P_2^{\bar{i}_2} \sqcap ... \sqcap P_n^{\bar{i}_n} |\ B'_k = \exists r.(Q_{i_k} \sqcap \bigsqcap_{i=1..n}(P_i^1 \sqcap P_i^2)), \bar{I}(k) = (\bar{i}_1, ..., \bar{i}_n), 1 \leq k \leq m\}$.

Afin de montrer $E(B'_1, ..., B'_m) = E_1(B'_1, ..., B'_m) \cup E_2(B'_1, ..., B'_m) \cup E_3(B'_1, ..., B'_m)$, montrons que si $e \in E(B'_1, ..., B'_m)$, $(B'_1, ..., B'_m) \notin \{(B_1, ..., B_1), (B_2, ..., B_2)\}$ et $e \notin E_3(B'_1, ..., B'_m)$, alors $e \in E_1(B'_1, ..., B'_m) \cup E_2(B'_1, ..., B'_m)$. En effet, si $e \notin E_3(B'_1, ..., B'_m)$ et $e \notin E_2(B'_1, ..., B'_m)$, alors :

$e = \exists r.lcs\{ex(B'_k), ex(A'_1), ex(A'_2), ..., ex(A'_p)\}$ où $A^{\bar{I}(k)} \subset \{A'_1, ..., A'_p\}$, $B'_k \in \{B'_1, ..., B'_m\}$, $B'_k \in ex(C_k)$, $k \in \{1, ..., m\}$. Puisque $p > n$, il existe A_v^i,$A_v^j \in \{A'_1, ..., A'_p\}$, $1 \leq v \leq n$, $i, j \in \{1, 2\}$. Cela implique que $e \in E_1(B'_1, ..., B'_m)$.

Par conséquent, $approx_{ALE}(C) \equiv$

$$\bigsqcap_{\substack{(B'_1,...,B'_m)\notin\{(B_1,...,B_1),(B_2,...B_2)\} \\ (B'_1,...,B'_m)\in(ex(C_1)\times...\times ex(C_m))}} \exists r.\Big(\bigsqcap_{e'\in E_3\{B'_1,...,B'_m\}} e' \sqcap \bigsqcap_{k=1}^{n} \exists r.(Q_1 \sqcap Q_2 \sqcap \bigsqcap_{l=1,l\neq k}^{n}(P_l^1 \sqcap P_l^2)) \sqcap \exists r.(\bigsqcap_{k=1}^{n}(P_k^1 \sqcap P_k^2)) \Big) \sqcap$$

$$\exists r.\Big(\bigsqcap_{e'\in E_3(B_1,...,B_1)} e') \sqcap \bigsqcap_{k=1}^{n} \exists r.(Q_1 \sqcap Q_2 \sqcap \bigsqcap_{l=1,l\neq k}^{n}(P_l^1 \sqcap P_l^2)) \sqcap \exists r.(\bigsqcap_{k=1}^{n}(Q_1 \sqcap P_k^1 \sqcap P_k^2)) \Big) \sqcap$$

$$\exists r.\Big(\bigsqcap_{e'\in E_3(B_2,...,B_2)} e' \sqcap \bigsqcap_{k=1}^{n} \exists r.(Q_1 \sqcap Q_2 \sqcap \bigsqcap_{l=1,l\neq k}^{n}(P_l^1 \sqcap P_l^2)) \sqcap \exists r.(\bigsqcap_{k=1}^{n}(Q_2 \sqcap P_k^1 \sqcap P_k^2)) \Big) \sqcap$$

$$\forall r.\Big(\bigsqcap_{(j_1,...,j_n)\in(\{1,2\}\times...\times\{1,2\})} \exists r.(P_1^{j_1} \sqcap ... \sqcap P_n^{j_n} \sqcap Q_1 \sqcap Q_2) \sqcap \bigsqcap_{k=1}^{n} \exists r.(Q_1 \sqcap Q_2 \sqcap \bigsqcap_{l=1,l\neq k}^{n}(P_l^1 \sqcap P_l^2)) \Big)$$

Maintenant, nous montrons que le nombre des restrictions existentielles dans $approx_{ALE}(C)$ est 2^{2^n} et ces restrictions existentielles ne subsument pas réciproquement. En effet,

Soit $(B'_1, ..., B'_m), (B''_1, ..., B''_m) \in \{(ex(C_1) \times ... \times ex(C_m))\} \setminus \{(B_1, ..., B_1), (B_2, ..., B_2)\}$ et $k \in \{1, .., m\}$ tel que $B'_k \neq B''_k$. Supposons que :

$B'_k = \exists r.(Q_1 \sqcap \bigsqcap_{l=1}^{n}(P_l^1 \sqcap P_l^2))$ et $B''_k = \exists r.(Q_2 \sqcap \bigsqcap_{l=1}^{n}(P_l^1 \sqcap P_l^2))$. Soit $e'' = \exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n}) \in E_3(B''_1, ..., B''_m)$ où $\exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n}) = \exists r.lcs\{ex(B''_k), ex(A_1^{\bar{i}_1}), ..., ex(A_n^{\bar{i}_n})\}$ et $\{A_1^{\bar{i}_1}, ..., A_n^{\bar{i}_n}\} = A^{\bar{I}(k)}$. D'abord, montrons que $e' \not\sqsubseteq e''$ pour tout $e' \in E(B'_1, ..., B'_m)$.

- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n})$ pour tout $e' \in E_1(B'_1, ..., B'_m)$ car $\{Q_2, P_1^{\bar{i}_1}, ..., P_n^{\bar{i}_n}\} \not\subseteq \{Q_1, Q_2\} \cup \bigcup_{l=1, l\neq h}^{n}\{P_l^1, P_l^2\}$ pour $h \in \{1..n\}$.

- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n})$ pour tout $e' \in E_2(B'_1, ..., B'_m)$ car $\{Q_2, P_1^{\bar{i}_1}, ..., P_n^{\bar{i}_n}\} \not\subseteq \bigcup_{l=1}^{n}\{P_l^1, P_l^2\}$.

- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n})$ pour tout $e' \in E_3(B'_1, ..., B'_m)$, $e' \notin E_1(B'_1, ..., B'_m)$ et $e' \notin E_2(B'_1, ..., B'_m)$. En effet, si e' contient Q_1, l'affirmation est évidente car $e' \notin E_2(B'_1, ..., B'_m)$. Sinon, puisque $B'_k = \exists r.(Q_1 \sqcap \bigsqcap_{v=1}^{n}(P_v^1 \sqcap P_v^2))$ contient Q_1, alors on obtient : $e' = \exists r.lcs\{ex(B'_l) \sqcap ex(A'_1) \sqcap ... \sqcap (A'_p)\}$ où $B'_l = \exists r.(Q_2 \sqcap \bigsqcap_{v=1}^{n}(P_v^1 \sqcap P_v^2))$, pour certain $l \in \{1, ..., m\}$, $l \neq k$. Ainsi, il existe $h \in \{1, ..., n\}$ tel que $A_h^{i_h} \in \{A'_1, ..., A'_p\}$ où $\{A_1^{\bar{i}_1}, ..., A_n^{\bar{i}_n}\} = A^{\bar{I}(k)}$ car B'_k contient Q_1 et $l \neq k$. Puisque $\{A'_1, ..., A'_p\} \subseteq \{A_1^{\bar{j}_1}, ..., A_n^{\bar{j}_n}\} = A^{\bar{I}(l)}$ et $e' \notin E_1(B'_1, ..., B'_m)$, alors $P_h^{\bar{i}_h} \notin \{P'_1, ..., P'_q\}$ où $e' = \exists r.lcs\{ex(B'_l) \sqcap ex(A'_1) \sqcap ... \sqcap ex(A'_p)\} = \exists r.(Q_2 \sqcap P'_1 \sqcap ... \sqcap P'_q)$. En conséquence, $\{Q_2, P_1^{\bar{i}_1}, ..., P_n^{\bar{i}_n}\} \not\subseteq \{Q_2, P'_1, ..., P'_q\}$.

Ensuite, montrons que $e' \not\sqsubseteq e''$ pour tout $e' \in E(B_1, ..., B_1)$.

- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n})$ pour tout $e' \in E_1(B_1, ..., B_1)$ car $\{Q_2, P_1^{\bar{i}_1}, ..., P_n^{\bar{i}_n}\} \not\subseteq \{Q_1, Q_2\} \cup \bigcup_{l=1, l\neq h}^{n}\{P_l^1, P_l^2\}$ pour $h \in \{1..n\}$.

- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n})$ pour tout $e' \in E_2(B_1, ..., B_1)$ car $\{Q_2, P_1^{\bar{i}_1}, ..., P_n^{\bar{i}_n}\} \not\subseteq \{Q_1\} \cup \bigcup_{l=1}^{n}\{P_l^1, P_l^2\}$.

- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P_1^{\bar{i}_1} \sqcap ... \sqcap P_n^{\bar{i}_n})$ pour tout $e' \in E_3(B_1, ..., B_1)$ car $\{Q_2, P_1^{\bar{i}_1}, ..., P_n^{\bar{i}_n}\} \not\subseteq \{Q_1\} \cup \{P'_1, ..., P'_q\}$.

Afin de montrer $e' \not\sqsubseteq e''$ pour tout $e' \in E(B_2, ..., B_2)$, soit $e'' = \exists r.(Q_1 \sqcap P_1^{\bar{j}_1} \sqcap ... \sqcap P_n^{\bar{j}_n}) \in E_3(B''_1, ..., B''_m)$ où $\exists r.(Q_1 \sqcap P_1^{\bar{j}_1} \sqcap ... \sqcap P_n^{\bar{j}_n}) = \exists r.lcs\{ex(B''_l), ex(A_1^{\bar{j}_1}), ..., ex(A_n^{\bar{j}_n})\}$, $l \neq k$ et $\{A_1^{\bar{j}_1}, ..., A_n^{\bar{j}_n}\} = A^{\bar{J}(l)}$.

De même, on peut montrer qu'il existe $e' \in E(B'_1, ..., B'_m)$ tel que $e'' \not\sqsubseteq e'$ pour tout $e'' \in E(B''_1, ..., B''_m) \cup E(B_1, ..., B_1)$ et $f' \in E(B'_1, ..., B'_m)$ tel que $e'' \not\sqsubseteq f'$ pour tout $e'' \in E(B_2, ..., B_2)$.

Il nous reste à montrer qu'il n'existe pas de $\mathcal{ALE}$-description de concept D telle que $D \equiv approx_{ALE}(C)$ et le nombre de restrictions existentielles de D est inférieur à 2^{2^n}. Supposons qu'il existe une description de concept D dont le nombre de restrictions existentielles de D est inférieur à 2^{2^n}. Puisque $C \sqsubseteq D$, la profondeur de D est inférieure ou égale à 2. Donc, il existe des restrictions existentielles $\exists r.C_1, \exists r.C_2$ de $approx_{ALE}(C)$ et une restriction existentielle $\exists r.D_1$ de D telles que

$D_1 \equiv C_1$, $D_1 \equiv C_2$. Cela implique que $C_1 \sqsubseteq C_2$ qui contredit la propriété montrée de $approx_{ALE}(C)$.

Remarque 3.4.3. La taille doublement exponentielle de $approx_{ALE}(C)$ vient des calculs *lcs* des 2^n restrictions existentielles, ni de la normalisation par les règles dans la Définition 2.2.4, ni de la normalisation selon la Définition 2.2.12. En effet, dans le langage $\mathcal{EL}$, la taille exponentielle d'un *lcs* peut résulter de l'application de ce *lcs* à des n $\mathcal{EL}$-description de concept [**BKM99 (5)**].

3.5. Conclusion

La représentation compacte pour les $\mathcal{ALE}$-descriptions de concept nous permet de décider la subsomption en espace polynômiale et de calculer le Plus Spécifique Subsumeur Commun de deux $\mathcal{ALE}$-descriptions de concept en temps polynômial. Par conséquent, la complexité du calcul de la $\mathcal{ALC}$-$\mathcal{ALE}$ approximation est diminuée à l'aide de cette représentation compacte car ce calcul peut être effectué sur la représentation polynômiale à la place de la représentation exponentielle de descriptions de concepts qui résulte de la normalisation.

Cependant, cette amélioration ne permet pas de changer la classe de complexité du calcul de la $\mathcal{ALC}$-$\mathcal{ALE}$ approximation. L'existence de la $\mathcal{ALC}$-description de concept, qui est présenté dans la Section 3.4 de ce chapitre, montre que dans le pire des cas, un algorithme exponentiel pour le calcul de l'approximation de $\mathcal{ALC}$-descriptions de concept par une $\mathcal{ALE}$- description de concept n'existe pas si les représentations actuelles, y inclus la représentation compacte, sont utilisées.

Une fois l'identification de la source de complexité du calcul de l'approximation est faite, des directions de recherche s'ouvrent. Une de ces directions est l'identification des cas pratiques dans lesquels la complexité du calcul peut être diminuée grâce à une sémantique affaiblie de l'approximation. La deuxième direction est inspirée de la Remarque 3.4.3. En effet, l'accroissement exponentiel de *lcs* pour n $\mathcal{EL}$-descriptions de concept peut être évité à l'aide d'une définition raisonnable de "voisinage" pour le nœud de l'arbre de description.

Finalement, la notion d'approximation a besoin d'une extension à d'autres constructeurs. Par exemple, le problème plus général concernant l'approximation d'une D.L L_s par une autre D.L L_d de telle sorte que l'ensemble de constructeurs de L_d soit un sous-ensemble de l'ensemble de constructeurs de L_s, est envisageable.

CHAPITRE 4

Révision de la Terminologie et Règles de Révision

4.1. Introduction

Dans les chapitres précédents, nos études s'appuient sur l'hypothèse de connaissances statiques, c'est-à-dire que les définitions des concepts ne changent pas dans le temps. Cependant, l'évolution des bases de connaissances, en particulier la terminologie, est inéluctable. Dans l'Exemple d'Introduction, nous avons présenté un scénario d'échanges qui montre la nécessité de la révision d'une base de connaissances terminologique. En effet, les définitions partagées peuvent être formalisées comme une terminologie. Si l'échange prend en compte le contexte, la règle devrait être appliquée pour permettre de *partager la compréhension* du terme **ProdRésAuPolluant** entre les deux acteurs. Un ajout simple d'un nouveau concept à la terminologie au lieu de la modification de la définition du concept n'est pas satisfaisant. Une modification de la définition du terme en conservant le nom **ProdRésAuPolluant** semble de capturer la compréhension intuitive de ce que la révision envisage. Alors, chaque référence à **ProdRésAuPolluant** dans les inférences sur la terminologie devrait désormais être interprétée selon la nouvelle définition du terme.

Le sujet de recherche sur la révision d'une base de connaissances est connu sous le nom : *révision de croyance*s. Les approches sémantiques proposées dans [**AGM85 (1)**] ne peuvent pas être directement utilisées pour les terminologies. La raison est que d'une part les problèmes essentiels posés par la révision d'une base de connaissances générale sont au-delà du cadre de la logique du premier ordre, d'autre part certains de ces problèmes deviennent évidents pour les terminologies. Par exemple, lorsque l'on ajoute une règle à une base de règles existante, il faut assurer que la base modifiée est toujours consistante. Par contre, un ajout d'un concept avec la définition à une terminologie ne cause jamais d'inconsistance (sous l'hypothèse que la terminologie accepte un concept insatisfiable). C'est à dire que la terminologie serait immunisée contre une telle sorte de modifications.

Cependant, la connaissance terminologique est un fondement de communication car elle est le vocabulaire utilisé pour interagir avec le monde. Il est évident que nous devons faire face à la possibilité selon laquelle la définition d'un terme n'est pas correcte à cause d'un malentendu ou d'une évolution de sens dans le temps. Donc, dans certains cas il est impératif de réviser la terminologie pour répondre à des besoins

pragmatiques. Une application dans laquelle une révision de la terminologie se déclenche par une valeur contextuelle, est présentée dans l'Exemple d'Introduction.

Ce chapitre commence par une brève introduction au canevas AGM (Alchourrón, Gärdenfors, Makinson) pour la révision d'une théorie. Ce canevas nous permet d'établir les principes fondamentaux sur lesquels différentes approches de la révision de la terminologie s'appuient. Par la suite, nous montrons que d'une part les approches basées sur la sémantique ne peuvent pas être directement appliquées à la révision de la terminologie, d'autre part l'approche syntaxique proposée dans [**Neb90a (58)**] ne respecte pas toujours les principes. Cela nous amène à construire une autre méthode, appelée *approche structurelle*, qui est présentée dans la troisième section du chapitre. Les opérations de révision issues de l'approche structurelle sont capables de traiter un langage L.D avec le constructeur existentiel ($\mathcal{FLE}$).

A partir des opérations de révision, les règles, appelées *règles de révision*, sont introduites dans la section suivante du chapitre, dont l'antécédent est une expression prenant en compte le contexte où la terminologie a besoin d'être révisée et, le conséquent est une opération de révision. La sémantique du *formalisme hybride* composé de la terminologie et des règles, est également présentée. Ce chapitre se termine par une introduction de l' inférence sur ce formalisme hybride.

4.2. Du Canevas AGM à la Révision de la Terminologie

Cette section commence par un exemple. Supposons qu'une B.C contienne les morceaux d'informations suivants :
α : Tous les français savent faire du vélo
β : Mon ami est français
γ : Mon ami vient de Côte d'Azûr
δ : Côte d'Azûr est une région de la France.
Si l'on a un moteur d'inférence relié à la BC, le fait suivant est déduit de $\alpha - \delta$:
ϵ : Mon ami sait faire du vélo.
Un jour je découvre que mon ami ne sait pas faire du vélo. Cela m'oblige d'ajouter $\neg\epsilon$ à la B.C. Toutefois, notre B.C devient incohérent. Si nous voulons maintenir la cohérence de la B.C, cette dernière nécessite une révision. C'est-à-dire qu'un certain nombre de croyances doivent être enlevées. Il est évident que nous ne souhaitons pas abandonner toutes les croyances car certaines croyances sont encore valables. Donc, nous devons choisir quelques croyances parmi $\alpha - \delta$ à enlever. Le problème de la révision des croyances est que la seule considération logique ne nous permet pas de déterminer la croyance à enlever. De plus, le fait que les croyances de la B.C comportent également les conséquences logiques complique davantage les choses. Par conséquence, lorsque nous abandonnons une croyance, nous devons décider de quelles conséquences à ajouter ou à enlever. Par exemple, α a deux conséquences comme suit :
α' : Tous les français savent faire du vélo sauf mon ami (on ne sait pas si mon ami sait faire du vélo)

α'' : Tous les français savent faire du vélo sauf certaines personnes qui viennent de Côte d'Azûr (on ne sait pas si ces personnes savent faire du vélo).
Si nous décidons d'enlever α pour la situation décrite au-dessus, quelles conséquences garderions-nous dans la B.C ?
La révision de croyance est un processus dans lequel on change les croyances de soi-même pour refléter l'acquisition de nouvelles informations. Si l'on considère une théorie comme un ensemble de croyances (propositions), l'abandon d'une croyance ou l'acceptation d'une nouvelle croyance correspondent aux opérations de contraction ou de révision, respectivement, sur cette théorie. Sans compter la contrainte de succès, l'exécution de ces opérations doit subir d'autres contraintes comme suit : (i) la théorie modifiée devrait être représentable par le langage utilisé ; (ii) la théorie devrait être toujours cohérente lorsque l'on ajoute (ou enlève) une nouvelle information ; (iii) la théorie devrait changer le moins possible.
Soient K un ensemble de propositions et P une nouvelle proposition. L est un langage comportant ces propositions et les constructeurs $\{\neg, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$ qui sont utilisés pour former de nouvelles propositions. L'opération de conséquence Cn, qui transforme un ensemble de propositions en un autre ensemble, satisfait les propriétés suivantes :

$$\begin{array}{rcl} K & \subseteq & Cn(K) \text{ (inclusion)} \\ Cn(K) & = & Cn(Cn(K)) \text{ (itération)} \\ Cn(K) & \subseteq & Cn(K') \text{ si } K \subseteq K' \text{ (monotonicité)} \end{array}$$

Supposons que K est un ensemble de propositions fermé par l'opération Cn (*i.e.* une théorie). On désigne par $K * P$ le résultat de la révision de K en ajoutant P et en procédant à d'autres modifications si nécessaires. Intuitivement, $K * P$ est K qui est modifié au minimum pour adapter P. Si P est logiquement cohérent avec K, P est simplement ajouté à K. Dans ce cas, on désigne :

$$K + P := Cn(K \cup \{P\}) \text{ où “+” est l'opérateur d'expansion.}$$

On désigne également un autre opérateur par $K - P$ où “$-$” est l'opérateur de contraction. Puisque l'ensemble de propositions K est logiquement fermé, P ne peut pas être enlevé de K sans prendre en compte les autres membres de K desquels P est déduit. Par conséquent, on doit enlever à la fois P et les autres membres dont P est conséquence logique.
Par la suite, le canevas AGM établit la relation entre l'opération de révision $K * P$ et celle de contraction $K - P$ en utilisant l'identité suivantes comme hypothèse :

$$\text{Identité de Levi : } K * P = (K - \neg P) + P$$

Grâce à cette identité, le processus de révision (opérateur $*$) peut être remplacé par le processus de contraction (opérateur $-$). Afin de développer la théorie de révision sous ces hypothèses, les auteurs (Alchourrón, Gärdenfors, Makinson) ont proposé l'ensemble des postulats que les opérations de révision et de contraction devraient satisfaire [**AGM85 (1)**].

(K1) $K - P$ est une théorie (fermeture)
(K2) $K - P \subseteq K$ (inclusion)
(K3) Si $P \notin K$, $K - P = K$ (vacuité)
(K4) Si $P \notin Cn(\emptyset)$, $P \notin K - P$ (succès)
(K5) Si $Cn(P) = Cn(Q)$, $K - P = K - Q$ (préservation)
(K6) $K \subseteq Cn(K - P) \cup \{P\}$(récupération)

Le postulat (K4) assure le succès de l'opération c'est-à-dire la théorie devrait être changée par l'exécution de l'opération. Les postulats (K2), (K3) et (K6) garantissent une modification minimale causée par l'opération. En effet, (K2) dit que rien d'inconnu n'entrera dans la théorie lorsque l'opération est effectuée. (K6) donne une borne inférieure pour l'opération et (K3) capture le cas limite où rien n'est enlevé.
Cependant, l'opérateur de contraction reste toujours problématique. En effet, un des problèmes essentiels du canevas AGM est de spécifier quels membres seraient enlevés avec P c'est-à-dire l'ensemble de propositions enlevées de K avec P devrait être minimal. Cela est capturé par la définition d'un ensemble $K \downarrow P$ comme suit : les sous-ensembles maximaux K' de K qui ne contiennent pas P et tel qu'aucun membre de K qui n'appartenant pas à K' ne peut être ajouté sans déduire P. A partir de cela, $K - P$ serait un membre de $K \downarrow P$. Il est évident que dans certains cas le résultat de l'opérateur de contraction n'est pas unique. Il y a quelques approches qui tentent de surmonter cet obstacle. Pour avoir plus de détails, voir [**AGM85 (1)**]. Une issue à ce problème est de supposer que certaines propositions soient plus importantes que d'autres. Cela est interprété dans le sens que certaines propositions sont emboîtées plus profondément que d'autres dans l'ensemble de propositions. Donc, elles sont plus difficiles à enlever.
En bref, en partant des hypothèses répandues et d'un ensemble de postulats rationnels, le canevas AGM offre un cadre pour définir des opérations de révision sur l'ensemble de propositions respectant ces hypothèses et l'ensemble de postulats.

Maintenant, nous projetons le canevas AGM sur les problèmes de la révision d'une base de connaissances générale.
L'opération DIRE correspondant à l'opération de révision ajoute une nouvelle connaissance à la BC. L'opération OUBLIER correspondant à l'opération de contraction enlève une connaissance de la BC. Formellement, soient $\mathcal{BC}$ un ensemble de toute base de connaissances possible, L un langage pour exprimer les requêtes de révision (langage de requête) :

$$\begin{aligned} \text{DIRE} : & \mathcal{BC} \times L \rightarrow \mathcal{BC} \\ \text{OUBLIER} : & \mathcal{BC} \times L \rightarrow \mathcal{BC} \end{aligned}$$

A partir du canevas AGM, les principes suivants sont identifiés pour que les opérations de révision les satisfassent [**Neb90a (58)**] :

(P1) **Bonne adaptation du langage de requête**. Ce critère exige que les expressions de requête de révision soient interprétables par des termes du formalisme utilisé dans la BC (Base de Connaissances).

(P2) **Indépendance de syntaxe**. Si les expressions de requête de révision sont sémantiquement équivalentes, les opérations respectives causent les mêmes effets sur la *BC*.
(P3) **Fermeture**. Ce critère exige que n'importe quelle opération de révision doive amener une $BC \in \mathcal{BC}$ à un état unique interprétable par le formalisme utilisé *i.e.* DIRE(*BC, Exp*) et OUBLIER(*BC, Exp*) doivent être bien définies et DIRE(*BC, Exp*), OUBLIER(*BC,Exp*) $\in \mathcal{BC}$ où $BC \in \mathcal{BC}$ et *Exp* est une expression de requête de révision.
(P4) **Succès**. Ce critère exige que n'importe quelle opération de révision doive être réussie, c'est-à-dire que après DIRE la nouvelle connaissance, expression de requête (*Exp*), doit être dérivable de la BC et après OUBLIER la connaissance enlevée (*Exp*) doit ne plus être dérivable de la BC.
(P5) **Modification minimale** (selon une mesure de minimalité). Cela nous entraîne à définir la distance entre deux états de la BC. Des considérations pragmatiques sont exigées pour fixer le sens de la notion "*modification minimale*" dans les cas réels.

Les principes (P1), (P2) et (P4) sont évidents (les principes (P2), (P4) reflètent respectivement le postulat de préservation (K5) et le postulat de succès (K4)). Cependant, le principe (P3) qui reflète le postulat de fermeture (K1) est plus problématique dans le contexte de la révision d'une base de connaissances terminologiques. Nous allons discuter de ce sujet dans le paragraphe qui suit.
Le principe (P5) capture les postulats (K2), (K6) et (K3). En effet, le postulat d'inclusion (K2) assure que la contraction n'ajoute pas de nouvelle connaissance à la base ; le postulat de récupération (K6) présente la borne inférieure pour la contraction ; et le postulat de vacuité (K3) dit que rien n'est effectué si la contraction n'est pas nécessaire.

A priori, ces principes ne s'adaptent pas directement à la révision d'une base de connaissances terminologiques. Une adaptation de ces principes à la terminologie nécessite une correspondance entre la théorie (BC) et la terminologie. Cela implique la question suivante : *à quoi dans la terminologie correspond chaque proposition d'une théorie ?*

Il existe deux points de vue différents sur cette question. Ces points de vue sont reflétés dans les approches qui sont étudiées dans les paragraphes suivants.

4.2.1. Approche sémantique. Cette approche se base sur le niveau de connaissances. Les connaissances que le système possède à n'importe quel moment sont caractérisées par l'ensemble de relations de subsomption possibles. Cela implique que les propositions d'une théorie sont projetées sur des relations de subsomption d'une terminologie. La première conséquence de telle approche est que l'opération OUBLIER ne peut pas être définie car, au niveau de connaissances il n'existe pas de manière de décider si une relation de subsomption résulte d'une définition particulière (cela contredit le principe (P3)). Le second point important de cette approche est que la sémantique du formalisme terminologique est définie comme l'ensemble des relations de subsomption de toute terminologie possible. L'opération DIRE sélectionne simplement un sous-ensemble de cet ensemble des relations c'est-à-dire qu'elle restreint

l'ensemble des terminologies possibles. Plus de connaissances sont acquises, plus l'ensemble de possibilités est réduit. Dans une telle spécification, la forme syntaxique de la définition d'un terme p n'est pas importante ; l'importance est la relation entre p et les autres termes. Par conséquent, toute terminologie dont les ensembles des relations de subsomption déduites sont les mêmes, est équivalente. Par exemple, $\mathcal{T}_1 \equiv \mathcal{T}_2$ où

$$\mathcal{T}_1=\{A := B \sqcap C\,;\, D \sqsubseteq A \sqcap E\}\text{ , et}$$

$$\mathcal{T}_2=\{A := B \sqcap C\,;\, D \sqsubseteq B \sqcap C \sqcap E\}$$

Il est évident que nous ne pouvons pas définir l'opération OUBLIER comme un enlèvement de l'introduction d'un terme dont l'effet est la suppression d'une relation de subsomption. Toutefois, nous pouvons réviser directement la terminologie comme suit :

$$\text{OUBLIER}(k, D \sqsubseteq A)\ (*)$$

où k est l'ensemble des relations de subsomption de la terminologie.
Selon le canevas AGM, cette opération correspond exactement à l'opération de contraction. De plus, l'opération DIRE correspond à l'opération d'expansion car DIRE ajoute une relation de subsomption sans causer aucune d'incohérence.
D'autre part, dans une base de connaissances terminologiques, il n'existe pas de manière d'exprimer la contre-partie de la disjonction ou de la négation *i.e.* nous ne pouvons pas dire que :

$$\neg(D \sqsubseteq A) \text{ ou } (D \sqsubseteq A) \vee (B \sqsubseteq C)$$

Pour expliquer cette impossibilité, nous reprenons la définition de l'opération OUBLIER (*). Cette opération peut amener à enlever l'introduction de A ou celle de D. Puisqu'aucune de mesure ne permet d'estimer le meilleur enlèvement, on peut utiliser la disjonction des deux introductions. Le formalisme terminologique autorise la conjonction de deux concepts comme suit :

$$A \sqcup B$$

Pourtant, cette expression est loin d'être

$$(A \sqsubseteq X) \vee (B \sqsubseteq Y)$$

4.2.2. Approche Conservatrice. Cette approche nous permet de modifier la définition d'un concept C en ajoutant un fragment à la définition ou enlevant un fragment de la définition. Le concept C et tous les concepts définis via C ont maintenant une nouvelle définition. De plus, l'approche exige que la modification de C ne devrait pas violer les relations de subsomption. C'est-à-dire que si $A \sqsubseteq B$ est vérifié avant la modification pour des concepts A, B dans le TBox, alors $A \sqsubseteq B$ reste toujours vérifié après la modification. Cette exigence s'adapte au point de vue dans lequel les relations de subsomption sont considérées comme les connaissances invariantes de la BCT. Ce fait est bien compatible avec le modèle de la base de données orienté-objet pourvu que la structure hiérarchique de classes soit persistante, même si les définitions de classes pourraient être changées. Dans ce modèle, chaque fois qu'une

définition de classe est changée, toutes les instances de l'ancienne classe doivent rester les instances de la nouvelle classe. En effet, l'évolution de notre compréhension sur le monde nous amène à des définitions avec plus de précisions. Ainsi, les objets qui vérifient l'ancienne définition sont maintenant décrits plus complètement par la nouvelle définition. Autrement dit, nous avons besoin de changer la définition d'un concept C alors que toutes les relations de subsomption sont préservées et toutes les assertions $C(a)$ (ainsi que tout $D(a)$ où D est changé à cause de la propagation de la modification) pour le nouveau concept C restent également vérifiées. Cette idée reflète la monotonicité de la base de connaissance.

Cependant, l'approche conservatrice ne peut pas éviter certaines incompatibilités entre les principes lorsque nous restons aux définitions d'inférences sous l'*hypothèse du monde ouvert*. En effet, nous supposons que l'opération de révision DIRE est définie comme suit :

- DIRE(C, Exp) := C si $C \sqsubseteq Exp$, sinon
- DIRE(C, Exp) := $C \sqcap Exp$ si $C \not\sqsubseteq Exp$,

où C est un nom de concept et Exp est une expression de requête de révision.

Selon la préservation d'assertion proposée par l'approche conservatrice, si $C^{\mathcal{I}}(a^{\mathcal{I}})$ est vérifiée, alors $(DIRE(C, Exp))^{\mathcal{I}}(a^{\mathcal{I}})$ est également vérifiée pour toute interprétation $\mathcal{I}$. C'est à dire que

$$C \sqsubseteq \text{DIRE}(C, Exp)$$

pour toute interprétation $\mathcal{I}$. Donc,

$$C \equiv \text{DIRE}(C, Exp) \text{ pour toute } Exp,$$

car DIRE(C, Exp) $\sqsubseteq C$ pour toute interprétation $\mathcal{I}$. Cela contredit le principe de succès (P4) dans le cas où $C \not\sqsubseteq Exp$.

Afin d'éviter cette incompatibilité, un ajout de l'*opération épistémique* est envisagé [**DLN$^+$98 (24)**]. Nous désignons par W_1 l'ensemble des interprétations de la BC avant la révision. Pour respecter le principe (P4), la relation de subsomption $C \sqsubseteq$ DIRE(C, Exp) doit être vérifiée sur l'ensemble W_2 des interprétations où $W_2 \subsetneq W_1$. Par conséquent, la préservation d'assertion diminue le nombre des modèles possibles pour la BC. Donc, toute inférence de la BC doit être définie dans un monde fermé.

Par ailleurs, l'hypothèse de la préservation de l'ABox ne semble pas capturer l'intuition. Dans l'Exemple d'Introduction, le concept **ProdRésAuFeu** peut être défini comme suit :

ProdRésAuFeu := résister à la température du 800∘C au 1000∘C, et pendant 60'

A partir de cette définition du concept, on peut instancier de **ProdRésAuFeu** un produit p qui peut résister *au plus* à la température 800∘C pendant 60'. S'il y a une requête qui exige que la borne inférieure de la température devoie être augmentée *i.e.*

DIRE(**ProdRésAuFeu**, "résister à la température du 900∘C au 1000∘C"),

, alors le produit p n'est évidemment plus une instance de **ProdRésAuFeu** après la révision.

4.2.3. Approche syntaxique dans le langage $\mathcal{TF}$. Si nous supposons que la sémantique de la terminologie soit capturée dans les définitions de terme, la suppression d'une relation de subsomption peut se traduire en une modification des définitions de terme. En effet, nous pourrions prendre une relation de subsomption "mauvaise" comme une indication pour modifier des définitions de terme. L'opération essaie de déterminer quelle définition sera modifiée. Sinon, la modification d'une "bonne" définition amène plus de "mauvaises" définitions qui sont introduites dans la terminologie. Il est évident que le niveau approprié pour cette opération de révision est celui de définitions de *termes littéraux*. Cette idée sera exploitée dans l'approche syntaxique.

Jusqu'ici nous ne différencions pas un terme et son nom. Il existe deux interprétations possibles lorsqu'un nom est utilisé pour se référer à un objet dans un système formel :

- Référence par sens, c'est à dire que lorsqu'un nom est utilisé, il se réfère au sens courant, et la référence est immédiatement résolue sur l'entrée de sorte que les modifications suivantes ne causent aucun effet sur l'expression saisie.
- Référence par nom, c'est à dire que l'usage d'un nom a pour objectif de se référer à quelque chose qui est modifiable.

Essentiellement, l'utilisation de la référence par sens implique qu'une redéfinition d'un concept est considérée comme une définition ajoutée sans influence sur la base terminologique. Pour cette raison, elle peut éviter les problèmes liés à l'utilisation de référence par nom. Pour clarifier ces notions, on examine l'exemple suivant.

Exemple 4.2.1. (TBox "Équipe")

Équipe :=	au moins 2 *membres* et tous les *membres* sont les **Personnes**
PetiteEquipe :=	**Équipe** et au plus 5 *membres*
EquipeModerne :=	**Équipe** et le *chef* est une **Femme**

Supposons maintenant que la définition du concept **Équipe** soit changée telle qu'elle exige au moins un membre au lieu de deux membres. Dans ce cas, selon le principe de référence par nom, la définition du concept **PetiteEquipe** doit être également changée *i.e.* elle exige au moins un membre. Néanmoins, selon le principe de référence par sens, la modification de la définition du concept **Équipe** ne cause aucune répercussion sur le concept **PetiteEquipe**. En d'autres termes, le principe de référence par sens correspond au TBox déplié.

Bien que le principe de référence par sens nous permette d'avoir les implémentations simples des opérations de révision, il ne semble pas capturer suffisamment la compréhension intuitive qu'une base de connaissances terminologiques envisage de formaliser. La raison est que l'occurrence du nom du concept signifie plus que la définition courante.

Nous avons examiné l'approche sémantique qui considère une BCT comme l'ensemble des relations de subsomption possibles et modifiables. Dans cette approche, il n'y a que l'ajout de nouvelles définitions des concepts qui est autorisé (opération d'expansion). Nous ne pouvons pas définir les opérations de révision, notamment OUBLIER, sans extension du formalisme utilisé. Pour éviter ces difficultés, B. Nebel dans [**Neb90a (58)**] a proposé l'approche syntaxique qui se base sur le point de vue suivant :

> *"Dans tous les cas, les descriptions d'objet (définitions) dans une BCT sont admises comme étant déterminées et sans ambiguïté et les relations de subsomption sont dérivées de ces descriptions"* [**Neb90a (58)**]

Dans l'évolution d'une terminologie, c'est la définition d'un terme, et non des relations de subsomption entre des termes, qui devient "fausse" . De plus, une définition est fausse dans le sens où quelqu'un d'autre l'interprète différemment. Cela nous amène à définir les opérations de révision selon lesquelles des *modifications partielles* sur des définitions de termes sont effectuées au lieu de l'ajout ou de la suppression entière de l'introduction d'un terme. Cette modification syntaxique reflète certaines modifications sémantique que l'on veut envisager. La solution, qui est proposée dans [**Neb90a (58)**], s'appuie sur la notion de *composant essentiel significatif* (*concept simple*) qui constituent le sens d'un concept. Un concept simple, qui est défini pour le langage $\mathcal{TF}$, ne contient pas de conjonction dans la définition du concept. Ce langage supporte les constructeurs suivants : conjonction, restriction universelle et restriction de nombre. Nous essayons de décrire informellement les opérations de révision, qui suivent la discussion ci-dessus, pour le langage $\mathcal{TF}$.

- DIRE. Cette opération est définie comme un ajout d'un *concept simple* à l'ensemble des concepts simples qui détermine la définition d'un concept.
- OUBLIER. Cette opération est définie comme l'enlèvement d'un *concept simple* de l'ensemble des concepts simples déterminant la définition d'un concept.

Plus formellement,

- DIRE(C, Exp) = $C \sqcap Exp$ et,
- OUBLIER($C \sqcap Exp$) = C où C est un nom de concept défini dans un TBox en langage $\mathcal{TF}$, Exp est un *concept simple*.

Notons que dans l'approche syntaxique les principes de révision sont considérés sous l'hypothèse que les propositions d'une théorie sont projetées sur des concept définis d'une terminologie.

La question importante est : à quel point cette interprétation des opérations de révision satisfait les principes présentés dans la section précédente au niveau de la sémantique ?

Tout d'abord, il est nécessaire de mettre en relief le rôle des concepts simples sur lesquels les opérations de révision sont définies. Le niveau des concepts simples se situe entre la syntaxe et la sémantique du terme. Les concepts simples permettent

de s'abstraire des distinctions syntaxiques : ordre de sous-expressions, conjonctions emboîtées, etc. mais ils ne considèrent pas tous les termes, qui ont le même sens dans une terminologie, comme équivalents. Par exemple, on examine l'exemple suivant :

$$A := (B \sqcap C) \text{ et } D := (A \sqcap C)$$

Nous voyons que les ensembles des concepts simples caractérisant les expressions de définition sont différents même si ces dernières ont la même extension (l'extension est générée en substituant l'occurrence de A à sa définition). Cela s'adapte à l'hypothèse selon laquelle les *parties littérales* de la définition d'un terme (définition littérale d'un terme) devraient être une partie du sens dans *toutes terminologies possibles*. L'ensemble des concepts simples caractérisant la définition de D contient A et C dans n'importe quelle terminologie. Donc, le niveau des concepts simples formalise la notion d'équivalence des définitions de concept selon laquelle l'équivalence ne dépend pas de la terminologie. Notons que l'équivalence basée sur les parties littérales sans dépendre des terminologies s'adapte également au principe de la référence par nom. Puisque le principe (P2)- les expressions de requête de révision portant le même sens donneront le même effet sur la BCT - est respecté par l'opération DIRE car les descriptions de concept, qui sont équivalentes dans toute terminologie, ont les mêmes ensembles des concepts simples. L'opération DIRE satisfait en général le principe de succès (P4) car

$$\text{DIRE}(A, \mathit{Exp}) \sqsubseteq \mathit{Exp} \text{ pour toute } \mathit{Exp}.$$

Toutefois, l'opération DIRE peut modifier la terminologie plus que nécessaire lorsque le concept simple à ajouter est hérité d'un autre concept simple dans la description de concept. Donc, l'opération DIRE ne satisfait pas le principe de modification minimale (P5). Quant à l'opération OUBLIER, elle ne respecte pas le principe de succès (P4). Par exemple, si nous essayons de supprimer un concept simple d'une expression de définition qui hérite d'un autre concept simple dans l'expression, ce concept simple supprimé est toujours dérivable de l'expression révisée. Nous allons revenir à ce propos dans la présentation de l'approche structurelle.

4.3. Approche structurelle pour le langage $\mathcal{FLE}$

Selon la brève évaluation de l'approche syntaxique, la problématique persiste dans les opérations DIRE et OUBLIER par rapport aux principes de succès (P4) et de modification minimale (P5). Cela montre la difficulté de l'approche basée sur la syntaxe. Si nous utilisons l'approche syntaxique pour définir formellement les opérations de révision, nous pouvons arriver à exprimer les expressions de requête de révision de façon indépendante de la syntaxe par le formalisme terminologique (P1,P3). Toutefois, les opérations de révision définies par cette manière n'assurent pas les principes (P2), (P4) et (P5), appelés *principes sémantiques*. Ces derniers s'appuient sur la relation de subsomption.
La source de la difficulté réside en différence de niveaux entre la notion de concept simple et la relation de subsomption. En effet, par définition la notion de concept simple capture le sens d'un concept au niveau universel selon lequel deux concepts

sont équivalents si deux ensembles des concepts simples correspondants sont identiques pour *toute terminologie*. Par exemple, il est possible que deux concepts soient équivalents dans *une* terminologie mais ils n'ont pas le même ensemble des concepts simples. Néanmoins, l'évaluation de la satisfaction de ces principes sémantiques se base sur *une terminologie*. On considère la terminologie suivante extraite de **[Neb90a (58)]** :

Exemple 4.3.1. (Suite à l'Exemple 4.2.1)
Selon la définition de concepts simple, on a

$$S(\textbf{PetiteEquipe}) = \{\textbf{Équipe}, \text{“au plus 5 } \textit{membres}\text{”}\}.$$

Il est évident que

$$\text{OUBLIER}(\textbf{PetiteEquipe}, \text{“au moins 2 } \textit{membres}\text{”})$$

ne satisfait pas le principe de succès (P4) car le concept simple “au moins 2 *membres*” $\notin S(\textbf{PetiteEquipe})$ et le concept **PetiteEquipe** hérite toujours le concept simple “au moins 2 *membres*” du concept **Équipe**.
De plus, l'opération

$$\text{DIRE}(\textbf{PetiteEquipe}, \text{“au moins 2 } \textit{membres}\text{”})$$

ne satisfait pas le principe de modification minimale (P5) car le concept simple “au moins 2 *membres*” est inutilement ajouté à $S(\textbf{PetiteEquipe})$.

Il n'est pas pertinent de justifier cette insatisfaction par l'argument qui dit que la considération de la satisfaction des principes sémantiques doit être effectuée sans compter les relations entre les concepts, par exemple, l'héritage **[Neb90a (58)]**. Cet argument contredit la signification de la notion de concept simple.
Par ailleurs, si la relation entre les deux concepts **Équipe**, **PetiteEquipe** est préservée malgré la révision suivante :

$$\text{OUBLIER}(\textbf{PetiteEquipe}, \text{“au moins 2 } \textit{membres}\text{”}),$$

alors la révision

$$\text{OUBLIER}(\textbf{Équipe}, \text{“au moins 2 } \textit{membres}\text{”})$$

est nécessaire pour assurer le principe de succès (P4). Toutefois, cette opération modifiera la signification du concept **EquipeModerne**. Il n'apparaît pas raisonnable que la révision

$$\text{OUBLIER}(\textbf{PetiteEquipe}, \text{“au moins 2 } \textit{membres}\text{”})$$

implique une révision sur **EquipeModerne** dans le contexte de cette terminologie.

La notion de concept simple permet de capturer le mécanisme de la référence par nom c'est-à-dire que la partie inexprimable du concept dans la terminologie (TBox non-vide) est prise en compte. Par contre, la relation de subsomption est définie en se basant sur la sémantique formelle et explicite *i.e.* il n'est tenu compte que de la partie formalisée de concepts (descriptions de concept). En outre, il existe un cas où deux concepts sont équivalents dans toute terminologie mais les deux ensembles des concepts simples ne sont pas le même (un exemple sera montré dans la section suivante). Cela explique la difficulté rencontrée lorsque l'on utilise l'approche syntaxique avec la notion forte des concepts simples.

Une issue à ce problème est de redéfinir les opérations de révision et d'affaiblir la notion de concept simple pour que la modification d'une définition de concept puisse être effectuée non seulement au niveau de concept simple mais aussi au niveau de la terminologie.

4.3.1. Concept simple pour le langage $\mathcal{FLE}$. D'abord, on remarque que la définition de concept simple pour le langage $\mathcal{TF}$ ne s'adapte pas au langage $\mathcal{FLE}$ qui inclut le constructeur existentiel. En effet, on considère la terminologie suivante :

$A := \forall r.P \sqcap \exists r.Q$;
$B := \forall r.P \sqcap \exists r.P \sqcap \exists r.Q$;
$C := \forall r.P \sqcap \exists r.(P \sqcap Q)$;

On se rend compte que les concepts A, B et C sont équivalents dans toute terminologie mais les ensembles des concepts simples déterminés par la définition précédente ne sont pas le même. Notons que même si les définitions de concept sont normalisées selon les règles de normalisation présentées dans la section 2.2.4, le problème persiste à moins que l'ensemble des concepts simples ne contiennent que les concepts non subsumés réciproquement. Les concepts B, C dans l'exemple ci-dessus sont toujours équivalents après la normalisation alors que les ensembles de concept simple ne sont pas identiques. Dans ce cas, la notion de concept simple doit être définie en utilisant la relation de subsomption. Cela contredit la signification de la notion de concept simple. Pour cette raison, nous proposons la définition de concept simple affaiblie comme suit :

Définition 4.3.2. Une $\mathcal{FLE}$-description de concept est appelée *simple* si elle ne contient pas de conjonction au *top-level*. Une description de concept est appelée *plate* si elle est une conjonction de descriptions de concept simples qui ne sont pas subsumées réciproquement. Une description de concept est appelée *irréductible* si les expressions des restrictions à tout niveau sont plates.

Notons que la notion de concept simple avec la présence du constructeur de restriction universelle est définie sous la condition qui suppose la décomposition exhaustive des restrictions universelle. Par exemple, la description de concept $\forall r.(A \sqcap B)$ n'est pas un concept simple. Par conséquent, si un $\mathcal{FLE}$-concept simple Exp contient une restriction universelle au top-level, alors Exp doit être de la forme :

$Exp = \forall r....\forall r.P$ où $P \in N_C$, ou
$Exp = \forall r....\forall r.\exists r.C'$ où C' est une $\mathcal{FLE}$-description de concept.

Nous désignons par $\mathcal{S}_C$ l'ensemble des concepts simples d'un concept C dans une terminologie.

Note 4.3.3. Dans un TBox acyclique statique *i.e* sans opération de révision, nous pouvons toujours transformer ce TBox en celui déplié *équivalent* dans lequel tous les côtés droits d'une définition ne contiennent plus de nom de concept défini. L'équivalence entre les deux TBox est interprétée au niveau de la relation de subsomption basée sur les mêmes ensembles des concepts et des rôles primitifs *i.e* ils

ont les mêmes modèles. Toutefois, cela n'est pas vérifié pour un TBox sur lequel les opérations de révision sont définies. Par exemple :

Soit TBox $\mathcal{T}_1$=$\{A := P \sqcap B\,;\, B := Q \sqcap R\}$ et,
TBox déplié : $\mathcal{T}_2$=$\{A := P \sqcap Q \sqcap R\,;\, B := Q \sqcap R\}$.

Si OUBLIER($\mathcal{T}_1$, B, Q) et OUBLIER($\mathcal{T}_2$, B, Q) sont appliquées, $\mathcal{T}'_1$ et $\mathcal{T}'_2$ obtenus respectivement ne sont plus équivalents.

Si l'on choisit la référence par nom pour le système de représentation *i.e* la description de concept correspondant au côté droit d'une définition de concept ne capture pas *entièrement* le sens du concept, alors le dépliage de description de concept est exclu dans ce système. Cela explique que l'équivalence des $\mathcal{T}_1$ et $\mathcal{T}_2$ n'est pas préservée suivant une révision. Par conséquent, dans les systèmes de représentation utilisant le mécanisme de la référence par nom avec le sens fort, nous ne pouvons pas définir la notion de concept simple comme un conjonct dans la forme dépliée d'une description de concept.

En revanche, pour assurer les principes sémantiques, les définitions des opérations de révision basées sur la notion de concept simple affaiblie exige un *dépliage local*. C'est à dire que le dépliage n'est nécessaire que pour le concept à réviser. Ce dépliage limité diminue la perturbation du TBox causée par la révision. Dans l'exemple ci-dessus, le concept A n'a pas besoin d'être déplié lorsque le concept B est révisé.

4.3.2. Opérations de révision. A partir de la notion de concept simple proposée ci-dessus, les opérations de révision sont définies de telle sorte que pour l'opération OUBLIER(C, *Exp*), la modification de l'ensemble des concepts simples du concept C soit effectuée seulement si $C \sqsubseteq$ *Exp* et *Exp* $\not\sqsubseteq C$. De même, pour l'opération DIRE(C, *Exp*) la modification de l'ensemble des concepts simple du concept C *n'est pas* effectué si $C \sqsubseteq$ *Exp*. Une telle définition nous permet à la fois de respecter les principes sémantiques et de capturer également le mécanisme de la référence par nom avec le sens affaibli.
Nous revenons à l'exemple "TBox Équipe" pour savoir comment ces définitions satisfont les principes sémantiques. En effet,
si on a :

$$S(\textbf{PetiteEquipe})=\{\textbf{Équipe}, \text{“au plus 5 } \textit{membres}\text{”}\} \text{ et}$$

$$\textbf{PetiteEquipe} \sqsubseteq \text{“au moins 2 } \textit{membres}\text{”},$$

alors l'opérateur

$$\text{OUBLIER}(\textbf{PetiteEquipe}, \text{“au moins 2 } \textit{membres}\text{”})$$

est effectuée de telle sorte que l'on obtient :

$$S'(\textbf{PetiteEquipe}) = \{\text{”tous les membres sont les } \textbf{Personnes}\text{”}, \text{“au plus 5 } \textit{membres}\text{”}\}.$$

Cela signifie que le principe de succès (P4) est assuré car le concept **PetiteEquipe** n'est plus défini via le concept **Équipe**. Il semble que cela capture l'intuition. Effectivement, d'une part lors de la révision d'un concept, la modification du concept en

question est plus raisonnable que la modification d'un autre concept. D'autre part, l'opération

OUBLIER(**PetiteEquipe**, "au moins 2 *membres*")

implique à la fois le refus explicite "**PetiteEquipe** n'est plus une **Équipe**" *i.e.* OUBLIER(**PetiteEquipe**, **Équipe**) et conserve l'information "tous les membres sont les **Personnes**" dans le concept résultat.
Pour l'opération DIRE(**PetiteEquipe**,"au moins 2 *membres*"), l'ensemble S(**PetiteEquipe**) n'est pas modifié car

PetiteEquipe $\sqsubseteq$ "au moins 2 *membres*".

Cela signifie que le principe de modification minimale (P5) est également assuré car l'ajout inutile du concept simple "au moins 2 *membres*" à S(**PetiteEquipe**) est évité.

En bref, les opérations de révision pour la terminologie qui sont construites dans cette section prennent en compte les trois aspects suivants :

(1) Le principe de la référence par nom est appliqué. C'est à dire que le *dépliage global* du TBox est exclu. D'autre part, les opérations de révision modifient la définition d'un concept en ajoutant à la définition du concept ou en enlevant un concept simple. La notion de concept simple est définie dans la Définition 4.3.2.

(2) Le succès des opérations de révision devrait être assuré (P4).

(3) Les opérations de révision devraient garantir la perturbation minimale du TBox suivant une révision d'un concept (P5).

Plus précisément, les opérations de révisions doivent satisfaire les principes de révision qui sont interprétés dans le nouveau contexte comme suit :

(1) Pour l'opération OUBLIER, le principe de succès (P4) exige que OUBLIER(C, *Exp*) $\not\sqsubseteq$ *Exp* et le principe de modification minimale (P5) exige que OUBLIER(C, *Exp*) doive être une description de concept la plus spécifique par rapport à la relation de subsomption telle qu'elle subsume C.

(2) Pour l'opération DIRE, le principe de succès (P4) exige que DIRE(C, *Exp*) $\sqsubseteq$ *Exp* et DIRE(C, *Exp*) $\sqsubseteq$ C. De plus, le principe de modification minimale (P5) qui reflète le postulat de récupération (K6) exige que l'expression de requête de révision *Exp* dans OUBLIER(C, *Exp*) doive être récupérable par DIRE(*C,Exp*) *i.e* DIRE(OUBLIER(C, *Exp*), *Exp*) $\sqsubseteq$ C.

(3) Quant à la notion du concept simple, les opérations de révision ne manipulent plus les ensembles de concepts simples par l'union ou la soustraction des ensembles. Une redéfinition d'un concept (DIRE) peut causer un changement entier de l'ensemble des concepts simples ou une modification de certains concepts simples (OUBLIER). Par conséquent, le rôle de la notion de concept simple dans l'approche structurelle est moins important.

Dans le paragraphe suivant, nous présentons les définitions formelles des opérations de révision pour les TBox qui utilisent les langages L.D permettant de définir la notion de concept simple.

Dans le reste de ce chapitre, une $\mathcal{L}$-description de concept normalisée est considérée comme une description de concept normalisée par les règles dans la Définition 2.2.4.

Définition 4.3.4. (SEMI-OUBLIER) Soit C une $\mathcal{L}$-description de concept normalisées. Soit *Exp* une $\mathcal{L}$-description de concept simple et $Exp \not\equiv \top, \bot$.

(1) Si $C \equiv \top$, SEMI-OUBLIER(C, *Exp*) := $\top$,

(2) Si $C \not\sqsubseteq Exp$, SEMI-OUBLIER(C, *Exp*) := C,

(3) Si $C \sqsubseteq Exp$, alors SEMI-OUBLIER(C, *Exp*) est une $\mathcal{L}$-description de concept qui a les propriétés suivantes :

   (a) $C \sqsubseteq$ SEMI-OUBLIER(C, *Exp*)

   (b) SEMI-OUBLIER(C, *Exp*) $\not\sqsubseteq$ *Exp*

   (c) Si D est une $\mathcal{L}$-description de concept telle que $C \sqsubseteq D$ et $D \not\sqsubseteq Exp$, alors $D \not\sqsubseteq$ SEMI-OUBLIER(C, *Exp*).

Maintenant, nous introduisons la définition de l'opération OUBLIER. La différence unique entre les deux définitions est que la condition 3. est renforcée dans la définition OUBLIER.

Définition 4.3.5. (OUBLIER) Soit C une $\mathcal{L}$-description de concept normalisée. Soit *Exp* une $\mathcal{L}$-description de concept simple et $Exp \not\equiv \top, \bot$.

(1) Si $C \equiv \top$, OUBLIER(C, *Exp*) := $\top$,

(2) Si $C \not\sqsubseteq Exp$, OUBLIER(C, *Exp*) := C,

(3) Si $C \sqsubseteq Exp$, alors OUBLIER(C, *Exp*) est une $\mathcal{L}$-description de concept qui a les propriétés suivantes :

   (a) $C \sqsubseteq$ OUBLIER(C, *Exp*),

   (b) OUBLIER(C, *Exp*) $\not\sqsubseteq$ *Exp*,

   (c) Si D est une $\mathcal{L}$-description de concept telle que $C \sqsubseteq D$ et $D \not\sqsubseteq Exp$, alors OUBLIER(C, *Exp*) $\sqsubseteq D$.

Les conditions 3.(a) et 3.(b) dans la Définition 4.3.5 assurent le principe (P4, *succès*) alors que la condition 3.(c) implique le principe (P5, *modification minimale*). Par ailleurs, OUBLIER n'existe pas dans certains langages L.D mais SEMI-OUBLIER toujours existe.

Remarque 4.3.6. Grâce à la condition $Exp \not\equiv \top, \bot$, on a : $C \sqsubseteq$ OUBLIER(C, *Exp*) pour tout C. Cependant, si $C \equiv \bot$ alors l'opération OUBLIER(C, *Exp*) dans la Définition 4.3.5 ne peut exister, car $\bot \sqsubseteq D$ pour toute description de concept D.

L'exemple suivant illustre les définitions ci-dessus.

Exemple 4.3.7.

(1) Soient C, Exp des $\mathcal{FLE}$-descriptions de concept :
$C := \forall r.(A_1 \sqcap A_2) \sqcap \exists r.(A_1 \sqcap A_2 \sqcap A_3)$, $Exp := \exists r.(A_1 \sqcap A_2)$ où $A_i \in N_C$ et $r \in N_R$.
On a : $C \sqsubseteq Exp$, et donc, SEMI-OUBLIER(*C, Exp*) $\in \{\forall r.A_1 \sqcap \exists r.(A_1 \sqcap A_3)$, $\forall r.A_2 \sqcap \exists r.(A_2 \sqcap A_3)$, $\exists r.(A_2 \sqcap A_3) \sqcap \exists r.(A_1 \sqcap A_3)\}$.
Effectivement, on a : $C \sqsubseteq \forall r.A_1 \sqcap \exists r.(A_1 \sqcap A_3)$ et $\forall r.A_1 \sqcap \exists r.(A_1 \sqcap A_3) \not\sqsubseteq Exp$.
De plus, s'il existe C' telle que $C \sqsubseteq C'$ et $C' \not\sqsubseteq Exp$, alors $C' \equiv \forall r.var(C') \sqcap \bigsqcap_{E' \in ex(C')} \exists r.E'$ où $var(C')$ et $ex(C')$ sont définis dans la Définition 2.2.12, $var(C')$ contient soit A_1, soit A_2, soit une conjonction de A_1, A_2 (par le Théorème 2.2.5, cela est déduit de $C \sqsubseteq C'$) . De plus, puisque $C' \not\sqsubseteq Exp$, alors E' ne contient pas tous les deux primitifs A_1, A_2 pour toute $E' \in ex(C')$. Par la Définition SEMI-OUBLIER, on obtient : SEMI-OUBLIER(*C, Exp*) $\equiv \forall r.A_1 \sqcap \exists r.(A_1 \sqcap A_3)$ (notons que toutes les descriptions de concept considérées sont normalisées). De même, on peut montrer que $\forall r.A_2 \sqcap \exists r.(A_2 \sqcap A_3)$ et $\exists r.(A_2 \sqcap A_3) \sqcap \exists r.(A_1 \sqcap A_3)$ satisfont également la Définition SEMI-OUBLIER.

(2) Soient C, Exp des $\mathcal{FLE}$-descriptions de concept :
$C := \forall r.B \sqcap \exists r.(A \sqcap B)$, $Exp := \exists r.A$ où $A, B \in N_C$ et $r \in N_R$.
On a : $C \sqsubseteq Exp$, et donc, OUBLIER(*C, Exp*) $= \forall r.B \sqcap \exists r.B$. Effectivement, on obtient : $C \sqsubseteq \forall r.B \sqcap \exists r.B$ et $\forall r.B \sqcap \exists r.B \not\sqsubseteq \exists r.A$. De plus, s'il existe C' telle que $C \sqsubseteq C'$ et $C' \not\sqsubseteq Exp$, alors $\forall r.B \sqcap \exists r.B \sqsubseteq C'$ (Théorème 2.2.5).

Remarque 4.3.8. Nous montrons que les opérations décrites ci-dessus sont bien définies pour certains langages L.D et considérons quelques cas particuliers.

(1) Il est évident que pour un langage L.D $\mathcal{L}$ où OUBLIER(*C*, *Exp*) existe, alors SEMI-OUBLIER(*C*, *Exp*) existe également et
SEMI-OUBLIER(*C*, *Exp*) $\equiv$ OUBLIER(*C*, *Exp*) car OUBLIER(*C*, *Exp*) $\sqsubseteq D$ déduit $D \not\sqsubseteq$ OUBLIER(*C*, *Exp*).

(2) Soient *C*, *Exp* des $\mathcal{L}$-descriptions de concept normalisées, *Exp* est une description de concept simple (il n'existe pas de conjonction au top-level). On peut démontrer que pour le langage $\mathcal{L} = \mathcal{EL}$, OUBLIER(*C*, *Exp*) est *unique* (s'il existe) modulo équivalence. En effet, supposons qu'il y ait des $\mathcal{EL}$-descriptions de concept C_1, C_2 telles que $C \sqsubseteq C_1$, $C_1 \not\sqsubseteq Exp$ et $C \sqsubseteq C_2$, $C_2 \not\sqsubseteq Exp$. Alors, on a : $C \sqsubseteq C_1 \sqcap C_2$. Puisque $C_1 \not\sqsubseteq Exp$, $C_2 \not\sqsubseteq Exp$ et *Exp* est une $\mathcal{EL}$-description de concept simple (*Exp* ne contient qu'une restriction existentielle ou qu'un concept primitif au top-level), alors par le Théorème 2.2.5, on a : $C_1 \sqcap C_2 \not\sqsubseteq Exp$. Par définition de l'opération OUBLIER, on obtient : $C_1 \sqsubseteq C_1 \sqcap C_2$ et donc $C_1 \equiv C_2$. De plus, à partir de $C \sqsubseteq Exp$, $C \sqsubseteq$ OUBLIER(*C*, *Exp*) et OUBLIER(*C*, *Exp*) $\not\sqsubseteq Exp$, on obtient également $C \sqsubset$ OUBLIER(*C*, *Exp*).

(3) Soient C_1, C_2, *Exp* des $\mathcal{L}$-descriptions de concept normalisées, *Exp* est une description de concept simple et $C_1 \sqsubseteq C_2$. Supposons qu'il existe OUBLIER(C_1, *Exp*) et OUBLIER(C_2, *Exp*). On a :

$$\text{OUBLIER}(C_1, \mathit{Exp}) \sqsubseteq \text{OUBLIER}(C_2, \mathit{Exp}).$$

En effet, puisque $C_1 \sqsubseteq C_2 \sqsubseteq \text{OUBLIER}(C_2, \mathit{Exp})$ et $\text{OUBLIER}(C_2, \mathit{Exp}) \not\sqsubseteq \mathit{Exp}$, alors par la définition de l'opération OUBLIER, on a : $\text{OUBLIER}(C_1, \mathit{Exp}) \sqsubseteq \text{OUBLIER}(C_2, \mathit{Exp})$.

(4) Pour le langage $\mathcal{FLE}$, SEMI-OUBLIER(C, *Exp*) peut ne pas être unique. Donc, OUBLIER(C, *Exp*) peut ne pas exister (Exemple 4.3.7).
Cependant, si C est une restriction existentielle *i.e.* $C = \exists r.C'$ où C' est une $\mathcal{FLE}$ -description de concept, alors SEMI-OUBLIER(C, *Exp*) est unique (s'il existe) et SEMI-OUBLIER(C, *Exp*) $\equiv$ OUBLIER(C,*Exp*). Cette affirmation est déduite de l'argument suivant : si C_1, C_2 sont des SEMI-OUBLIER du concept C, alors $C \sqsubseteq C_1$, $C \sqsubseteq C_2$, $C_1 \not\sqsubseteq \mathit{Exp}$, $C_2 \not\sqsubseteq \mathit{Exp}$, et donc $C \sqsubseteq C_1 \sqcap C_2$. Puisque C est une restriction existentielle, alors C_1, C_2 sont également les restrictions existentielles et donc $C_1 \sqcap C_2 \not\sqsubseteq \mathit{Exp}$ (le Théorème 2.2.5). Par la définition de l'opération SEMI-OUBLIER, on obtient : $C_1 \sqcap C_2 \not\sqsubset C_1$. Puisque $C_1 \sqcap C_2 \not\sqsubseteq C_1$ est impossible, donc $C_1 \equiv C_2$. De même argument, on peut démontrer que OUBLIER(C, *Exp*) est unique (s'il existe). Selon 1. de cette remarque, on obtient : SEMI-OUBLIER(C, *Exp*) $\equiv$ OUBLIER(C, *Exp*).

Calcul de OUBLIER dans $\mathcal{EL}$. Nous nous rendons compte que les définitions de l'opération OUBLIER ne nous permet pas de calculer la description de concept OUBLIER. Le calcul de OUBLIER exige une procédure qui construit une description de concept satisfaisant les conditions dans la Définition 4.3.5. Dans ce cas, l'approche pour l'algorithme de construction montre à nouveau son importance. Par la suite, nous montrons que OUBLIER existe toujours dans le langage $\mathcal{EL}$ et présentons un algorithme pour calculer la description de concept OUBLIER dans ce langage. Cette démarche est décisive pour traiter les langages contenant le constructeur de restriction existentielle.
Sans perte de généralité, nous supposons que l'ensemble de rôles N_R des langages L.D considérés dans le reste du chapitre comporte seulement un élément *i.e.* $N_R = \{r\}$. Tous les résultats obtenus sont encore valables pour l'ensemble arbitraire N_R.

Algorithme 4.3.9. (*MINUS pour $\mathcal{EL}$*) *Soient* C, D *des $\mathcal{EL}$-descriptions de concept normalisées,* D *est une description de concept simple. La description de concept MINUS(C,D) est calculé comme une conjonction des restes des arbres de description $\mathcal{G}_C$ après une* suppression minimale *de l'image $\varphi(\mathcal{G}_D)$ dans $\mathcal{G}_C$ pour tout homomorphisme φ de $\mathcal{G}_D$ dans $\mathcal{G}_C$. Plus formellement, soient $\mathcal{G}$ =(N_C, E_C, v_0, l_C), $\mathcal{H}$ =(N_D, E_D, w_0, l_D) et $(\mathcal{G} - \mathcal{H})$ = (N_{C-D}, E_{C-D}, u_0, l_{C-D}). L'arbre $(\mathcal{G} - \mathcal{H})$ est calculé par récurrence sur la profondeur de $\mathcal{G}$ comme suit :*

***Entrée** : $\mathcal{G}$, $\mathcal{H}$*
***Sortie** : $(\mathcal{G} - \mathcal{H}) = \mathcal{G}_{MINUS(C,D)}$*
Si $C_{\mathcal{G}} \not\sqsubseteq C_{\mathcal{H}}$, alors $(\mathcal{G} - \mathcal{H}) := \mathcal{G}$. Sinon,

(1) *Si $|\mathcal{G}| = 0$, le nœud (v_0, w_0) étiqueté par $l_C(v_0) \setminus l_D(w_0)$ est la racine de l'arbre de $(\mathcal{G} - \mathcal{H})$.*

(2) *Si w_0 n'a pas de successeur, $(\mathcal{G} - \mathcal{H})$ est une copie de $\mathcal{G}$ avec pour racine le nœud (v_0, w_0) étiqueté par $l_C(v_0) \setminus l_D(w_0)$.*

(3) *Si v est le successeur unique de v_0,*

(a) *Pour chaque r-successeur v' de v dans $\mathcal{G}$ où $l_D(w) \not\subseteq l_C(v)$, on obtient un r-successeur (v, w) de (v_0, w_0) avec $l(v, w) = l_C(v)$ dans $(\mathcal{G} - \mathcal{H})$, qui est la racine d'une copie du sous-arbre $\mathcal{G}(v)$.*

(b) *Pour chaque r-successeur v' de v_0 dans $\mathcal{G}$ où $l_D(w) \subseteq l_C(v)$ et chaque ensemble d'étiquette $l_C(v) \setminus \{P\}$ où $P \in l_C(v) \cap l_D(w)$, on obtient un r-successeur (v, w) de (v_0, w_0) avec $l(v, w) = l_C(v) \setminus \{P\}$ dans $(\mathcal{G} - \mathcal{H})$, qui est la racine d'une copie du sous-arbre $\mathcal{G}(v)$.*

(c) *Pour chaque r-successeur w' de w dans $\mathcal{H}$, on obtient un r-successeur (v, w) de (v_0, w_0) avec $l(v, w) = l_C(v)$ dans $(\mathcal{G} - \mathcal{H})$, qui est la racine des arbres $(r.\mathcal{G}(v') - r.\mathcal{H}(w'))$ pour tout r-successeur v' de v.*

(4) *S'il existe plusieurs successeurs v de v_0,*
Pour chaque r-successeur v de v_0 dans $\mathcal{G}$, on calcule $(r.\mathcal{G}(v') - r.\mathcal{H}(w'))$ par l'étape 3. de l'algorithme et on obtient les r-successeurs (v, w) correspondant de (v_0, w_0) dans $(\mathcal{G} - \mathcal{H})$.

Exemple. Soient $C := \exists r.(A \sqcap B) \sqcap \exists r.(A \sqcap C)$, $Exp := \exists r.A$.

– Par l'étape 3.(b) de l'Algorithme 4.3.9, on a :
MINUS(C, Exp) = MINUS($\exists r.(A \sqcap B)$, Exp) $\sqcap$ MINUS($\exists r.(A \sqcap C)$, Exp).
– Par l'étape 3.(b) de l'Algorithme 4.3.9, on a :
MINUS($\exists r.(A \sqcap B)$, Exp) = $\exists r.B$ et MINUS($\exists r.(A \sqcap C)$, Exp) = $\exists r.C$
– Par conséquent, on obtient : MINUS(C, Exp) = $\exists r.B \sqcap \exists r.C$.

Remarque 4.3.10. Selon l'Algorithme 4.3.9, la taille de MINUS(C,D) est un produit des tailles de C et de D. Soit $m_C^i = max\{r(u^i) \times |l(v^i)|\}$ où $|l(v^i)|$ est la taille de l'ensemble d'étiquettes du nœud $v^i \in N_C$ au niveau i-ème de l'arbre $\mathcal{G}_C$, $r(u^i)$ est le nombre de successeurs du nœud au niveau i-ème $u^i \in N_C$. Le nombre de nœuds au niveau i-ème de l'arbre de description $\mathcal{G}_{MINUS(C,D)}$ est borné par $(m_C^{i-1} \times m_D^{i-1})$ $\times$ $(m_D^i \times m_D^i)$

Maintenant, nous formulons et montrons un théorème qui établit l'équivalence entre la OUBLIER définie dans la Définition 4.3.5 et la description de concept calculée par l'Algorithme 4.3.9.

Théorème 4.3.11. *Soient* C, D des $\mathcal{EL}$*-descriptions de concept normalisées et* D *est une description de concept simple. Alors OUBLIER(C, D) toujours existe et il y a un algorithme polynômial pour le calculer.*

DÉMONSTRATION. Puisque OUBLIER(C, D) = C si $C \not\sqsubseteq D$, alors on peut supposer que $C \sqsubseteq D$. Il suffit de démontrer que OUBLIER(C, D) $\equiv$ MINUS(C, D). La preuve est effectuée par récurrence sur la profondeur de l'arbre de description de C.
Soient $\mathcal{G} = (N_C, E_C, v_0, l_C)$, $\mathcal{H} = (N_D, E_D, w_0, l_D)$ et $(\mathcal{G} - \mathcal{H}) = G_{MINUS(C,D)} = (N_{C-D}, E_{C-D}, u_0, l_{C-D})$.

– $|C| = 0$. Puisque D est une description simple, alors $l_D(w_0) = \emptyset$ si $|D| > 0$ et $l_D(w_0) = \{P\}$ si $|D| = 0$. On doit démontrer les propriétés suivantes de MINUS.

(1) $C \sqsubseteq$ MINUS(C, D). Puisque $C \sqsubseteq D$, alors $|D| \leq |C|$ et $l_D(w_0) \subseteq l_C(v_0)$. Donc, $|D| = |C| = 0$ et $l_{C-D}(u_0) = l_C(v_0) \setminus l_D(w_0) \subseteq l_C(v_0)$.

(2) MINUS(C, D) $\not\sqsubseteq D$. De même, on a : $|D| = |C| =$|A| $= 0$ et $l_D(w_0) \not\subseteq l_C(v_0) \setminus l_D(w_0) = l_{C-D}(u_0)$.

(3) Si A est une $\mathcal{EL}$-description de concept telle que $C \sqsubseteq A$ et $A \not\sqsubseteq D$, alors MINUS(C, D) $\sqsubseteq A$. Soit $\mathcal{G}_A = (N_A, E_A, x_0, l_A)$. De même, on a : $|D| = |C| = 0$ et $l_A(x_0) \subseteq l_C(v_0)$, $l_{C-D}(u_0) = l_C(v_0) \setminus l_D(w_0)$, et $l_D(w_0) \cap l_A(x_0) = \emptyset$, donc $l_A(x_0) \subseteq l_{C-D}(u_0)$.

– $|C| > 0$. On doit démontrer les propriétés suivantes de MINUS.

(1) $C \sqsubseteq$ MINUS(C, D). En effet, par l'Algorithme 4.3.9, on a : $l_{C-D}(u_0) \subseteq l_C(v_0)$.
Si le successeur v de v_0 est le r-successeur unique, alors soient v le r-successeur de v_0, w le r-successeur de w_0. Si (v, w) correspondant aux sous-arbres créés par l'étape 3.(a) de l'Algorithme 4.3.9, alors $l_{C-D}(v, w) = l_C(v)$ et $(\mathcal{G} - \mathcal{H})(v, w)$ est une copie du sous-arbre $\mathcal{G}(v)$. Donc, $C_{\mathcal{G}(v)} \sqsubseteq C_{(\mathcal{G}-\mathcal{H})(v,w)}$. Si (v, w) correspondant aux sous-arbres créés par l'étape 3.(b) de l'Algorithme 4.3.9, alors $l_{C-D}(v, w) \subseteq l_C(v)$ et $(\mathcal{G} - \mathcal{H})(v, w)$ est une copie du sous-arbre $\mathcal{G}(v)$. Donc, $C_{\mathcal{G}(v)} \sqsubseteq C_{(\mathcal{G}-\mathcal{H})(v,w)}$. Si (v, w) correspondant aux sous-arbres créés par l'étape 3.(c) de l'Algorithme 4.3.9, alors par hypothèse de récurrence, on obtient : $C_{\mathcal{G}(v)} = C_{r.\mathcal{G}(v')} \sqsubseteq$ MINUS($C_{r.\mathcal{G}(v')} - C_{r.\mathcal{H}(w')}$).
Supposons que v_0 a plusieurs r-successeurs v. Selon l'Algorithme 4.3.9, MINUS(C, D) se compose des MINUS(C_1, D),..., MINUS(C_n, D) où $C = C_1 \sqcap ... \sqcap C_n$, chaque C_i correspond à un r-successeur. D'après la démonstration ci-dessus, $C_i \sqsubseteq$ MINUS(C_i, D) pour tout $i \in \{1, ..., n\}$. Par conséquent, $C \sqsubseteq$ MINUS(C, D).

(2) MINUS(C, D) $\not\sqsubseteq D$.
Si le successeur v de v_0 est le r-successeur unique , alors soient v le r-successeur de v_0, w le r-successeur de w_0. Si (v, w) correspondant aux sous-arbres créés par l'étape 3.(a) de l'Algorithme 4.3.9, alors $l_{C-D}(v, w) = l_D(v) \not\subseteq l_C(w)$ et $(\mathcal{G} - \mathcal{H})(v, w)$ est une copie du sous-arbre $\mathcal{G}(v)$. Donc, $C_{(\mathcal{G}-\mathcal{H})(v,w)} \not\sqsubseteq D$. Si (v, w) correspondant aux sous-arbres créés par l'étape 3.(b) de l'Algorithme 4.3.9, alors $l_{C-D}(v, w) \not\subseteq l_D(w)$ et $(\mathcal{G} - \mathcal{H})(v, w)$ est une copie du sous-arbre $\mathcal{G}(v)$. Donc, $C_{(\mathcal{G}-\mathcal{H})(v,w)} \not\sqsubseteq D$. Si (v, w) correspondant aux sous-arbres créés par l'étape 3.(c) de l'Algorithme 4.3.9, alors par hypothèse de récurrence, on a : $C_{MINUS(r.\mathcal{G}(v'),r.\mathcal{H}(w'))} \not\sqsubseteq C_{r.\mathcal{H}(w')}$. Cela implique que $C_{(\mathcal{G}-\mathcal{H})(v,w)} \not\sqsubseteq D$.
Supposons que v_0 a plusieurs r-successeurs v. Selon l'Algorithme 4.3.9, MINUS(C, D) se compose des MINUS(C_1, D),..., MINUS(C_n, D) où $C = C_1 \sqcap ... \sqcap C_n$, chaque C_i correspond à un r-successeur. D'après la démonstration ci-dessus, MINUS(C_i, D) $\not\sqsubseteq D$ pour tout $i \in$\{1,...,n\}.

Puisque C est une $\mathcal{EL}$-description de concept et D est une $\mathcal{EL}$-description de concept simple, alors MINUS(C, D) $\not\sqsubseteq D$ par le Théorème 2.2.5.

(3) Montrons que si A est une $\mathcal{EL}$-description de concept normalisée telle que $C \sqsubseteq A$ et $A \not\sqsubseteq D$, alors MINUS(C, D) $\sqsubseteq A$. Soient $\mathcal{F} = (N_A, E_A, x_0, l_A)$, w est le r-successeur de w_0.
Si $l_D(w) \not\subseteq l_A(x)$ ou $l_A(x) \subset l_C(v)$ où x est un r-successeur de x_0 , alors d'après les étapes 3.(a) et 3.(b) de l'Algorithme 4.3.9, on a : $C_{(\mathcal{G}-\mathcal{H})(v,w)} \sqsubseteq C_{\mathcal{F}(x)}$. Supposons que $l_D(w) \subseteq l_A(x)$ et $l_A(x) = l_C(v)$. Puisque $C \sqsubseteq A$, alors pour chaque r-successeur x de x_0, il existe un r-successeur v de v_0 tel que $l_A(x) \subseteq l_C(v)$ et $C_{\mathcal{G}(v)} \sqsubseteq C_{\mathcal{F}(x)}$. D'autre part, puisque $A \not\sqsubseteq D$, pour chaque r-successeur x de x_0, on a : $C_{\mathcal{F}(x)} \not\sqsubseteq C_{\mathcal{H}(w)}$. Par hypothèse de récurrence et l'étape 3.(c) de l'Algorithme 4.3.9, on obtient : $C_{(\mathcal{G}-\mathcal{H})(v',w')} \sqsubseteq C_{\mathcal{F}(x')}$ pour chaque r-successeur x' de x où v' est un r-successeur de v tel que $C_{\mathcal{G}(v')} \sqsubseteq C_{\mathcal{F}(x')}$, w' est un r-successeur de w tel que $C_{\mathcal{F}(x')} \not\sqsubseteq C_{\mathcal{H}(w')}$ pour tout r-successeur x' de x (puisque $C_{\mathcal{F}(x)} \not\sqsubseteq C_{\mathcal{H}(w)}$, alors un tel w' existe), et (v', w') est un r-successeur de (v, w). Cela implique que pour chaque r-successeur x de x_0, il existe un r-successeur (v, w) de (v_0, w_0) tel que $l_A(x) \subseteq l_{C-D}(v, w)$ et $C_{(\mathcal{G}-\mathcal{H})(v,w)} \sqsubseteq C_{\mathcal{F}(x)}$. Par conséquent, on obtient : MINUS(C, D) $\sqsubseteq A$.

Maintenant, nous montrons que la taille de la description de concept MINUS(C, D) obtenue est polynômiale en fonction de la taille de C et de D. En effet, Selon l'étape 3. de l'Algorithme 4.3.9, le nombre de nœuds de l'arbre de description $\mathcal{G}_{MINUS(C,D)}$ est borné par $|V_C| \times |V_D| \times max(|l_C|, |l_D|)$. □

Calcul de OUBLIER dans $\mathcal{FLE}$. Comme mentionné dans la Remarque 4.3.8, l'unicité de SEMI-OUBLIER n'est plus assurée lorsque le constructeur de restriction universelle est présent dans les langages L.D en question. Cependant, il existe toujours SEMI-OUBLIER dans ces langages. Notons que la notion de concept simple avec la présence du constructeur de restriction universelle est définie sous la condition qui suppose la décomposition exhaustive des restrictions universelle. Par exemple, la description de concept $\forall r.(A \sqcap B)$ n'est pas un concept simple. Par conséquent, si un $\mathcal{FLE}$-concept simple Exp contient une restriction universelle au top-level, alors Exp doit être de la forme :

$Exp = \forall r....\forall r.P$ où $P \in N_C$, ou
$Exp = \forall r....\forall r.\exists r.C'$ où C' est une $\mathcal{FLE}$-description de concept.

La proposition suivante présente une restriction sur les $\mathcal{FLE}$-descriptions de concept pour que SEMI-OUBLIER soit unique.

Proposition 4.3.12. *Soient C, Exp des $\mathcal{FLE}$-descriptions de concept, Exp est une description de concept simple et $C \sqsubseteq Exp$. De plus, C est de la forme $C := \forall r.C' \sqcap D$ où D ne contient pas de restriction universelle au top-level et soit C' est un nom de concept ($C' \in N_C$) soit C' est une restriction existentielle (i.e. $C' := \exists r.C''$ où C'' est une $\mathcal{FLE}$-description de concept). Alors,*

SEMI-OUBLIER(C, Exp) est unique s'il existe, et

$$SEMI\text{-}OUBLIER(C,\ Exp) \equiv OUBLIER(C, Exp).$$

DÉMONSTRATION. Soient C_1, C_2 des SEMI-OUBLIER du C, alors $C \sqsubseteq C_1$, $C \sqsubseteq C_2$, $C_1 \not\sqsubseteq Exp$, $C_2 \not\sqsubseteq Exp$, et donc $C \sqsubseteq C_1 \sqcap C_2$. Si Exp est un nom de concept, la proposition est évidente. Supposons que Exp est une restriction universelle. Dans ce cas, on a : $val(C) = C' \neq \top$, $val(C_1) \not\sqsubseteq val(Exp)$ et $val(C_2) \not\sqsubseteq val(Exp)$. De plus, puisque $val(C)$ est une restriction existentielle et $val(C) \sqsubseteq val(C_1), val(C_2)$, alors $val(C_1) \sqcap val(C_2) \not\sqsubseteq val(Exp)$. On obtient : $C_1 \sqcap C_2 \not\sqsubseteq Exp$.

Supposons que Exp soit une restriction existentielle. Par le Théorème 2.2.13, on a : $val(C_1) \sqcap D' \not\sqsubseteq ex(Exp)$ pour toute $D' \in ex(C_1)$ et $val(C_2) \sqcap D'' \not\sqsubseteq val(Exp)$ pour toute $D'' \in ex(C_2)$ (les notations val, ex sont introduites dans la Définition 2.2.12). Puisque $val(C_1)$ et $val(C_2)$ ne contiennent que des restrictions existentielles au top-level (car $val(C)$ est une restriction existentielle et $val(C) \sqsubseteq val(C_1), val(C_2)$), alors

$$\bigsqcap_{D \in \{ex(C_1) \cup ex(C_2)\}} D \sqcap val(C_1) \sqcap val(C_2) \not\sqsubseteq ex(Exp).$$

Par conséquent, on obtient $C_1 \sqcap C_2 \not\sqsubseteq Exp$.
Par ailleurs, la définition de l'opération SEMI-OUBLIER implique que $C_1 \sqcap C_2 \not\sqsubset C_1$. Puisque $C_1 \sqcap C_2 \not\sqsubseteq C_1$ est impossible, donc $C_1 \equiv C_2$. De même argument, on peut démontrer que OUBLIER(C, Exp) est unique (s'il existe). Selon 1. de la Remarque 4.3.8, on obtient : SEMI-OUBLIER(C, Exp) $\equiv$ OUBLIER(C, Exp). □

On désigne par $\mathcal{FLE}^-$ le sous-langage du $\mathcal{FLE}$ comportant les descriptions de concept C qui sont déterminées dans la Proposition 4.3.12.
La Remarque 4.3.8 montre que OUBLIER(*C, Exp*) pour le langage $\mathcal{FLE}$ n'existe pas toujours. De plus, elle montre également SEMI-OUBLIER(*C, Exp*) existe et peut ne pas être unique. Cependant, si l'ensemble de *toutes* les descriptions de concept SEMI-OUBLIER(*C, Exp*) est déterminé, alors OUBLIER(*C, Exp*) peut être approximée en quelque sorte par cet ensemble. Pour cet objectif, nous commençons par proposer une procédure qui calcule l'ensemble {SEMI-OUBLIER(*C, Exp*)} où C est une $\mathcal{FLE}$-description qui contient soit une restriction existentielle, soit une restriction universelle (*Exp* reste toujours une $\mathcal{FLE}$-description de concept simple).

Algorithme 4.3.13. (*MINUS pour les concepts simples*) *Soient* C, D *des $\mathcal{FLE}$-descriptions de concept normalisées et C est une description telle que $C = \alpha.C'$ où $\alpha \in \{r, \forall r\}$, C' est une $\mathcal{FLE}$-description de concept. De plus, supposons que* D *soit une description de concept simple. On désigne $\mathcal{G}$ =(N_C, E_C, v_0, l_C) et $\mathcal{H}$ =(N_D, E_D, w_0, l_D). L'ensemble des descriptions de concept MINUS(C, D) est calculé par récurrence sur la profondeur de $\mathcal{G}$ comme suit :*

***Entrée** : $\mathcal{G}$, $\mathcal{H}$*
***Sortie** : Ensemble des arbres $T = \{\mathcal{G} - \mathcal{H}\} = \{\mathcal{G}_{C'} |\ C' \in MINUS(C, D)\}$*
Si $C_{\mathcal{G}} \not\sqsubseteq C_{\mathcal{H}}$, alors $T := \mathcal{G}$. Sinon,

(1) *$T := \emptyset$. Si $|\mathcal{G}| = 0$, et donc $|l_C(v_0)| = |l_D(w_0)| = 1$,*
T comporte un arbre qui possède un seul nœud et $l(v_0, w_0) = l_C(v_0) \setminus l_D(w_0)$.

(2) *Si* $|\mathcal{G}| > 0$ *et* $|\mathcal{H}| = 0$,
T comporte un seul arbre qui est une copie de $\mathcal{G}$ *où le nœud* (v_0, w_0) *étiqueté par* $l_C(v_0) \setminus l_D(w_0)$.

(3) *Sinon, soient* v, w *les* α*-successeurs respectivement de* v_0 *et de* w_0,

(a) *Pour chaque* $P \in (l_C(v) \cap l_D(w)) \neq \emptyset$, *un arbre t est créé où* t_0 *est la racine de t,* $l_t(t_0) = l_C(v_0)$ $(l_D(w_0) = \emptyset)$. *De plus, le nœud* (v, w) *est un* α*-successeur de* t_0 *avec* $l_t(v, w) = l_C(v) \setminus \{P\}$. *Ce nœud est aussi la racine d'une copie du sous-arbre* $\mathcal{G}(v)$ *de* $\mathcal{G}$. *On obtient :* $T := T \cup \{t\}$.

(b) *Pour chaque arbre* $t' \in \{\mathcal{G}_{C'} |\ C' \in MINUS(\forall r.C_{\mathcal{G}(v')}, \forall r.C_{\mathcal{H}(w')})\}$ *où* v', w' *sont les* $\forall r$*-successeurs respectivement de v et de w, un arbre t est créé où* t_0 *est la racine de t tel que* (v, w) *soit un* α*-successeur de* t_0, $l_t(v, w) = l_C(v)$ *avec* $l_t(t_0) = l_C(v_0)$, *et soit la racine de l'arbre* t' *et des copies des sous-arbres* $r.\mathcal{G}(v'')$ *où* v'' *est un r-successeur de v. On obtient :* $T := T \cup \{t\}$.

(c) *Pour chaque r-successeur* w'' *de w dans* $\mathcal{H}$:
Soit U l'ensemble des r-successeurs v^i *de v où* $l_D(w'') \subseteq l_C(v^i)$. *Pour chaque sous-ensemble* $V = \{v^1, ..., v^m\} \subseteq U$, *on calcule récursivement les ensembles des arbres* $\{\mathcal{G}_{C'} |\ C' \in MINUS(C_{r.\mathcal{G}(v^i)}, C_{r.\mathcal{H}(w'')})\}$. *On désigne par* $T_{v^i} = \{r.F^1_{v^i}, ..., r.F^{k_i}_{v^i}\}$ *l'ensemble des sous-arbres* $\{\mathcal{G}_{C'} |\ C' \in MINUS(C_{r.\mathcal{G}(v^i)}, C_{r.\mathcal{H}(w'')})\}$ *pour tout* $i \in \{1, ..., m\}$.
Maintenant, pour chaque $(l_1, ..., l_m) \in \{1..k_1\} \times ... \times \{1..k_m\}$, *un arbre t est créé où* t_0 *est la racine de t tel que* (v, w) *soit un r-successeur de* t_0 *avec* $l_t(v, w) = l_C(v)$, *et soit la racine des sous-arbres suivants :*

(i) $r.\mathcal{G}(v'')$ *où* v'' *est un r-successeur de v tel que* $l_D(w'') \not\subseteq l_C(v'')$.

(ii) $r.F^{l_1}_{v^1}, ..., r.F^{l_m}_{v^m}$ *où* $(r.F^{l_1}_{v^1}, ..., r.F^{l_m}_{v^m}) \in (T_{v^1} \times ... \times T_{v^m})$

(iii) $\forall r.lcs\{\ \{C_{F^{l_j}_{v^j}} \mid 1 \leq j \leq m\ \} \cup \{C_{\mathcal{G}(v')}\}\ \}$ où v' *est le* $\forall r$*-successeur de v.*

On obtient $T := T \cup \{t\}$. *Notons que pour chaque r-successeur* w'' *de w, chaque ensemble V et chaque* $(l_1, ..., l_m) \in \{1..k_1\} \times ... \times \{1..k_m\}$, *on obtient un arbre t.*

La Figure 4.3.1 illustre le calcul de l'Exemple 4.3.14 selon l'Algorithme 4.3.13.

Exemple 4.3.14. (MINUS pour les concepts simples)
– Soient
$C := \exists r.(A \sqcap B \sqcap \forall r.(A \sqcap B) \sqcap \exists r.(A \sqcap B \sqcap C))$,
$Exp := \exists r.(A \sqcap \forall r.A \sqcap \exists r.(A \sqcap B))$.
Par l'algorithme, on a : MINUS(C, Exp) $= \{M_1, M_2, M_3, M_4\}$ où
– Par l'étape 3.(a) de l'algorithme, on obtient :
$M_1 = \exists r.(B \sqcap \forall r.(A \sqcap B) \sqcap \exists r.(A \sqcap B \sqcap C))$,
– Par l'étape 3.(b) de l'algorithme, on obtient :
$M_2 = \exists r.(A \sqcap B \sqcap \forall r.B \sqcap \exists r.(A \sqcap B \sqcap C))$

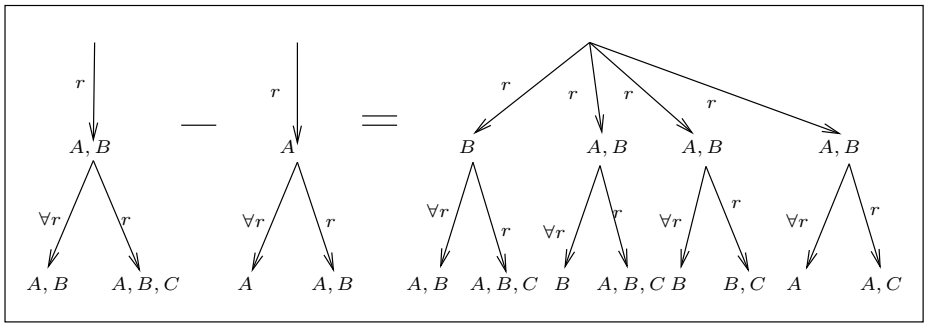

FIG. 4.3.1. MINUS pour les Concepts Simples

- Par les étapes 3.(c).(ii) et 3.(c).(iii) de l'algorithme, on obtient : $M_3 = \exists r.(A \sqcap B \sqcap \forall r.B \sqcap \exists r.(B \sqcap C))$
- Par les étapes 3.(c).(ii) et 3.(c).(iii) de l'algorithme, on obtient : $M_4 = \exists r.(A \sqcap B \sqcap \forall r.A \sqcap \exists r.(A \sqcap C))$

De plus, on a également OUBLIER(C,Exp) $= M_1 \sqcap M_2 \sqcap M_3 \sqcap M_4$

– Soient
$C := \forall r.(A \sqcap B \sqcap \forall r.A \sqcap \exists r.(A \sqcap B \sqcap C))$,
$Exp := \forall r.(A \sqcap \forall r.A \sqcap \exists r.(A \sqcap B))$.
De la même manière,
on obtient : MINUS(C, Exp) $= \{M'_1, M'_2, M'_3, M'_4\}$ où
- $M'_1 = \forall r.(B \sqcap \forall r.(A \sqcap B) \sqcap \exists r.(A \sqcap B \sqcap C))$,
- $M'_2 = \forall r.(A \sqcap B \sqcap \forall r.B \sqcap \exists r.(A \sqcap B \sqcap C))$.
- $M'_3 = \forall r.(A \sqcap B \sqcap \forall r.B \sqcap \exists r.(B \sqcap C))$
- $M'_4 = \forall r.(A \sqcap B \sqcap \forall r.C \sqcap \exists r.(A \sqcap C))$

Cependant, $C \equiv M'_1 \sqcap M'_2 \sqcap M'_3 \sqcap M'_4 \sqsubseteq Exp$.

Remarque 4.3.15. Selon l'Algorithme 4.3.13, la taille de chaque élément T de l'ensemble MINUS(C,D) est bornée par un nombre exponentiel en taille de C. En effet, les étapes 1-2 de l'algorithme accroît de façon polynômiale la taille des arbres dans MINUS(C,D). Nous montrons que la taille des arbres créés dans l'étape 3. de l'algorithme est bornée par un nombre exponentiel en taille de C. D'abord, le nombre d'éléments d'un ensemble $V \subseteq U$ est borné par la taille de C *i.e.* $m = |V| \leq |C|$. De plus, supposons que la taille de chaque arbre $r.F_{v^j}^{i_j}$ est bornée par un nombre exponentiel en taille de C (il est facile de vérifier l'hypothèse de récurrence dans le cas où la profondeur de C est égale à 2). A partir des étapes 3.(a) et 3.(b), on obtient l'arbre t qui est composé d'un nombre polynômial des arbres dont la taille est bornée par un nombre exponentiel en taille de C. Cela implique que la taille de la partie composée des restrictions existentielles du arbre $T \in$ MINUS(C,D) est bornée par nombre exponentiel en taille de C. Selon le travail dans [**BKM99 (5)**], la taille de l'arbre généré par le calcul lcs de l'étape 3.(c).(iii) est également bornée par nombre exponentiel en taille de C.
La proposition suivante établit la correction de l'Algorithme 4.3.13.

Proposition 4.3.16. *Soient* C, D des $\mathcal{FLE}$*-descriptions de concept normalisées.* C *peut être écrit sous la forme de* $\alpha.C'$ *où* $\alpha \in \{r, \forall r\}$, C' *est une* $\mathcal{FLE}$*-description de concept et* D *est une description de concept simple. Alors, l'ensemble des arbres* $\{\mathcal{G}_{C'} \mid C' \in \{MINUS(C, D)\}\}$ *calculé par l'Algorithme 4.3.13 a les propriétés suivantes :*

(1) $C \sqsubseteq C_t$ *pour tout* $t \in \{\mathcal{G}_{C'} \mid C' \in \{MINUS(C, D)\}\}$

(2) $C_t \not\sqsubseteq D$ *pour tout* $t \in \{\mathcal{G}_{C'} \mid C' \in \{MINUS(C, D)\}\}$

(3) *Pour toute* $\mathcal{FLE}$*-description de concept* A*, si* $C \sqsubseteq A$, $A \not\sqsubseteq D$*, alors il existe* $t \in \{\mathcal{G}_{C'} \mid C' \in \{MINUS(C, D)\}\}$ *tel que* $C_t \sqsubseteq A$.

En particulier,

(1) *si* $\alpha = r$ *i.e.* $C = r.C'$ *où* C' *est une* $\mathcal{FLE}$*-description de concept, l'opération OUBLIER peut être calculée comme suit :*

$$OUBLIER(C,\ D) \equiv \bigsqcap_{t \in \{\mathcal{G}_{C'} | C' \in MINUS(C,D)\}} C_t$$

(2) *si* $\alpha = \forall r$ *i.e.* $C = \forall r.C'$ *où* C' *est une* $\mathcal{FLE}$*-description de concept, on obtient :*

$$\{SEMI\text{-}OUBLIER(C,\ D)\} \subseteq \{MINUS(C,\ D)\}$$

Démonstration. Puisque MINUS(C, D) = C si $C \not\sqsubseteq D$, alors on peut supposer que $C \sqsubseteq D$. On démontre la proposition par récurrence sur la profondeur de C. Soient $\mathcal{G}$ =(N_C, E_C, v_0, l_C), $\mathcal{H}$ =(N_D, E_D, w_0, l_D) et $t \in \{\mathcal{G}_{C'} |\ C' \in$MINUS($C$, D)$\}$, t = (N_{C-D}, E_{C-D}, t_0, l_{C-D}).

– $|C| = 0$. Puisque D est une description simple, alors $l_D(w_0) = \emptyset$ si $|D| > 0$ et $l_D(w_0) = \{P\}$ si $|D| = 0$. On doit démontrer les propriétés suivantes de MINUS.

(1) $C \sqsubseteq C_t$, $\{\mathcal{G}_{C'} |\ C' \in$MINUS($C$, D)$\} = \{t\}$. Puisque $C \sqsubseteq D$, alors $|D| \leq |C|$ et $l_D(w_0) \subseteq l_C(v_0)$. Donc, $|D| = |C| = 0$ et $l_{C-D}(t_0) = l_C(v_0) \setminus l_D(w_0) \subseteq l_C(v_0)$.

(2) MINUS(C, D) $\not\sqsubseteq D$. De même, on a : $|D| = |C| = 0$ et $l_D(w_0) \not\subseteq l_C(v_0) \setminus l_D(w_0) = l_{C-D}(t_0)$.

(3) Si A est une $\mathcal{FLE}$-description de concept telle que $C \sqsubseteq A$ et $A \not\sqsubseteq D$, alors $A \sqsubseteq C_t$, $t \in \{\mathcal{G}_{C'} |\ C' \in$MINUS($C$, D)$\}$. Soit $\mathcal{G}_A$ =(N_A, E_A, x_0, l_A). De même, on a : $|A| = |D| = |C| = 0$ et $l_A(x_0) \subseteq l_C(v_0)$, $l_{C-D}(t_0) = l_C(v_0) \setminus l_D(w_0)$, et $l_D(w_0) \cap l_A(x_0) = \emptyset$, donc $l_A(x_0) \subseteq l_{C-D}(t_0)$.

– $|C| > 0$. On doit démontrer les propriétés suivantes de l'ensemble des arbres MINUS.

(1) $C \sqsubseteq C_t$ pour tout $t \in \{\mathcal{G}_{C'} |\ C' \in$MINUS($C$, D)$\}$. Selon l'Algorithme 4.3.13, on a : $l_{D-C}(t_0) \subseteq l_C(v_0)$. Soit (v, w) un $\forall r$-successeur de t_0 où v, w sont les $\forall r$-successeurs de v_0 et de w_0.

(a) Si $t \in \{\mathcal{G}_{C'} |\ C' \in$MINUS($C$, D)$\}$ est créé par l'étape 3.(a) de l'Algorithme 4.3.13, alors il est évident que $C \sqsubseteq C_t$.

(b) Si $t \in \{\mathcal{G}_{C'} |\ C' \in$MINUS($C$, D)$\}$ est créé par l'étape 3.(b) de l'Algorithme 4.3.13, alors $l_{C-D}(v, w) = l_C(v)$ où (v, w) est le α-successeur

de t_0 dans t, et par l'hypothèse de récurrence, $C_{G(v')} \sqsubseteq C_{t'}$ où $t' \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{\forall r.G(v')}, C_{\forall r.G(w')})\}$, v', w' sont les $\forall r$-successeurs de v et de w. De plus, les sous-arbres correspondant aux r-successeurs (v'', w'') de (v, w) sont des copies des sous-arbres $G(v'')$. Donc, $C \sqsubseteq C_t$.

(c) Si $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C, D)\}$ est créé par l'étape 3.(c) de l'Algorithme 4.3.13, alors $l_{C-D}(v, w) = l_C(v)$ où (v, w) est le α-successeur de t_0 dans t. On a : (v, w) est la racine des sous-arbres qui sont des copies des sous-arbres $r.\mathcal{G}(v'')$ où $l_D(w'') \not\subseteq l_C(v'')$, et des arbres $t' \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{r.\mathcal{G}(v'')}, C_{r.\mathcal{H}(w'')})\ \}$ créés par l'étape 3.(c), v'', w'' sont des r-successeurs respectivement de v et de w où $l_D(w'') \subseteq l_C(v'')$. Par hypothèse de récurrence, $C_{r.\mathcal{G}(v'')} \sqsubseteq C_{t'}$. De plus, $C_{\mathcal{G}(v')} \sqsubseteq lcs\{C_{F^{i_1}_{v_1}}, ..., C_{F^{i_m}_{v_m}}, C_{\mathcal{G}(v')}\}$ où v' est le $\forall r$-successeur de v et $r.F^{i_j}_{v_i} \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{r.\mathcal{G}(v^i)}, C_{r.\mathcal{H}(w'')})\}$, $i \in \{1, ...m\}$. Donc, $C \sqsubseteq C_t$. Par conséquent, $C \sqsubseteq \text{MINUS}(C, D)$.

(2) $C_t \not\sqsubseteq D$ pour tout $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C, D)\}$. Soit (v, w) un $\forall r$-successeur de t_0 où v, w sont les $\forall r$-successeurs de v_0 et de w_0.

(a) Si $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C, D)\}$ est créé par l'étape 3.(a) de l'Algorithme 4.3.13, alors il est évident que $C_t \not\sqsubseteq D$.

(b) Si $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C, D)\}$ est créé par l'étape 3.(b) de l'Algorithme 4.3.13, alors $l_{C-D}(v, w) = l_C(v)$ où (v, w) est le α-successeur de t_0 dans t. On a : (v, w) est la racine des sous-arbres qui sont des copies des sous-arbres $r.\mathcal{G}(v'')$ pour tout r-successeur v'' de v, et un arbre $t' \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{\forall r.\mathcal{G}(v')}, C_{\forall r.\mathcal{H}(w')})\ \}$, v', w' sont les $\forall r$-successeurs respectivement de v et de w. Par hypothèse de récurrence, $C_{t'} \not\sqsubseteq C_{\forall r.\mathcal{H}(w')}$. Donc, $C_t \not\sqsubseteq D$.

(c) Si $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C, D)\}$ est créé par l'étape 3.(c) de l'Algorithme 4.3.13, alors $l_{C-D}(v, w) = l_C(v)$ où (v, w) est le α-successeur de t_0 dans t. On a : (v, w) est la racine des sous-arbres qui sont des copies des sous-arbres $r.\mathcal{G}(v'')$, $l_D(w'') \not\subseteq l_C(v'')$, et les arbres $t' \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{r.\mathcal{G}(v'')}, C_{r.\mathcal{H}(w'')})\}$, $l_D(w'') \subseteq l_C(v'')$ où v'', w'' sont des r-successeurs respectivement de v et de w. Par hypothèse de récurrence, $C_{t'} \not\sqsubseteq C_{r.\mathcal{H}(w'')}$. Donc, $C_t \not\sqsubseteq D$.

(3) Montrons que si A est une $\mathcal{FLE}$-description de concept normalisée telle que $C \sqsubseteq A$ et $A \not\sqsubseteq D$, alors il existe $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$ tel que $C_t \sqsubseteq A$. Soient $\mathcal{F} = (N_A, E_A, x_0, l_A)$ et (v, w) le successeur de t_0 où v, w sont les successeurs de v_0 et de w_0. Puisque $C \sqsubseteq A$, alors pour tout α-successeur x de x_0, on a : $l_A(x) \subseteq l_C(v)$ et $l_A(x_0) \subseteq l_C(v_0) = l(t_0)$.

(a) si $l_D(w) \not\subseteq l_A(x)$, il existe un arbre $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$ qui est créé par l'étape 3.(a) de l'Algorithme 4.3.13 tel que $C_t \sqsubseteq A$.

(b) si $l_D(w) \subseteq l_A(x)$, $C_{\forall r.\mathcal{F}(x')} \not\sqsubseteq C_{\forall r.\mathcal{H}(w')}$ où x', w' sont les $\forall r$-successeurs de x et de w, alors, par hypothèse de récurrence et $C_{\forall r.\mathcal{G}(v')} \sqsubseteq C_{\forall r.\mathcal{F}(x')}$, il existe $t' \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(\forall r.\mathcal{G}(v'),\ \forall r.\mathcal{H}(w'))\}$ tel que $C_{t'} \sqsubseteq C_{\forall r.\mathcal{F}(x')}$. Donc, il existe un arbre t qui est créé par l'étape 3.(b) de l'Algorithme 4.3.13, $C_t \sqsubseteq C_{\alpha.\mathcal{F}(x)} \equiv A$.

(c) si $l_D(w) \subseteq l_A(x)$, $C_{\forall r.\mathcal{F}(x')} \sqsubseteq C_{\forall r.\mathcal{H}(w')}$ où x', w' sont les $\forall r$-successeurs de x et de w, alors, par $A \not\sqsubseteq D$, il existe un r-successeur w'' de w tel que $C_{r.\mathcal{F}(x'')} \not\sqsubseteq C_{r.\mathcal{H}(w'')}$ pour tous les r-successeurs x'' de x. Soient $x^1, ..., x^n$ les r-successeurs de x tels que $l_D(w'') \subseteq l_A(x^i)$, et $v^1, ..., v^m$ les r-successeurs de v, $m \leq n$ où $C_{\mathcal{G}(v^{I(i)})} \sqsubseteq C_{\mathcal{F}(x^i)}$ pour $i \in \{1, ..., n\}$, I est une surjection de $\{1, ..., n\}$ sur $\{1, ..., m\}$. Par hypothèse de récurrence, pour chaque $j \in \{1, ..., n\}$ il existe $t^{I(j)} \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{r.\mathcal{G}(v^{I(j)})},\ C_{r.\mathcal{H}(w'')})\}$, tel que $C_{t^{I(j)}} \sqsubseteq C_{r.\mathcal{F}(x^j)}$. Selon l'étape 3.(c) de l'Algorithme 4.3.13, on obtient un arbre t où (v, w) est le α-successeur de t_0 dans t , et (v, w) est la racine de *i)* les copies des sous-arbres $r.\mathcal{G}(v'')$ où $l_D(w'') \not\subseteq l_C(v'')$, *ii)* les arbres : $t^{I(1)} = r.F^{i_{I(1)}}_{v_{I(1)}}, ..., t^{I(n)} = r.F^{i_{I(n)}}_{v_{I(n)}}$ où $t^{I(j)} \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{r.\mathcal{G}(v^{I(j)})},\ C_{r.\mathcal{H}(w'')})$ *iii)* un arbre $\forall r.t'$ où $C_{t'} = lcs\{C_{F^{i_{I(1)}}_{v_{I(1)}}}, ..., C_{F^{i_{I(n)}}_{v_{I(n)}}},\ C_{\mathcal{G}(v')}\}$, v' est le $\forall r$-successeur de v. Puisque $C_{\mathcal{G}(v')} \sqsubseteq C_{\mathcal{F}(x')}$, $C_{F^{i_{I(k)}}_{v_{I(k)}}} \sqsubseteq C_{\mathcal{F}(x^k)} \sqsubseteq C_{\mathcal{F}(x')}$ pour tout $k \in \{1, ..., n\}$, par la définition de *lcs*, alors $lcs\{C_{F^{i_{I(1)}}_{v_{I(1)}}}, ..., C_{F^{i_{I(n)}}_{v_{I(n)}}},\ C_{\mathcal{G}(v')}\} \sqsubseteq C_{\mathcal{F}(x')}$. Par conséquent, on obtient : $C_t \sqsubseteq A$.

Les cas particuliers :

(1) Puisque C est une restriction existentielle, alors par l'Algorithme 4.3.13, C_t est également une restriction existentielle pour tout $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$. De plus, on a : $C \sqsubseteq C_t$ et $C_t \not\sqsubseteq D$ pour tout $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$. Donc, on obtient : $C \sqsubseteq \bigsqcap_{t \in MINUS(C,D)}\{C_t\}$ et $\bigsqcap_{t \in MINUS(C,D)}\{C_t\} \not\sqsubseteq D$. A partir de 3. de la cette proposition , on obtient *cqfd*.

(2) Supposons que $S \in \{\text{SEMI-OUBLIER}(C,D)\}$. On montre que $\mathcal{G}_S \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$. En effet, selon la démonstration de 3. de la Proposition, il existe un arbre $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$: $C_t \sqsubseteq S$. Par la Définition 4.3.4, cela implique que $C_t \equiv S$.

□

A partir de l'algorithme pour calculer l'ensemble MINUS d'une $\mathcal{FLE}$-description de concept simple, nous pouvons construire un algorithme qui permet de calculer l'ensemble MINUS d'une $\mathcal{FLE}$-description de concept arbitraire. Cette extension est directe de l'algorithme précédent sauf que le calcul *lcs* doive être effectué dès le premier niveau des arbres.

Algorithme 4.3.17. (*MINUS*) *Soient* C, D *des $\mathcal{FLE}$-descriptions de concept normalisées,* D *est une description de concept simple. On désigne $\mathcal{G}$ =(N_C, E_C, v_0, l_C) et $\mathcal{H}$ =(N_D, E_D, w_0, l_D). L'ensemble des descriptions de concept MINUS(C, D) est calculé par récurrence sur la profondeur de $\mathcal{G}$ comme suit.*

***Entrée** : $\mathcal{G}$, $\mathcal{H}$*
***Sortie** : Ensemble des arbres $T = \{\mathcal{G} - \mathcal{H}\} = \{\mathcal{G}_{C'} | \ C' \in MINUS(C, D)\}$*
Si $C_{\mathcal{G}} \not\sqsubseteq C_{\mathcal{H}}$, alors $T := \mathcal{G}$. Sinon,

(1) $T := \emptyset$.

(2) *Si $|\mathcal{G}| = 0$, T comporte un arbre qui possède un seul nœud et $l(v_0, w_0) = l_C(v_0) \setminus l_D(w_0)$. Notons que $|l_C(v_0)| = |l_D(w_0)| = 1$.*

(3) *Si $|\mathcal{H}| = 0$, T comporte un arbre qui est une copie de $\mathcal{G}$ avec pour racine le nœud (v_0, w_0) étiqueté par $l_C(v_0) \setminus l_D(w_0)$.*

(4) *Si le successeur w de w_0 est un $\forall r$-successeur dans $\mathcal{H}$, on a : $l_D(w) \subseteq l_C(v)$ où v est également le $\forall r$-successeur de v_0 dans $\mathcal{G}$.*
Pour chaque arbre $t' \in \{\mathcal{G}_{C'} | \ C' \in MINUS(C_{\forall r.\mathcal{G}(v')}, \ C_{\forall r.\mathcal{H}(w)})\}$ où v' est le $\forall r$-successeur de v_0, un arbre t est créé tel que t_0 soit la racine de t' avec $l_t(t_0) = l_C(v_0)$, et soit des copies des sous-arbres $r.\mathcal{G}(v)$ pour tout r-successeur v de v_0. On a : $T := T \cup \{t\}$.

(5) *Si le successeur w de w_0 est un r-successeur dans $\mathcal{H}$, alors on crée un ensemble vide d'arbres T'.*

   (a) *Si v est le successeur unique de v_0, on a : $l_D(w) \subseteq l_C(v)$. Appliquer l'Algorithme 4.3.13 pour calculer l'ensemble des arbres : $\{\mathcal{G}_{C'} | \ C' \in MINUS(C_{r.\mathcal{G}(v)}, \ C_{r.\mathcal{H}(w)})\}$. Pour chaque arbre $t'' \in \{\mathcal{G}_{C'} | \ C' \in MINUS(C_{r.\mathcal{G}(v)}, C_{r.\mathcal{H}(w)})\}$, on obtient un arbre t' dont la racine t'_0 est la racine de t'' avec $l_{t'}(t'_0) = l_C(v_0)$. On a : $T' := T' \cup \{t'\}$.*

   (b) *Sinon, (il existe plusieurs successeurs v de v_0),*
   Pour chaque r-successeur v de v_0 dans $\mathcal{G}$ où $l_D(w) \subseteq l_C(v)$, et pour chaque arbre $t'' \in \{\mathcal{G}_{C'} | \ C' \in MINUS(C_{r.\mathcal{G}(v)}, \ C_{r.\mathcal{H}(w)})\}$ qui est calculé récursivement par l'Algorithme 4.3.13, on obtient un arbre t' dont la racine t'_0 est la racine de t'' avec $l_{t'}(t'_0) = l_C(v_0)$. De plus, pour chaque r-successeur v de v_0 dans $\mathcal{G}$ où $l_D(w) \not\sqsubseteq l_C(v)$, on obtient un r-successeur (v, w) de t_0 qui est la racine d'une copie du sous-arbre $\mathcal{G}(v)$.
   On a : $T' := T' \cup \{t'\}$

   (c) *A partir de l'ensemble T' créé,*
   Pour chaque sous-ensemble des arbres $V = \{t^1, ..., t^m\}$ de T', on obtient un arbre t dont la racine t_0 avec $l_t(t_0) = l_{t^1}(t^1_0) \cup ... \cup l_{t^m}(t^m_0)$ est la racine de tous les arbres $t^i \in V$. De plus, le $\forall r$-successeur (v', w') de t_0 est la racine de l'arbre :
   lcs{ $\{C_{t(v,w)}$ | pour tout r-successeur (v, w) de $t_0\} \cup \{C_{\mathcal{G}(v')}$ | v' est le $\forall r$-successeur de v_0 } }.
   On a : $T' := T' \cup \{t\}$.

La Figure 4.3.2 illustre le calcul de l'Exemple 4.3.18 par l'Algorithme 4.3.17.

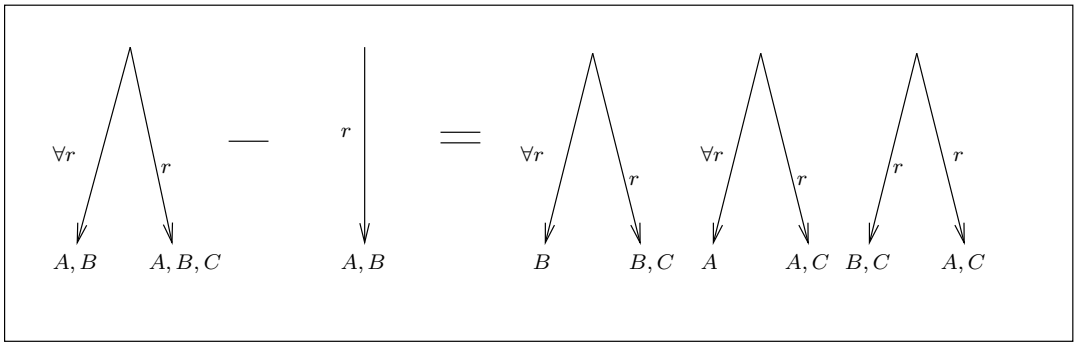

FIG. 4.3.2. MINUS où Exp est une restriction existentielle

Exemple 4.3.18. (MINUS où *Exp* est une restriction existentielle)
- Soient
  $C := \forall r.(A \sqcap B) \sqcap \exists r.(A \sqcap B \sqcap C)$
  $Exp := \exists r.(A \sqcap B)$.
- Par l'étape 5.b de l'Algorithme 4.3.17, d'abord on obtient l'ensemble des arbres $T' = \{T_1, T_2\}$ où

$C_{T_1} = \exists r.(B \sqcap C)$, $C_{T_2} = \exists r.(A \sqcap C)$

- Par l'étape 5.c de l'algorithme, on obtient : MINUS(C, Exp) $= \{M_1, M_2, M_3\}$ où
  - $M_1 = \forall r.B \sqcap \exists r.(B \sqcap C)$,
  - $M_2 = \forall r.A \sqcap \exists r.(A \sqcap C)$.
  - $M_3 = \exists r.(B \sqcap C) \sqcap \exists r.(A \sqcap C)$

Le théorème suivant établit la relation entre les descriptions de concept calculées par l'Algorithme 4.3.17 et les SEMI-OUBLIER.

Théorème 4.3.19. *Soient* C, D des $\mathcal{FLE}$*-descriptions de concept normalisées et* D *est une description de concept simple. Il existe un algorithme pour calculer tous les SEMI-OUBLIER*(C, D)*. Plus précisément,*

$$SEMI\text{-}OUBLIER(C, D) \subseteq MINUS(C, D)$$

DÉMONSTRATION. Puisque MINUS(C, D) $= C$ si $C \not\sqsubseteq D$, alors on peut supposer que $C \sqsubseteq D$. Nous démontrons le théorème par récurrence sur la profondeur de C. Soient $\mathcal{G} = (N_C, E_C, v_0, l_C)$, $\mathcal{H} = (N_D, E_D, w_0, l_D)$ et $t = (N_t, E_t, u_0, l_t)$ pour un arbre $t \in \{\mathcal{G}_{C'} | \ C' \in \text{MINUS(C, D)}\}$.
- $|C| = 0$. Puisque D est une description simple, alors $l_D(w_0) = \emptyset$ si $|D| > 0$ et $l_D(w_0) = \{P\}$ si $|D| = 0$. On doit démontrer les propriétés suivantes de MINUS. Notons que $\{\mathcal{G}_{C'} | \ C' \in \text{MINUS(C, D)}\}$ contient l'arbre unique dans ce cas.

(1) $C \sqsubseteq$ MINUS(C, D). Puisque $C \sqsubseteq D$, alors $|D| \leq |C|$ et $l_D(w_0) \subseteq l_C(v_0)$. Donc, $|D| = |C| = 0$ et $l_{C-D}(u_0) = l_C(v_0) \setminus l_D(w_0) \subseteq l_C(v_0)$.

(2) MINUS(C, D) $\not\sqsubseteq D$. De même, on a : $|D| = |C| = 0$ et $l_D(w_0) \not\subseteq l_C(v_0) \setminus l_D(w_0) = l_{C-D}(u_0)$.

(3) Si A est une $\mathcal{FLE}$-description de concept telle que $C \sqsubseteq A$ et $A \not\sqsubseteq D$, alors $A \sqsubseteq$ MINUS(C, D). Soit $\mathcal{G}_A$ =(N_A, E_A, x_0, l_A). De même, on a : $|D| = |C| = 0$ et $l_A(x_0) \subseteq l_C(v_0)$, $l_t(u_0) = l_C(v_0) \setminus l_D(w_0)$, et $l_D(w_0) \cap l_A(x_0) = \emptyset$, donc $l_A(x_0) \subseteq l_t(u_0)$.

– $|C| > 0$. On doit démontrer les propriétés suivantes de MINUS.

(1) $C \sqsubseteq C_t$ où $t \in \{\mathcal{G}_{C'} |\ C' \in$ MINUS(C, D)$\}$. Selon l'Algorithme 4.3.17 , t est normalisé et $l_t(u_0) \subseteq l_C(v_0)$.

(a) Si le successeur w de w_0 est un $\forall r$-successeur, alors la racine t_0 de t a un $\forall r$-successeur (v, w), $\forall r.t(v, w) \in \{\mathcal{G}_{C'} |\ C' \in$ MINUS($C_{\forall r.\mathcal{G}(v)}$, $C_{\forall r.\mathcal{H}(w)}$)$\}$ où v est le $\forall r$-successeur v_0 et $l_t(v, w) \subseteq l_C(v)$. Par hypothèse de récurrence, on a : $C_{\forall r.\mathcal{G}(v)} \sqsubseteq C_{\forall r.t(v,w)}$. De plus, la racine t_0 de t a les r-successeurs (v', w) où $t(v', w) = \mathcal{G}(v')$, v' est un r-successeur de v_0. Donc, on a : $l_t(v, w) \subseteq l_C(v)$ et $C_{\mathcal{G}(v)} \sqsubseteq C_{t(v,w)}$ pour tout successeur (v, w) de t_0.

(b) Si le successeur w de w_0 est un r-successeur, supposons qu'il existe plusieurs successeurs v de v_0 au premier niveau. Selon l'Algorithme 4.3.17, les r-successeurs (v, w) de (v_0, w_0) sont créés de i) les arbres $t \in \{\mathcal{G}_{C'} |\ C' \in$ MINUS($C_{r.\mathcal{G}(v)}$, $C_{r.\mathcal{H}(w)}$)$\}$ où $l_D(w) \subseteq l_C(v)$, v, w sont les r-successeurs de v_0 et de w_0. Par l'Algorithme 4.3.13, on obtient : $C \sqsubseteq C_{r.\mathcal{G}(v)} \sqsubseteq C_t$, ii) les arbres qui sont copiés des sous-arbres $\mathcal{G}(v)$ où $l_t(v, w) = l_C(v)$, $l_D(w) \not\subseteq l_C(v)$. Donc, $C_{r.\mathcal{G}(v)} \sqsubseteq C_{r.(\mathcal{G}-\mathcal{H})(v,w)}$ pour tous les r-successeurs (v, w) de t_0. De plus, selon l'Algorithme 4.3.17, un arbre t est créé d'un sous-ensemble des arbres T' qui correspondent aux restrictions existentielles *i.e.* chaque $t' \in T'$ a une racine t'_0 qui a un seul r-successeur (v, w), et l'expression sous la restriction universelle de t est calculée comme *lcs* de *i)* les expressions sous la restriction existentielles et *ii)* l'expression sous la restriction universelle $C_{\mathcal{G}(v')}$. Donc, d'après la définition de *lcs*,
$C_{\mathcal{G}(v')} \sqsubseteq lcs\{\{C_{t(v,w)} \mid$ pour tout r-successeur (v, w) de $t_0\} \cup \{C_{\mathcal{G}(v')} \mid v' : \forall r$-successeur de $v_0\}\}$. Par conséquent, $C \sqsubseteq$ MINUS(C,D).

(2) $C_t \not\sqsubseteq D$ où $t \in \{\mathcal{G}_{C'} |\ C' \in$ MINUS(C, D)$\}$

(a) Si le successeur w de w_0 est un $\forall r$-successeur, t a un $\forall r$-successeur (v, w) où $\forall r.t(v, w) \in \{\mathcal{G}_{C'} |\ C' \in$MINUS($C_{\forall r.\mathcal{G}(v)}$,$C_{\forall r.\mathcal{H}(w)}$)$\}$, v est le $\forall r$-successeur de v_0. Par la Proposition 4.3.16, on a : $C_{\forall r.t(v,w)} \not\sqsubseteq C_{\forall r.\mathcal{H}(w)} = D$. Donc, $C_t \not\sqsubseteq D$.

(b) Si le successeur w de w_0 est un r-successeur, supposons qu'il existe plusieurs successeurs v de v_0 au premier niveau. Selon l'Algorithme 4.3.17, les r-successeurs (v, w) de t_0 d'un arbre t sont créés de i) les

arbres $t' \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{r.\mathcal{G}(v)}, Cr.\mathcal{H}(w))\}$ où $l_D(w) \subseteq l_C(v)$, v, w sont les r-successeurs de v_0 et de w_0, et par la Proposition 4.3.16, $C_t \not\sqsubseteq C_{r.\mathcal{H}(w)}$ ii) les copies des sous-arbres $\mathcal{G}(v)$ où $C_{r.\mathcal{G}(v)} \not\sqsubseteq D$. Par conséquent, MINUS($C$,$D$) $\not\sqsubseteq D$.

(3) Montrons que si A est une $\mathcal{FLE}$-description de concept normalisée telle que $C \sqsubseteq A$ et $A \not\sqsubseteq D$, alors il existe $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$ tel que $C_t \sqsubseteq A$. Soit $\mathcal{F} = (N_A, E_A, x_0, l_A)$.

(a) si le successeur w de w_0 est un $\forall r$-successeur, alors selon l'Algorithme 4.3.17 et Proposition 4.3.16, il existe un arbre $t' \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{\forall r.\mathcal{G}(v)}, C_{\forall r.\mathcal{H}(w)})\}$ tel que $C_{t'} \sqsubseteq C_{\mathcal{F}(x)}$ où v est le $\forall r$-successeur de v_0 dans $\mathcal{G}$, et x le $\forall r$-successeur de x_0 dans $\mathcal{F}$. Donc, on obtient : $C_t \sqsubseteq A$ où l'arbre t avec la racine t_0, $l_t(t_0) = l_C(v_0)$, est construit de l'arbre t' et des copies des arbres $r.\mathcal{G}(v)$ où les v sont les r-successeur de v_0 dans $\mathcal{G}$. Selon l'Algorithme 4.3.17, on a : $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$.

(b) si le successeur w de w_0 est le r-successeur, supposons que $x^1, ..., x^m$ les r-successeurs de x_0 tels que $l_D(w) \subseteq l_A(x^i)$, $C_{\mathcal{F}(x^i)} \not\sqsubseteq C_{\mathcal{H}(w)}$, $C_{\mathcal{G}(v^i)} \sqsubseteq C_{\mathcal{F}(x^i)}$ pour $i \in \{1, ..., m\}$, et $x^{m+1}, ..., x^{m+n}$ les r-successeurs de x_0 tels que $l_D(w) \not\subseteq l_A(x^{m+i})$, $C_{\mathcal{G}(v^{m+i})} \sqsubseteq C_{\mathcal{F}(x^{m+i})}$ pour $i \in \{1, ..., n\}$. Pour simplifier l'écriture, supposons que $v^i \neq v^j$, sinon on peut définir une surjection de $\{x^1, ..., x^{m+n}\}$ dans $\{v^1, ..., v^l\}$ comme dans la démonstration de la Proposition 4.3.16. Par l'Algorithme 4.3.17, on peut trouver un sous-ensemble $V = \{t^1, ..., t^{m+n}\}$ de l'ensemble des arbres :
$\bigcup\{\ \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C_{r.\mathcal{G}(v^i)}, C_{r.\mathcal{H}(w)})\} \mid i \in \{1, ..., m\}\} \cup$
$\{\mathcal{G}(v^{m+i}) |\ i \in \{1, ..., n\}\}$ tel que $C_{t^j} \sqsubseteq C_{\mathcal{F}(x^j)}$ pour tout $j \in \{1, ..., m+n\}$. De plus, si x'' est le $\forall r$-successeur de x_0, alors $C_{\mathcal{G}(v'')} \sqsubseteq C_{\mathcal{F}(x'')}$ et $C_{t(v^i,w^i)} \sqsubseteq C_{\mathcal{F}(x^i)} \sqsubseteq C_{\mathcal{F}(x'')}$ pour tout $i \in \{1, ..., m+n\}$ où v'' est le $\forall r$-successeur de v_0 et donc, $lcs\{C_{t(v^1,w^1)},...,C_{t(v^{m+n},w^{m+n})}, C_{\mathcal{G}(v'')}\} \sqsubseteq C_{\mathcal{F}(x'')}$. Cela implique qu'un arbre t se compose des arbres $t^i \in V$ qui correspondent aux restrictions existentielles et d'un arbre $t' = \forall r.lcs\{C_{t(v^1,w^1)},...,C_{t(v^{m+n},w^{m+n})}, C_{\mathcal{G}(v'')}\}$ qui correspond à la restriction universelle. Par conséquent, on obtient : MINUS(C,D) $\sqsubseteq A$.

Maintenant, on démontre que si $S \in \{\text{SEMI-OUBLIER}(C,D)\}$ alors $\mathcal{G}_S \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$. En effet, selon la démonstration ci-dessus, il existe un arbre $t \in \{\mathcal{G}_{C'} |\ C' \in \text{MINUS}(C,D)\}$: $C_t \sqsubseteq S$. Par la Définition 4.3.4, cela implique que $C_t \equiv S$.

□

En général, l'Algorithme 4.3.17 génère un ensemble des arbres correspondant aux SEMI-OUBLIER. Cela signifie que la description de concept OUBLIER(C, *Exp*) selon la Définition 4.3.5 n'existe pas dans certains cas. Le cas le plus important est

que la définition du concept C contient la restriction universelle au top-level et Exp est une restriction existentielle. Cependant, une analyse des descriptions de concept obtenues dans l'ensemble SEMI-OUBLIER montre que ces descriptions de concept ne jouent pas le même rôle. En effet, il existe une SEMI-OUBLIER qui peut générer les autres SEMI-OUBLIER sans recourir à la description de concept Exp. D'autre part, cette SEMI-OUBLIER est la description de concept OUBLIER(C', Exp') où C' et Exp' sont obtenues de C et Exp en remplaçant de façon exhaustive toutes les restrictions universelles par le concept $\top$. Cela signifie que cette SEMI-OUBLIER est compatible à OUBLIER qui est définie dans le langage $\mathcal{EL}$. Pour cette raison, en se basant sur l'opérateur MINUS, nous proposons une nouvelle définition pour l'opération OUBLIER dans le langage $\mathcal{FLE}$ comme suit :

Définition 4.3.20. (OUBLIER pour $\mathcal{FLE}$) Soit C une $\mathcal{FLE}$-description de concept normalisée. Soit Exp une $\mathcal{FLE}$ -description de concept simple et $Exp \not\equiv \top$.

(1) Si C ne contient pas de restriction universelle au top-level, alors
OUBLIER$(C, Exp) := \bigsqcap_{t \in \{\mathcal{G}_{C'} | C' \in MINUS(C, Exp)\}} C_t$

(2) Si Exp est une restriction universelle, alors
OUBLIER$(C, Exp) := \bigsqcap_{P \in Prim(C)} P \sqcap \forall r.$OUBLIER$(var(C), var(Exp)) \sqcap \bigsqcap_{C' \in ex(C)} \exists r.C'$

(3) Si C contient la restriction universelle au top-level et Exp est une restriction existentielle, alors
OUBLIER$(C, Exp) := \bigsqcap_{t \in \{\mathcal{G}_{C'} | C' \in MINUS(\exists r.D, Exp), D \in ex(C)\}} C_t \sqcap$
$\forall r.lcs\{\{var(C)\} \cup \{ex(C_t) | t \in \{\mathcal{G}_{C'} | C' \in MINUS(\exists r.D, Exp), D \in ex(C)\}\}\}$

où $Prim(C)$ est l'ensemble des concepts primitifs au top-level de C, $var(C)$ est l'expression sous la restriction universelle au top-level de C et $ex(C)$ est l'ensemble des expressions sous les restrictions existentielles au top-level de C (cf. Définition 2.2.12)

Dans l'Exemple 4.3.18, on a :

$$\text{OUBLIER}(C, Exp) := \exists r.(B \sqcap C) \sqcap \exists r.(A \sqcap C)$$

Remarque 4.3.21. Même si la Définition 4.3.20 n'assure en général pas la minimalité (la condition 3.(c) dans le Définition 4.3.5), elle a les propriétés importantes suivantes :

(1) OUBLIER(C, Exp) selon la Définition 4.3.20 assure la minimalité dans les cas où le concept C ne contient pas de restriction universelle au top-level, ou Exp est une restriction universelle au top-level (les cas 1. et 2. dans la Définition 4.3.20). En effet, si le concept C ne contient pas de restriction universelle au top-level, l'affirmation est déduite directement de la Proposition 4.3.16. Si la description de concept Exp est une restriction universelle, on considère les deux cas suivants :

(a) Si $Exp = \forall r...\forall r.P$, $P \in N_C$, alors SEMI-OUBLIER$(var(C), var(Exp))$ existe uniquement. Donc, l'affirmation est évidente.

(b) Si $Exp = \forall r...\forall r.\exists r.D$ où D est une $\mathcal{FLE}$-description de concept. Puisque $C \sqsubseteq Exp$ et C est normalisée, alors C peut être de la forme : $C = \forall r...\forall r.\exists r.C' \sqcap C''$ où $\forall r...\forall r.\exists r.C' \sqsubseteq Exp$ et donc,
OUBLIER(C, Exp) = OUBLIER$(\forall r...\forall r.\exists r.C'$, $\forall r...\forall r.\exists r.D) \sqcap C''$
Cela implique que le calcul de cette OUBLIER peut se traduire en le calcul dans le cas où le concept C ne contient pas de restriction universelle au top-level.

(2) OUBLIER$(C, Exp) \in$ SEMI-OUBLIER(C, Exp) . Cela est déduit directement du Théorème 4.3.19. En effet, les cas 1. et 2. dans la Définition 4.3.20 sont évidents. Le cas 3. dans la Définition 4.3.20 est l'arbre obtenu de l'étape 5.(c) de l'Algorithme 4.3.17 où $V = T'$.

(3) A l'exception du cas où $C_1 \sqsubseteq Exp$, $C_2 \not\sqsubseteq Exp$, on a : OUBLIER(C_1, Exp) $\sqsubseteq$ OUBLIER(C_2, Exp) si $C_1 \sqsubseteq C_2$.

En effet, si C_1 ne contient que des restrictions existentielles, alors C_2 et Exp ne contiennent aussi que des restrictions existentielles. L'affirmation est évidente.

Supposons que C_1 contienne la restriction universelle.

Si $C_1 \not\sqsubseteq Exp$, alors $C_2 \not\sqsubseteq Exp$ et donc, OUBLIER$(C_1, Exp) = C_1$ et OUBLIER$(C_2, Exp) = C_2$.

Si $C_1 \sqsubseteq Exp$, $C_2 \sqsubseteq Exp$, et Exp est une restriction existentielle, alors d'après la Définition 4.3.20 et la Proposition 4.3.16, l'affirmation est déduite des faits suivants :
$\bigsqcap_{C' \in MINUS(\exists r.D, Exp), D \in ex(C_1)} C' \sqsubseteq \bigsqcap_{C'' \in MINUS(\exists r.D, Exp), D \in ex(C_2)} C''$,
et
$\forall r.lcs\{\{var(C_1)\} \cup \{ex(C_t) | t \in \{\mathcal{G}_{C'} | C' \in \text{MINUS}(\exists r.D, Exp), D \in ex(C_1)\}\}\}$
$\sqsubseteq$
$\forall r.lcs\{\{var(C_2)\} \cup \{ex(C_t) | t \in \{\mathcal{G}_{C'} | C' \in \text{MINUS}(\exists r.D, Exp), D \in ex(C_2)\}\}\}$
car $C_1 \sqsubseteq C_2$.

Si $C_1 \sqsubseteq Exp$, $C_2 \sqsubseteq Exp$, et Exp est une restriction universelle, alors d'après la Définition 4.3.20 et la Proposition 4.3.16, l'affirmation est déduite du fait suivant :

$\forall r$.OUBLIER$(var(C_1), var(Exp)) \sqsubseteq \forall r$.OUBLIER$(var(C_2), var(Exp))$ où la $var(C_1)$, $var(C_2)$ et $var(Exp)$ sont des restrictions existentielles.

Opération OUBLIER dans $\mathcal{ELN}^-$. Maintenant, nous essayons d'étendre la procédure du calcul OUBLIER à un sous-langage du $\mathcal{ELN}$, appelé $\mathcal{ELN}^-$. Ce dernier est obtenu du $\mathcal{ELN}$ en limitant aux descriptions de concept dans lesquelles deux conjoncts de la forme : $(\leq mr)$ et $(\geq nr)$ où $m < n$ n'apparaissent pas simultanément à tout niveau. De plus, le langage $\mathcal{ELN}^-$ n'autorise pas à tout niveau le

conjonct de la forme $(\leq 0r)$. Ces limites permettent d'éliminer sémantiquement le concept vide (bottom-concept) des descriptions de concept.

D'abord nous simplifions un résultat présenté dans **[KM01 (49)]** qui caractérise la subsomption dans le langage $\mathcal{ELN}^-$. Cette simplification nous permet de montrer l'existence de SEMI-OUBLIER et son unicité. Nous avons besoin de la définition suivante.

Définition 4.3.22. Soit C une $\mathcal{ELN}^-$-description de concept.

(1) C est normalisée si chaque conjonction dans C contient au plus d'une restriction de cardinalité de la forme $(\geq kr)$ et au plus d'une restriction de cardinalité de la forme $(\leq kr)$. Puisque $(\geq mr) \sqcap (\geq nr) \equiv (\geq nr)$ si $n \geq m$ et $(\leq mr) \sqcap (\leq nr) \equiv (\leq nr)$ si $n \leq m$, toute $\mathcal{ELN}^-$-description de concept est normalisable.

(2) On désigne par $prim(C)$ l'ensemble des noms de concept apparaissant au top-level de C.

(3) On désigne par $ex_r(C)$ l'ensemble de tous les C' qui apparaissent dans les restrictions existentielles $\exists r.C'$ de C.

(4) $min_r(C) := max\{k | C \sqsubseteq (\geq kr)\}$. Notons que $min_r(C)$ est toujours fini.

(5) $max_r(C) := min\{k | C \sqsubseteq (\leq kr)\}$; s'il n'existe pas k tel que $C \sqsubseteq (\leq kr)$, alors $max_r(C) = \infty$.

(6) Une application $\alpha : \{1, ..., n\} \longrightarrow 2^{\{1,...,m\}}$ où $n := min(max_r(C), |ex_r(C)|)$ et $m := |ex_r(C)|$ ($|E|$ signifie la cardinalité de l'ensemble E) , est appelée *application existentielle* si elle satisfait les conditions suivantes :

   (a) $\alpha(i) \neq \emptyset$ pour tout $1 \leq i \leq n$;

   (b) $\bigcup_{1 \leq i \leq n} \alpha(i) = \{1, ..., m\}$ et $\alpha(i) \cap \alpha(j) = \emptyset$ pour tout $1 \leq i < j \leq n$;

   On désigne par $\Gamma_r(C)$ l'ensemble des applications existentielles α sur C où $\Gamma_r(C) = \emptyset$ si $ex_r(C) = \emptyset$. On utilise également la notation suivante :

$$ex_r(C)^\alpha := \{\textstyle\bigsqcap_{j \in \alpha(i)} C_j | C_j \in ex_r(C), 1 \leq i \leq n\}$$

Remarque 4.3.23. A partir de la définition, on obtient les propriétés suivantes : $min_r(C)$ est k maximum tel que $m \leq k \leq max(m, |ex_r(C)|)$ si C contient la restriction de cardinalité $(\geq mr)$ au top-level. Sinon, $min_r(C)$ est k maximum tel que $0 \leq k \leq |ex_r(C)|$. D'autre part, $max_r(C) = m$ s'il existe le conjonct $(\leq mr)$ au top-level de C. Sinon, $max_r(C) = \infty$.

Le théorème suivant est une version simplifiée du théorème 2. dans **[KM01 (49)]** pour le langage $\mathcal{ELN}^-$. Nous utilisons ce théorème pour montrer l'unicité de SEMI-OUBLIER dans $\mathcal{ELN}^-$.

Théorème 4.3.24. [KM01 (49)] *Soient C, D des $\mathcal{ELN}^-$-descriptions de concept normalisées (Définition 4.3.22). Alors, $C \sqsubseteq D$ ssi $D \equiv \top$, ou les conditions suivantes sont vérifiées :*

(1) $prim(D) \subseteq prim(C)$;

(2) $max_r(C) \leq max_r(D)$;

(3) $min_r(C) \geq min_r(D)$;

(4) *Pour toute* $D' \in ex_r(D)$, *on a :*

(a) $ex_r(C) = \emptyset$, $min_r(C) \geq 1$ *et* $D' \equiv \top$; *ou*

(b) $ex_r(C) \neq \emptyset$, *pour chaque* $\alpha \in \Gamma_r(C)$, *il existe* $C' \in ex_r(C)^\alpha$ *telle que* $C' \sqsubseteq D'$.

Le Théorème 4.3.24 fournit une condition nécessaire et suffisante pour la subsomption dans la langage $\mathcal{ELN}^-$. En particulier, la condition 4.(b) caractérise la fusion de restrictions existentielles au même niveau à cause de la restriction de cardinalité de la forme $(\leq mr)$. Par exemple, la subsomption suivante est établie grâce à la fusion des restrictions existentielles du côté gauche :

$$\exists r.(A \sqcap B) \sqcap \exists r.(A \sqcap C) \sqcap \exists r.(B \sqcap C) \sqcap (\leq 1r) \sqsubseteq \exists r.(A \sqcap B \sqcap C)$$

En effet, il existe uniquement l'application $\alpha : \{1\} \longrightarrow 2^{\{1,\ldots,3\}}$ et $\alpha(1) = \{1, 2, 3\}$ satisfait la condition 4.(b) du théorème.
La proposition suivante établit l'unicité de SEMI-OUBLIER dans le langage $\mathcal{ELN}^-$.

Proposition 4.3.25. *Soient* C, Exp *des* $\mathcal{ELN}^-$*-descriptions de concept normalisées (Définition 4.3.22) où* Exp *ne contient pas de conjonction au top-level et* $Exp \neq \top$. *Alors,*

(1) *Si* Exp *n'est pas une restriction existentielle et SEMI-OUBLIER*(C, Exp) *existe , alors SEMI-OUBLIER*(C, Exp) *est unique modulo équivalence.*

(2) *Si* Exp *est une restriction existentielle,* $max_r(C) = \infty$ *et SEMI-OUBLIER*(C, Exp) *existe, alors SEMI-OUBLIER*(C, Exp) *est unique modulo équivalence.*

Démonstration. Nous montrons les cas suivants.

(1) Si Exp est un nom de concept, la proposition est évidente.

(a) Si $Exp = (\leq mr)$. Supposons que $ex_r(C) \neq \emptyset$ et qu'il n'existe pas de conjonct $(\geq nr)$ au top-level de C. Sans perte de généralité, C peut être écrite comme suit : $C \equiv \exists r.C_1 \sqcap \exists r.C_2 \sqcap (\leq m'r)$ où C_1, C_2 sont des $\mathcal{ELN}^-$-descriptions de concept, $0 < m' \leq m$ (Théorème 4.3.24). D'autre part, on a :

$$\exists r.(C_1 \sqcap C_2) \sqsubseteq \exists r.C_1 \sqcap \exists r.C_2$$

Puisque toute $\mathcal{ELN}^-$- description de concept est consistante, alors il existe un modèle $\mathcal{I}$ de $D := \exists r.(C_1 \sqcap C_2)$ où $x \in D^\mathcal{I}$ et il existe un r-successeur y de x dans le modèle $\mathcal{I}$ tel que $y \in C_1^\mathcal{I} \cap C_2^\mathcal{I}$. Il est évident que $\mathcal{I}$ est également un modèle de C. Donc, dans le modèle $\mathcal{I}$, on a : $x \in C^\mathcal{I}$ et $x \notin (\geq 2r)^\mathcal{I}$. Cela implique qu'il existe un modèle $\mathcal{I}$ tel que $C^\mathcal{I} \not\subseteq (\geq 2r)^\mathcal{I}$. Par conséquent, $min_r(C) = 1$.
Supposons qu'il existe un conjonct $(\geq nr)$ au top-level de C. Sans perte

de généralité, C peut être écrite comme suit : $C \equiv \exists r.C_1 \sqcap \exists r.C_2 \sqcap (\geq nr)$ où , $n > 1$ et C_1, C_2 sont des $\mathcal{ELN}^-$-descriptions de concept. Puisque toute $\mathcal{ELN}^-$- description de concept est consistante, alors il existe un modèle $\mathcal{I}$ de C où $x \in C^{\mathcal{I}}$ et il existe n r-successeurs y_i de x dans le modèle $\mathcal{I}$ tel que $y_1 \in C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$. Cela implique qu'il existe un modèle $\mathcal{I}$ tel que $C^{\mathcal{I}} \not\subseteq (\geq (n+1)r)^{\mathcal{I}}$. Donc, $min_r(C) = n$. Cela nous permet de conclure que $min_r(C)$ ne dépend pas d'occurrences de restriction existentielle. En conséquence, $min_r(C \sqcap D) = max(min_r(C), min_r(D))$ (*) où C, D sont des $\mathcal{ELN}^-$-descriptions de concept. De plus, on obtient également $max_r(C \sqcap D) = min(max_r(C), max_r(D))$ (**).
Soient C_1, C_2 des SEMI-OUBLIER du C. On a : $C \sqsubseteq C_1$, $C \sqsubseteq C_2$, $C_1 \not\sqsubseteq Exp$, $C_2 \not\sqsubseteq Exp$, et donc $C \sqsubseteq C_1 \sqcap C_2$. Par le Théorème 4.3.24, on a : $max_r(C_1) > max_r(Exp)$ et $max_r(C_2) > max_r(Exp)$. De plus, à partir du résultat (**) obtenu ci-dessus, on a : $max_r(C_1 \sqcap C_2) = min(max_r(C_1), max_r(C_2)) > max_r(Exp)$. Donc, le Théorème 4.3.24 nous permet de déduire : $C_1 \sqcap C_2 \not\sqsubseteq Exp$. D'autre part, par la définition de l'opération SEMI-OUBLIER on obtient : $C_1 \sqcap C_2 \not\sqsubset C_1$. Puisque $C_1 \sqcap C_2 \not\sqsubseteq C_1$ est impossible, alors $C_1 \equiv C_2$.

(b) Si $Exp = (\geq mr)$. On a : $min_r(Exp) = m$, $max_r(Exp) = \infty$. Soient C_1, C_2 des SEMI-OUBLIER du C. On a : $C \sqsubseteq C_1$, $C \sqsubseteq C_2$, $C_1 \not\sqsubseteq Exp$, $C_2 \not\sqsubseteq Exp$, et donc $C \sqsubseteq C_1 \sqcap C_2$. A partir (*), on obtient : $min_r(C \sqcap D) = max(min_r(C_1), min_r(C_2)) < m$. Le même argument dans 1.(a) permet d'obtenir $C_1 \equiv C_2$.

(2) Si $Exp = \exists r.D'$. On a : $min_r(Exp) = 1$, $max_r(Exp) = \infty$. Puisque $C \sqsubseteq Exp$, alors $min_r(C) \geq 1$. Soient C_1, C_2 des SEMI-OUBLIER du C. On a : $C \sqsubseteq C_1$, $C \sqsubseteq C_2$ et donc $C \sqsubseteq C_1 \sqcap C_2$. De plus, on a également : $C_1 \not\sqsubseteq Exp$, $C_2 \not\sqsubseteq Exp$. A partir du Théorème 4.3.24, nous avons les cas suivants :

(a) Supposons que $min_r(C_1) < 1$ ou $min_r(C_2) < 1$. Si $min_r(C_1) < 1$ et $min_r(C_2) < 1$, alors $min_r(C_1 \sqcap C_2) = max(min_r(C_1), min_r(C_2)) < 1$, et donc $C_1 \sqcap C_2 \not\sqsubset Exp$. Sans perte de généralité, supposons que $min_r(C_1) \geq 1$ et $min_r(C_2) < 1$. On a :
$min_r(C_1 \sqcap C_2) = max(min_r(C_1), min_r(C_2)) \geq 1$. Si $ex_r(C_1 \sqcap C_2) = \emptyset$, alors $ex_r(C_1) = \emptyset$, et donc $D' \neq \top$ (Théorème 4.3.24). Cela implique que $C_1 \sqcap C_2 \not\sqsubset Exp$. Si $ex_r(C_1 \sqcap C_2) \neq \emptyset$, alors $ex_r(C_1) \neq \emptyset$ et $ex_r(C_2) = \emptyset$. Selon le Théorème 4.3.24, il existe d'une application $\alpha \in \Gamma_r(C_1)$ telle que $C' \not\sqsubseteq D'$ pour tout $C' \in ex_r(C_1)^{\alpha}$. Puisque $ex_r(C_2) = \emptyset$, $m = |ex(C_1)| = |ex(C_1 \sqcap C_2)|$, alors $min(max_r(C_1), m) \geq min(max_r(C_1 \sqcap C_2), m)$. D'autre part, puisque $C \sqsubseteq C_1$, $C \sqsubseteq C_2$ et $max_r(C) = \infty$, alors $max_r(C_1) = max_r(C_2) = \infty$, et donc $m = min(max_r(C_1), m) = min(max_r(C_1 \sqcap C_2), m)$. Cela nous permet de construire une application $\alpha_1 \in \Gamma_r(C_1 \sqcap C_2)$ qui est identique au $\alpha \in \Gamma_r(C_1)$. On obtient : $C_1 \sqcap C_2 \not\sqsubseteq Exp$.

(b) Supposons que $min_r(C_1) \geq 1$ et $min_r(C_2) \geq 1$. On a : $min_r(C_1 \sqcap C_2) = max(min_r(C_1), min_r(C_2)) \geq 1$. Si $ex_r(C_1 \sqcap C_2) = \emptyset$, alors $ex_r(C_1) = \emptyset$, et donc $D' \neq \top$ (Théorème 4.3.24). Cela implique que $C_1 \sqcap C_2 \not\sqsubset$

Exp. Supposons que $ex_r(C_1 \sqcap C_2) \neq \emptyset$ et $ex_r(C_1) \neq \emptyset$, $ex_r(C_2) \neq \emptyset$. Selon le Théorème 4.3.24, il existe deux applications $\alpha_1 \in \Gamma_r(C_1)$ et $\alpha_2 \in \Gamma_r(C_2)$ telles que $C' \not\sqsubseteq D'$ et $C'' \not\sqsubseteq D'$ pour tout $C' \in ex_r(C_1)^{\alpha_1}$ et $C'' \in ex_r(C_2)^{\alpha_2}$. On a : $m = |ex(C_1)| + |ex(C_2)| = |ex(C_1 \sqcap C_2)|$ et $min(max_r(C_1 \sqcap C_2), m) = m$ car $max_r(C_1)$=$max_r(C_2) = \infty$ et $max_r(C_1 \sqcap C_2) = min(max_r(C_1), max_r(C_2))$. On peut construire une application $\alpha \in \Gamma_r(C_1 \sqcap C_2)$ sur $\{1..m\}$ où $\alpha(i) = \alpha_1(i)$ pour tout $i \in [1, .., |ex(C_1)|]$ et $\alpha(i) = \alpha_2(i)$ pour tout $i \in \{|ex(C_1)|+1, |ex(C_1)|+ |ex(C_2)|\}$. Il est évident que $C^x \not\sqsubseteq D'$ pour tout $C^x \in ex_r(C_1 \sqcap C_2)^{\alpha}$. Par le Théorème 4.3.24, on obtient : $C_1 \sqcap C_2 \not\sqsubseteq Exp$.

Dans tous les cas, le même argument dans 1.(a) permet d'obtenir $C_1 \equiv C_2$. □

Remarque 4.3.26. La condition $max_r(C) = \infty$ est importante pour assurer l'unicité de SEMI-OUBLIER. Nous considérons l'exemple suivant.

$$C := \exists r.(A \sqcap B \sqcap C) \sqcap (\leq 1r)$$
$$Exp := \exists r.(B \sqcap C)$$

On a : $max_r(C) = 1$ et donc,

$$\text{SEMI-OUBLIER} = \{\exists r.(A \sqcap B) \sqcap \exists r.(A \sqcap C) \sqcap (\leq 2r), \exists r.(A \sqcap B) \sqcap (\leq 1r), \exists r.(A \sqcap C) \sqcap (\leq 1r)\}.$$

La proposition 4.3.25 éclaircit certains aspects du calcul OUBLIER dans la langage $\mathcal{EL}\mathcal{N}^-$. Une procédure permettant de calculer tous les SEMI-OUBLIER peut être développée en se basant sur les résultats obtenus. Cependant, cette tâche fera l'objet de travaux ultérieurs.

Définition et Calcul de DIRE dans $\mathcal{FLE}$. Maintenant, nous essayons de définir l'opération DIRE de telle sorte que les deux opérateurs OUBLIER et DIRE vérifient le postulat de récupération (K6). C'est-à-dire que, approximativement, la connaissance qui est enlevée par OUBLIER peut être récupérée par DIRE. D'autre part, la modification de la définition d'un concept effectuée par DIRE doit être "minimale" de telle sorte que la connaissance ajoutée par DIRE doive être déduite de la nouvelle définition du concept. Notons que la satisfaction du postulat de récupération (K6) dans ce cas n'est pas triviale. Effectivement, par exemple, on considère les $\mathcal{EL}$-descriptions de concept suivantes :

$$C := \exists r.(A \sqcap B) \sqcap \exists r.(A \sqcap C)$$
$$Exp := \exists r.A$$

Selon l'Algorithme 4.3.9, on a :

$$\text{OUBLIER}(C,\ Exp) = \exists r.B \sqcap \exists r.C.$$

Cependant, si l'opération DIRE(*C,Exp*) est définie comme la plus générique description de concept par rapport à la subsomption telle que :

$$\text{DIRE}(C,Exp) \sqsubseteq C \text{ et } \text{DIRE}(C,Exp) \sqsubseteq Exp,$$

alors

$$\text{DIRE(OUBLIER(}C\text{, }Exp\text{)},}Exp\text{)}=\exists r.B \sqcap \exists r.C \sqcap \exists r.A,$$

et donc

$$C \sqsubset \text{DIRE(OUBLIER(}C\text{, }Exp\text{)},}Exp\text{)},$$

qui ne satisfait pas le principe de modification minimale (P5) (Cela implique le postulat de récupération (K6)).

Pour cette raison, une définition optimale et déclarative pour l'opération DIRE doit prendre en compte directement le principe (P5). Cela peut être réalisé à l'aide de l'opération OUBLIER.

Définition 4.3.27. (Opération DIRE) Soient C, D des $\mathcal{FLE}$-descriptions de concept normalisées, D une description de concept simple et $D \not\equiv \top$.

(1) Si $C \equiv \top$, DIRE(C, D) := D,

(2) Si $C \sqsubseteq D$, DIRE(C, D) := C,

(3) Si $C \not\sqsubseteq D$,

$$\text{DIRE}(C, D) := D \sqcap \bigsqcap C' \text{ pour toute } C' \text{ telle que OUBLIER}(C', D) \equiv C$$

Nous pouvons démontrer que DIRE(C, D) $\sqsubseteq C$ et DIRE(C, D) $\sqsubseteq D$. En effet, l'affirmation est évidente si $C \sqsubseteq D$. Supposons que $C \not\sqsubseteq D$. Par la définition OUBLIER 4.3.20, on a OUBLIER$(C, D) \equiv C$ car $C \not\sqsubseteq D$. Cela signifie que C est un conjoint dans la définition DIRE et D aussi. Par conséquent, DIRE(C, D) $\sqsubseteq C$ et DIRE(C, D) $\sqsubseteq D$.

De plus, la DIRE avec l'opération OUBLIER satisfait le principe de modification minimale (P5). En effet, si $C \not\sqsubseteq D$, alors DIRE(OUBLIER(C, D) , D) $\equiv$ DIRE(C , D) $\sqsubseteq C$. Supposons que $C \sqsubseteq D$. Puisque OUBLIER(C, D) = OUBLIER(C, D), alors par la définition DIRE, C est un conjoint de DIRE(OUBLIER(C, D) , D). Par conséquent, DIRE(OUBLIER(C, D) , D) $\sqsubseteq C$.

D'autre part, l'opération DIRE est optimale dans le sens où DIRE(C, D) est la description de concept la plus générique par rapport à la subsomption telle que :

- DIRE$(C, D) \sqsubseteq C$,
- DIRE$(C, D) \sqsubseteq D$, et
- S'il existe une description de concept C' telle que OUBLIER$(C', D) \equiv C$, alors DIRE(C, D) $\sqsubseteq C'$.

Pour préparer un algorithme qui calcule l'opération DIRE, on propose d'abord un algorithme lcs^{-1} permettant de retrouver une description de concept d'opérande à partir du lcs.

Algorithme 4.3.28. (lcs^{-1}) *Soient* C, D, E *des $\mathcal{FLE}$-descriptions de concept normalisées telles que $E \sqsubseteq D$ et $D, E \not\equiv \top$. Une $\mathcal{FLE}$-description de concept $lcs^{-1}(C, D, E)$ est définie comme suit :*

- *$lcs\{lcs^{-1}(C, D, E), D\} \equiv C$ et $E \sqsubseteq lcs^{-1}(C, D, E)$.*
- *s'il y a une $\mathcal{FLE}$-description de concept F' telle que $E \sqsubseteq F'$, $lcs(F', D) \equiv C$ et $F' \sqsubseteq lcs^{-1}(C, D, E)$, alors $F' \equiv lcs^{-1}(C, D, E)$.*

Alors, la description de concept $lcs^{-1}(C, D, E)$ *est calculée comme suit :*
Soient $\mathcal{C}$ =(N_C, E_C, c_0, l_C), $\mathcal{D}$ = (N_D, E_D, d_0, l_D) *et* $\mathcal{E}$ = (N_E, E_E, e_0, l_E)
Entrée : $\mathcal{C}$, $\mathcal{D}$, $\mathcal{E}$
Sortie : $lcs^{-1}(C, D, E) = F$
Soit $\mathcal{F}$ = (N_F, E_F, f_0, l_F)

(1) *Si* $|E| = 0$ *ou* $|D| = 0$ *, alors* $l_F(f_0) := l_C(c_0) \cup \{l_E(e_0) \setminus (l_D(d_0)\}$. *Si* $|E| > 0$ *et* $|D| > 0$, *alors*

(2) *Pour chaque r-successeur* ($\forall r$-*successeur*) c *de* c_0 *dans l'arbre* $\mathcal{C}$, *chaque r-successeur* ($\forall r$-*successeur*) d *de* d_0 *dans l'arbre* $\mathcal{D}$ *et chaque r-successeur* ($\forall r$-*successeur*) e *de* e_0 *dans l'arbre* $\mathcal{E}$ *tels que* $l_C(c) \subseteq l_D(d)$, $l_C(d) \subseteq l_E(e)$ *et* $C_{\mathcal{D}(d)} \sqsubseteq C_{\mathcal{C}(c)}$, $C_{\mathcal{D}(e)} \sqsubseteq C_{\mathcal{C}(d)}$, *on obtient un r-successeur* ($\forall r$-*successeur*) f *de* f_0 *dans l'arbre* $\mathcal{F}$ *tel que* $l_F(f) := l_C(c) \cup \{l_E(e) \setminus l_D(d)\}$ *et* f *est la racine du sous-arbre* $lcs^{-1}(C_{\mathcal{C}(v)}, C_{\mathcal{D}(d)}, C_{\mathcal{E}(e)})$.

(3) *Pour chaque r-successeur (*$\forall r$-*successeur) e de* e_0 *dans l'arbre* $\mathcal{E}$ *tel que* $l_C(c) \not\subseteq l_E(e)$ *pour tout r-successeur (*$\forall r$-*successeur) c de* c_0 *dans l'arbre* $\mathcal{C}$, *alors on obtient un r-successeur* ($\forall r$-*successeur*) f *de* f_0 *dans l'arbre* $\mathcal{F}$ *tel que* $l_F(f) := l_E(e)$ *et* f *est la racine de la copie du sous-arbre* $\mathcal{E}(e)$.

DÉMONSTRATION. D'abord, nous montrons l'existence unique de $lcs^{-1}(C, D, E)$ dans le cas où $lcs(C, D)$ est calculé dans l'Algorithme 4.3.17. En effet, supposons que Z_1, Z_2 soient deux $lcs^{-1}(C, D, E)$ *i.e.* $lcs(Z_1, D) = C$ et $lcs(Z_2, D) = C$. On a : $E \sqsubseteq Z_1 \sqcap Z_2$. De plus, puisque $E \sqsubseteq Z_1$ et $E \sqsubseteq Z_2$ et D est obtenue de E par l'Algorithme 4.3.17, alors toute propagation entre une restriction universelle et une restriction existentielle dans E, donc dans $Z_1 \sqcap Z_2$, est également appliquée dans D. Par conséquent, on a : $lcs(Z_1 \sqcap Z_2, D) \equiv C$.

Nous montrons que l'Algorithme 4.3.28 permet de calculer $lcs^{-1}(C, D, E)$. Supposons qu'une $\mathcal{FLE}$-description de concept X ait les propriétés suivantes : $lcs\{X, D\} \equiv C$ et $E \sqsubseteq X$. Nous montrons que $F \sqsubseteq X$ où F est construite de l'Algorithme 4.3.28. Soit $\mathcal{X}$ = (N_X, E_X, x_0, l_X).

Selon la procédure de calcul lcs, on a : $l(x_0) \cap l(d_0) = l(c_0)$. De plus, on a : $l(x_0) \subseteq l(e_0)$. Supposons que $P \in l(x_0)$. Si $P \in l(c_0)$, alors $P \in l(f_0)$. Si $P \notin l(c_0)$, alors $P \notin l(d_0)$ et donc, $P \in l_E(e_0) \setminus l_D(d_0)$. Par conséquent, $l(x_0) \subseteq l(f_0)$.

Soit x un r-successeur de x_0. Selon la procédure de calcul lcs, on désigne par d et c des r-successeurs de d_0 et de c_0 tels que $l(x) \cap l(d) = l(c)$, $lcs\{C_{\mathcal{X}(x)}, C_{\mathcal{D}(d)}\} \equiv C_{\mathcal{C}(c)}$ et $l_X(x) \subseteq l_E(e)$ et $C_{\mathcal{E}(e)} \sqsubseteq C_{\mathcal{X}(x)}$ où e un r-successeurs de e_0. Selon l'étape (2) de l'Algorithme 4.3.28, il existe un r-successeur f de f_0 dans l'arbre $\mathcal{F}$ tel que $l(x) \subseteq l(f)$. De plus, selon toujours l'étape (2) de l'Algorithme 4.3.28, on obtient également $lcs\{C_{\mathcal{F}(f)}, C_{\mathcal{D}(d)}\} \equiv C_{\mathcal{C}(c)}$. Par l'hypothèse de récurrence, on a : $C_{\mathcal{F}(f)} \sqsubseteq C_{\mathcal{X}(x)}$. De même argument, on peut montrer qu'il existe un $\forall r$-successeur f de f_0 dans l'arbre $\mathcal{F}$ tel que $l(x) \subseteq l(f)$ et $C_{\mathcal{F}(f)} \sqsubseteq C_{\mathcal{X}(x)}$ où x un $\forall r$-successeur de x_0. □

En basant sur les algorithmes de calcul lcs^{-1} et de calcul OUBLIER, nous présentons l'algorithme de calcul de l'opération DIRE comme suit.

Algorithme 4.3.29. (*Opérateur* PLUS) *Soient* C, D *des* $\mathcal{FLE}$*-descriptions de concept normalisées,* D *est une description de concept simple. La description de*

concept PLUS(C, D) *est calculée comme un arbre de description obtenu d'une fusion des deux arbres* $\mathcal{G}_\mathcal{C}$ *et* $\mathcal{G}_\mathcal{D}$. *Plus formellement, soient* $\mathcal{G}$ =(N_C, E_C, v_0, l_C), $\mathcal{H}$ = (N_D, E_D, w_0, l_D) *et* $(\mathcal{G}+\mathcal{H}) = \mathcal{G}_{PLUS(C,D)} = (N_{C+D}, E_{C+D}, u_0, l_{C+D})$
Entrée *:* $\mathcal{G}$, $\mathcal{H}$
Sortie *:* $(\mathcal{G}+\mathcal{H}) = \mathcal{G}_{PLUS(C,D)}$
Si $C_\mathcal{G} \sqsubseteq C_\mathcal{H}$, *alors* $(\mathcal{G}+\mathcal{H}) := \mathcal{G}$. *Sinon,*

(1) *Si* $|C| = 0$ *ou* $|D| = 0$, *alors* $\mathcal{G}_{PLUS(C,D)} := \mathcal{G}_{C\sqcap D}$;

(2) *Sinon,* $(\mathcal{G}+\mathcal{H})$ *est calculé par récurrence sur la profondeur des arbres. Le nœud* (v_0, w_0) *étiqueté par* $l_G(v_0) \cup l_H(w_0)$ *est la racine de* $(\mathcal{G}+\mathcal{H})$.

(a) *Si le successeur* w *de* w_0 *est un* $\forall r$*-successeur, alors* $\mathcal{G}_{PLUS(C,D)} := \mathcal{G}_{C\sqcap D}$, *sinon*

(b) *S'il n'existe pas de* $r-$*successeur* v *de* v_0 *dans* $\mathcal{G}$, *alors* $\mathcal{G}_{PLUS(C,D)} := \mathcal{G}_{C\sqcap D}$, *sinon*

(c) *Soit* $\mathcal{T}$ *un arbre de description tel que l'étiquette de la racine* $l_{C_\mathcal{T}}(t_0)$ *soit définie comme* $l_G(v_0) \cup l_H(w_0)$. *Si le successeur* w *de* w_0 *est un* r*-successeur, alors pour chaque* r*-successeur* v *de* v_0 *dans* $\mathcal{G}$ *et chaque successeur* w *de* w_0 *dans* $\mathcal{H}$, *on obtient*

(i) *pour chaque* $P \in l_D(w) \setminus l_C(v)$, *on obtient un* r*-successeur* t *de* t_0 *dans* $\mathcal{T}$ *tel que* $l_{C_\mathcal{T}}(t) := l_C(v) \cup \{P\}$ *et le nœud* t *soit la racine de la copie du sous-arbre* $\mathcal{G}(v)$ *si* $l_D(w) \not\subseteq l_C(v)$ *et* $l_D(w) \setminus l_C(v) \neq \emptyset$. *De plus, on obtient également un* r*-successeur* t *de* t_0 *dans* $\mathcal{T}$ *tel que* $l_{C_\mathcal{T}}(t) := l_C(v)$ *et le nœud* t *soit la racine de la copie du sous-arbre* $\mathcal{G}(v)$ *si* $l_D(w) \not\subseteq l_C(v)$. *Sinon,*

(ii) *un* r*-successeur* t *de* t_0 *dans* $\mathcal{T}$ *tel que* $l_{C_\mathcal{T}}(t) := l_C(v)$ *et le nœud* t *soit la racine du sous-arbre* $\mathcal{G}(v) + \alpha.\mathcal{H}(w')$ *où* w' *est un* $\alpha-$*successeur de* w, $\alpha \in \{r, \forall r\}$.

Si $C_{\mathcal{T}-\mathcal{H}} \not\equiv \bigsqcap_{C' \in ex(C)} C'$, *alors* $\mathcal{G}+\mathcal{H} := \mathcal{G}_{C\sqcap D}$, *sinon* $\mathcal{G}+\mathcal{H} := \mathcal{G}_{C_\mathcal{T}\sqcap D}$

(d) *Le* $\forall r$*-successeur* u' *de* u_0 *est la racine du sous-arbre*

$$lcs^{-1}(\mathcal{G}(v'),\ lcs\{C' | C' \in ex(C_\mathcal{G})\},\ lcs\{E' | E' \in ex(C_{\mathcal{G}+\mathcal{H}})\})$$

Exemple 4.3.30. Nous reprenons l'Exemple 4.3.14.
$C := \exists r.(A \sqcap B \sqcap \forall r.(A \sqcap B) \sqcap \exists r.(A \sqcap B \sqcap C))$,
$Exp := \exists r.(A \sqcap \forall r.A \sqcap \exists r.(A \sqcap B))$
DIRE(OUBLIER(C,Exp), Exp) $= \exists r.(A \sqcap B \sqcap \forall r.(A \sqcap B) \sqcap \exists r.(A \sqcap B \sqcap C))$
La Figure 4.3.3 illustre le calcul de l'Exemple 4.3.30 selon l'Algorithme 4.3.29.

Le théorème suivant établit la correction de l'Algorithme 4.3.29.

Théorème 4.3.31. *Soient* C, D *des* $\mathcal{FLE}$*-descriptions de concept normalisées,* D *une description de concept simple et* $D \not\equiv \top$. *On a :*

$$DIRE(C,\ D) \equiv PLUS(C,\ D)$$

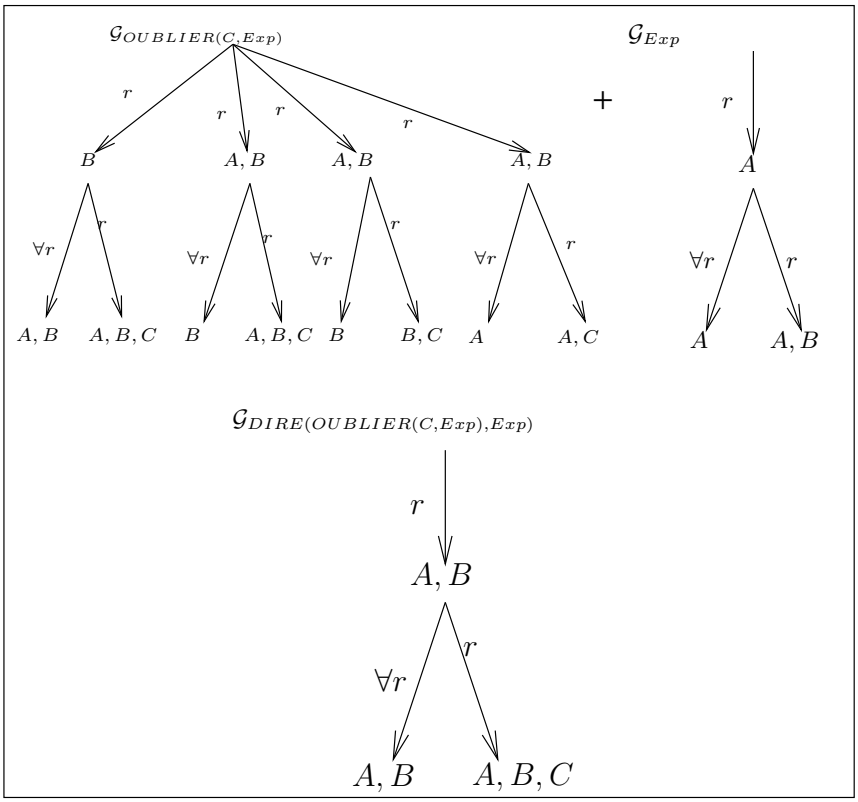

FIG. 4.3.3. Arbres de Description pour le calcul DIRE

DÉMONSTRATION. Soient $\mathcal{G} = (N_C,\ E_C,\ v_0,\ l_C)$, $\mathcal{H} = (N_D,\ E_D,\ w_0,\ l_D)$ et $(\mathcal{G} + \mathcal{H}) = \mathcal{G}_{PLUS(C,D)} = (N_{C+D},\ E_{C+D},\ u_0,\ l_{C+D})$. Si $C \sqsubseteq D$, alors DIRE(C, D) = PLUS(C, D) = C. Supposons que $C \not\sqsubseteq D$. L'affirmation est vérifiée pour $|\mathcal{G}_C| = 0$. Supposons que $|\mathcal{G}_C| > 0$.
Si le successeur w de w_0 est un $\forall r$-successeur ou v_0 n'a qu'un $\forall r$-successeur v, l'affirmation du théorème est vérifiée. Supposons que le successeur w de w_0 est un r-successeur et v_0 a au moins un r-successeur.

Dans l'étape 2.(c) de l'Algorithme 4.3.29, on considère le cas $C_{\mathcal{T}-\mathcal{H}} \not\equiv \bigsqcap_{C' \in ex(C)} C'$. Supposons qu'il existe une $\mathcal{FLE}$-description de concept normalisée X (qui ne contient pas de restriction universelle au top-level) telle que OUBLIER(X,D) $\equiv \bigsqcap_{C' \in ex(C)} C'$ et $X \sqsubseteq D$. Montrons que $C_{\mathcal{T}-\mathcal{H}} \equiv \bigsqcap_{C' \in ex(C)} C'$. En effet,

Grâce à la propriété de l'opération OUBLIER, afin de montrer $\bigsqcap_{C' \in ex(C)} C' \sqsubseteq C_{\mathcal{T}-\mathcal{H}}$, il suffit de montrer que $X \sqsubseteq C_{\mathcal{T}-\mathcal{H}}$. Par l'Algorithme 4.3.29, pour chaque $E' \in ex(C_{\mathcal{T}-\mathcal{H}})$ créée par l'Algorithme MINUS 4.3.17, il existe $C' \in ex(C_{\mathcal{T}})$ et un nœud $v' \in V_T$ situé dans $\mathcal{G}_{C'}$ et des nœuds $w' \in V_H$, $u' \in V_{T-H}$ tels que $C' \sqsubseteq E'$ et $l(u')$=$l(v') \setminus \{P\}$; $P \in l(v') \cap l(w')$. Par l'étape 2.(c).(i) de l'Algorithme 4.3.29, il existe $C'' \in ex(C)$ telle que $C' \sqsubseteq C''$ et un nœud $v \in V_G$ situé dans $\mathcal{G}_{C''}$, des nœuds $w \in V_H$, $t \in V_T$ tels que $l(t)$=$l(v) \cup \{P\}$; $P \in l(v) \setminus l(w)$. Donc, il existe $X' \in ex(X)$ telle que $X' \sqsubseteq E'$ car $X \sqsubseteq C$. De plus, pour chaque $E' = r.C_{\mathcal{G}_{E''}} \in ex(C_{\mathcal{T}-\mathcal{H}})$ où $E'' \in$

$ex(C_{\mathcal{T}(t)-\alpha.\mathcal{H}(w')})$ créée par l'étape 2.(c).(ii) de l'Algorithme 4.3.29, par l'hypothèse de récurrence, il existe $X'' \in ex(C_{\mathcal{G}_X(x)})$ telle que $C_{r.\mathcal{G}_{X''}} \sqsubseteq E'$. Par conséquent, on obtient $X \sqsubseteq C_{\mathcal{T}-\mathcal{H}}$. Cela implique que $\bigsqcap_{C' \in ex(C)} C' \sqsubseteq C_{\mathcal{T}-\mathcal{H}}$ (selon la propriété de l'opération OUBLIER). D'autre part, on a également $C_{\mathcal{T}-\mathcal{H}} \sqsubseteq \bigsqcap_{C' \in ex(C)} C'$ car $C_{\mathcal{T}} \sqsubseteq \bigsqcap_{C' \in ex(C)} C'$ et $\bigsqcap_{C' \in ex(C)} C' \not\sqsubseteq D$.

Nous avons montré que $C_{\mathcal{T}-\mathcal{H}} \equiv \bigsqcap_{C' \in ex(C)} C'$ qui contredit $C_{\mathcal{T}-\mathcal{H}} \not\equiv \bigsqcap_{C' \in ex(C)} C'$. Cela implique qu'il n'existe pas de $\mathcal{FLE}$-description de concept normalisée X telle que OUBLIER(X,D) $\equiv \bigsqcap_{C' \in ex(C)} C'$ et $X \sqsubseteq D$. Par conséquent, $\mathcal{G} + \mathcal{H} \equiv \mathcal{G}_{C \sqcap D}$.

Dans l'étape 2.(c) de l'Algorithme 4.3.29, on considère le cas $C_{\mathcal{T}-\mathcal{H}} \equiv \bigsqcap_{C' \in ex(C)} C'$. Nous montrons que $C_{\mathcal{T}} \sqsubseteq X$ pour toute $\mathcal{FLE}$-description de concept normalisée X (qui ne contient pas de restriction universelle au top-level) telle que OUBLIER(X,D) $\equiv \bigsqcap_{C' \in ex(C)} C'$ et $X \sqsubseteq D$. En effet, pour chaque $X' \in ex(X)$, par l'Algorithme MINUS 4.3.17 , il existe $C' \in ex(C)$ et un nœud $v \in V_G$ situé dans $\mathcal{G}_{C'}$ et des nœuds $w \in V_H$, $x' \in V_{X'}$ tels qu'il existe un isomorphisme entre les deux arbres $\mathcal{G}_{\mathcal{C}'}$ et $\mathcal{X}'$ sauf que $l(x')$=$l(v) \setminus \{P\}$; $P \in l(v) \cap l(w)$. Par l'étape 2.(c).(i) de l'Algorithme 4.3.29, il existe $E' \in ex(C_{\mathcal{T}})$ telle que $E' \sqsubseteq X'$ et un nœud $t \in V_T$ situé dans $\mathcal{G}_{E'}$, des nœuds $w \in V_H$, $v \in V_G$ tels que $l(t)$=$l(v) \cup \{P\}$; $P \in l(v) \setminus l(w)$. Par conséquent, pour chaque $X' \in ex(X)$ il existe $E' \in ex(C_{\mathcal{T}})$ telle que $E' \sqsubseteq X'$.

Finalement, selon la propriété de lcs^{-1}, le sous-arbre $(\mathcal{G} + \mathcal{H})(u')$ obtenu de l'étape 2.(d) est le plus spécifique tel que $C_E \sqsubseteq C_{(\mathcal{G}+\mathcal{H})(u)}$ où $E = \bigsqcap_{E' \in ex(\mathcal{G}+\mathcal{H})} E'$ et u' est le $\forall r$-successeur de u_0. Cela implique que $C_{(\mathcal{G}+\mathcal{H})(u')} \sqsubseteq X'$ où $X' = var(X)$ et OUBLIER(X,D) $\equiv C$. □

4.3.3. Révision de l'ABox. Dans les sous-sections précédentes, nous avons étudié la révision du TBox pour le langage $\mathcal{FLE}$ et défini les opérations de révision OUBLIER et DIRE pour les concepts définis dans le TBox. Nous avons également proposé les algorithmes qui permettent de calculer les descriptions de concepts caractérisées par les définitions des opérations de révision. Dans cette sous-section nous considérons le problème de la révision de l'ABox lorsque le TBox est révisé. Selon la discussion dans la section 4.1, une expansion d'un TBox *i.e.* l'ajout d'un nouveau concept au TBox n'exige pas de révision de l'ABox correspondant. En effet, toute assertion de l'ABox est toujours vérifiée suivant un nouveau concept ajouté. De même, l'opération OUBLIER(*C, Exp*) qui est définie dans la section 4.3 modifie la définition du concept C de telle sorte que si un individu vérifie la définition du concept C, alors cet individu vérifie également la nouvelle définition de C. Cela implique qu'aucune révision n'est nécessaire après avoir révisé le concept C par OUBLIER(*C, Exp*). Toutefois, dans tous les cas certaines nouvelles assertions doivent être ajoutées à l'ABox. Par exemple, pour la révision OUBLIER(*C, Exp*) les assertions, qui ne vérifient pas le concept C mais vérifient le concept OUBLIER(*C, Exp*), peuvent être ajoutées.

En revanche, l'opération DIRE(*C, Exp*) qui est définie dans la section 4.3 exige une révision de l'ABox. Il peut exister un individu qui vérifie le concept C mais ne vérifient pas le concept DIRE(*C, Exp*).

Soit C une définition de concept modifiée dans le TBox par une opération de révision. Soit $(\mathcal{O}, .^{\mathcal{I}})$ un modèle de la base de connaissances $\Delta = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}})$ où $\Delta_{\mathcal{T}}$ est un TBox et $\Delta_{\mathcal{A}}$ est l'ABox correspondant. Pour chaque assertion $C^I(d^I)$ où $d^I \in \mathcal{O}$, nous pouvons utiliser l'inférence de *vérification d'instance* **[BCM$^+$03 (8)]** pour déterminer si l'assertion $C^I(d^I)$ est vérifié *i.e.* $\Delta \models (DIRE(C, Exp))^I(d^I)$. En outre, pour les TBox non-dépliés, il est nécessaire de réviser également toutes les assertions $D^I(d^I)$ où D qui est défini via le concept C révisé.

4.4. Règle de Révision

4.4.1. Définition. La règle qui sera définie ci-dessous est une forme spécifique de règles de production, appelée *règle de révision* . En effet, le conséquent de ces règles est une opération de révision. La forme générale de ces règles dans la base de connaissances $\Delta = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ est définie comme suit :
$r \in \Delta_{\mathcal{R}}$
$r : p \Rightarrow$ DIRE($\Delta_{\mathcal{T}}$, *Concept*, *Exp*) ou
$r : p \Rightarrow$ OUBLIER($\Delta_{\mathcal{T}}$, *Concept*, *Exp*)
où p est un prédicat formé d'assertions dans $\Delta_{\mathcal{A}}$ avec le constructeur de *conjonction d'assertion* :
$p := C_1(d_1) \wedge \ldots \wedge C_n(d_n)$ où $C_i(d_i) \in \Delta_{\mathcal{A}}$. Les opérations de révision DIRE et OUBLIER ont été définies dans la section 4.3.
Nous désignons par $\Delta_{\mathcal{R}}$ l'ensemble des règles de révision qui est considéré comme le troisième composant de la base de connaissances : $\Delta = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$.

Exemple 4.4.1. (Formalisation de l'Exemple d'Introduction)
Soit $\Delta_{\mathcal{T}}$:
$\Delta_{\mathcal{T}} := \{$
EntrepriseEurop := $\exists$*associer*.**Européen** *;*
EntrepriseAmér := $\exists$*associer*.**Américain** *;*
ProdRésPolluant := **Produit** $\sqcap$ $\exists$*résister*.**Bruit**
$\}$
Les concepts **Produit**, **Polluant**, **Européen**, **Américain** sont des concepts primitifs. Les rôles *associer*, *résister*, *acheté-par*, *vendu-par* sont les rôles primitifs. Le concept **EntrepriseEurop** (respectivement, **EntrepriseAmér**) est défini comme un ensemble d'individus qui possède un *associé* **Européen** (respectivement, **Américain**). Le concept **ProdRésPolluant** est défini comme un **Produit** capable de *résister* au **Bruit**.
Soient $\Delta_{\mathcal{A}}$ des faits :
$\Delta_{\mathcal{A}}$:=$\{$*vendu-par*(a,b), *acheté-par*(a,c), **ProdRésPolluant***(a),* **EntrepriseEurop**(b), **EntrepriseAmér**(c), **Polluant**(*Bruit*) $\}$
Le fait *vendu-par*(a,b) signifie que le produit a est vendu par l'entreprise b. Le fait *acheté-par*(a,c) signifie que le produit a est acheté par l'entreprise c.
Soit $\Delta_{\mathcal{R}}$ comporte une règle de révision *:*

$$\Delta_{\mathcal{R}} := \begin{array}{ll} \{r_1 : & \textit{vendu-par}(X, Y_1) \wedge \textit{acheté-par}(X, Z_1) \wedge \\ & \textit{associer}(Y_1, Y_2) \wedge \textit{associer}(Z_1, Z_2) \wedge \\ & \textbf{Européen}(Y_2) \wedge \textbf{Américain}(Z_2) \\ & \Rightarrow \text{DIRE}(\textbf{ProdRésPolluant}, \exists \textit{résister}.\textbf{Feu}) \ \} \end{array}$$

La règle r est une règle de révision dit que : si un produit X est vendu par une entreprise Y_1 qui a un associé **Européen**, le produit X est acheté par une entreprise Z_1 qui un associé **Américain**, et le produit X est un produit **ProdRésPolluant**, alors le concept **ProdRésPolluant** doit être interprété comme un produit capable de résister à la fois au **Feu** et au **Bruit**.
Les faits **EntrepriseEurop**(b) et **EntrepriseEurop** := $\exists$*associer*.**Européen** impliquent *associer*(b, b') et **Européen**(b') .
Les faits **EntrepriseAmér**(c) et **EntrepriseAmér** := $\exists$*associer*.**Américain** impliquent *associer*(c, c') et **Américain**(c'). Donc, la condition de la règle r est vérifiée.

4.4.2. Sémantique du formalisme hybride. Maintenant nous pouvons définir la sémantique du formalisme hybride qui résulte de la combinaison des règles de révision et la logique de description $\mathcal{FLE}$. Traditionnellement, une base de connaissances basée sur la Logique de Description comporte deux composants : TBox et ABox. La sémantique de ces deux composants est normalement définie (*cf.* Chapitre 1). Une définition de la sémantique du composant de règles de révision résulte de la sémantique des opérations de révision qui est bien introduite dans la section 4.3.

Soit Δ une base de connaissances composée de trois composants $\Delta = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$. Une interprétation $\mathcal{I}$ est un couple $(\mathcal{O}, \cdot^{\mathcal{I}})$ où $\mathcal{O}$ est un ensemble non-vide et une fonction $\cdot^{\mathcal{I}}$ transforme chaque constante a dans Δ en un objet $a^{\mathcal{I}} \in \mathcal{O}$. Une interprétation $\mathcal{I}$ est un modèle de Δ si elle est un modèle de chaque composant de Δ. Les modèles du composant TBox (terminologie) $\Delta_{\mathcal{T}}$ sont normalement définies comme les modèles des logiques de description. Une interprétation $\mathcal{I}$ est un modèle d'une assertion $C(d)$ dans $\Delta_{\mathcal{A}}$ si $d^{\mathcal{I}} \in C^{\mathcal{I}}$.
Une interprétation $\mathcal{I}$ est un modèle d'une règle $r : p \Rightarrow$ DIRE(C, *Exp*) si, lorsque la fonction $\cdot^{\mathcal{I}}$ transforme les individus $d^{\mathcal{I}} \in C_i^{\mathcal{I}}$ dans le domaine $\mathcal{O}$ pour chaque concept C_i dans l'antécédent de la règle r, alors

– Si $a^{\mathcal{I}} \in C^{\mathcal{I}}$ alors $a^{\mathcal{I}} \in (\text{DIRE}(C, Exp))^{\mathcal{I}}$

Une interprétation $\mathcal{I}$ est un modèle de $r : p \Rightarrow$ OUBLIER(C, *Exp*) si, lorsque la fonction $\cdot^{\mathcal{I}}$ transforme les individus $d^{\mathcal{I}} \in C_i^{\mathcal{I}}$ dans le domaine $\mathcal{O}$ pour chaque concept C_i dans l'antécédent de la règle r, alors

– Si $a^{\mathcal{I}} \in C^{\mathcal{I}}$ alors $a^{\mathcal{I}} \in (\text{OUBLIER}(C, Exp))^{\mathcal{I}}$

Remarque 4.4.2. La sémantique du composant $\Delta_{\mathcal{R}}$ qui est définie ci-dessus porte sur la sémantique de la *règle procédurale* $C \Rightarrow D$ où C, D sont les concepts, et sur les opérations de révision. La différence importante entre la règle de révision et la règle procédurale est qu'un modèle $\mathcal{I}$ de la base de connaissances Δ peut ne plus être un modèle de Δ après l'application de la règle de révision r. Cela implique que la sémantique présentée est au-delà de la Logique Classique. C'est pourquoi

l'extension "procédurale" de la base de connaissances Δ par l'application des règles du composant $\Delta_{\mathcal{R}}$ est considérée comme la transformation d'états de Δ plutôt que comme une inférence dans le cadre de la Logique Classique.

La Section 4.5 propose une "stratégie" de transformations de la base de connaissances effectuée par les règles de révision.

4.5. Inférence sur le formalisme hybride

Dans la section précédente nous avons défini les opérations de révision sur la base de connaissances basée sur la Logique de Description comportant deux composants, appelés TBox et ABox. Nous avons également proposé un nouveau formalisme hybride dont le troisième composant comporte les règles de révision définies à partir des opérations de révision. L'objectif de cette section est d'introduire dans le formalisme hybride une procédure d'inférence qui permet l'extension de la base de connaissances effectuée par les règles de révision. En d'autres termes, cette procédure nous permet d'exploiter les connaissances formalisées dans le composant $\Delta_{\mathcal{R}}$ pour obtenir l'harmonie entre les ontologies des acteurs dans un contexte d'échanges de données.

Définition 4.5.1. Soit $\Delta := (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ une base de connaissances où $\Delta_{\mathcal{R}} := \{r_1,...,r_n\}$. Soit $r \in \Delta_{\mathcal{R}}$ et $r = (C_1(d_1) \wedge ... \wedge C_k(d_k) \Rightarrow REV(C, Exp))$ où $REV \in$ {DIRE,OUBLIER}. On désigne $Ant(r) := \{C_1, ..., C_k\}$, $Cons(r) := \{C\}$, $Rev(r) :=$ {REV} et $Ex(r) := \{Exp\}$. La règle r est appelée *acyclique* si les conditions suivantes sont satisfaites :

(1) $Ex(r)$ est immodifiable *i.e.* $Ex(r)$ ne dépend d'aucun concept C à réviser. Un concept C dépend d'un concept D si C est directement ou indirectement défini via D.

(2) Chaque $C_i \in Ant(r)$, $i \in \{i, ..., k\}$, ne dépend pas de $Cons(r)$.

Remarque 4.5.2. Si $\Delta_{\mathcal{T}}$ est acyclique et $r \in \Delta_{\mathcal{R}}$ est acyclique pour toute $r \in \Delta_{\mathcal{R}}$, alors $\Delta_{\mathcal{R}}$ reste acyclique après l'application de r. En effet, si $Rev(r) =$ OUBLIER, r n'ajoute aucune dépendance au $\Delta_{\mathcal{T}}$ (Définition de OUBLIER). Par conséquent, $\Delta_{\mathcal{T}}$ et $\Delta_{\mathcal{R}}$ restent toujours acycliques. Si $Rev(r)$=DIRE, alors DIRE($Cons(r)$, $Ex(r)$) peut être dépendant des concepts X dont $Ex(r)$ dépend. Pourtant, X ne dépend pas de $Cons(r)$ pour toute règle $r \in \Delta_{\mathcal{R}}$, donc l'application de la règle r n'ajoute pas d'un cycle de dépendance au $\Delta_{\mathcal{T}}$ et au $\Delta_{\mathcal{R}}$.

Définition 4.5.3. Soit $\Delta := (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ une base de connaissances où $\Delta_{\mathcal{R}} :=$ {r_1,...,r_n} est un ensemble des règles de révision. $\Delta_{\mathcal{R}}$ est *non-récursif* s'il n'existe pas de cycle des règles *i.e.* il n'existe pas un sous-ensembles R $= \{r_{i_1}, ..., r_{i_l}\} \subseteq \Delta_{\mathcal{R}}$ tel que :

- Il existe $C_{i_2} \in Ant(r_{i_2})$, C_{i_2} dépend de $Cons(r_{i_1})$
- ...
- Il existe $C_{i_l} \in Ant(r_{i_l})$, C_{i_l} dépend de $Cons(r_{i_{l-1}})$, et

– Il existe $C_{i_1} \in Ant(r_{i_1})$, C_{i_1} dépend de $Cons(r_{i_l})$.

Remarque. La transformation de la $\Delta = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ par une règle $r \in \Delta_{\mathcal{R}}$ préserve la non-récursivité.

Les définitions 4.5.1, 4.5.3 nous permettent d'éviter les cas bouclés qui peuvent être causés par des applications de règles. D'autre part, la procédure d'inférence sur le formalisme hybride (extension de la base de connaissances effectuée par les règles) nécessite une certaine condition afin d'assurer la terminaison. L'exemple suivant montre que le problème de terminaison de la procédure d'inférence n'est pas trivial.

Exemple 4.5.4. Soient C une $\mathcal{FLE}$-description de concept suivantes :
$C := \forall r.(A \sqcap C) \sqcap \exists r.(A \sqcap B \sqcap C)$
et r_1, r_2 des règles de révision
$r_1 : p_1 \Rightarrow$OUBLIER(C, $Exp1$)
$r_2 : p_2 \Rightarrow$DIRE(C, $Exp2$) où $Exp1 := \exists r.(A \sqcap B)$ et $Exp2 := \forall r.A$
On a : $C \sqsubseteq Exp1$, alors OUBLIER$(C, Exp1) = \forall r.C \sqcap \exists r.(B \sqcap C) \sqcap \exists r.(A \sqcap C)$.
Donc, on a : OUBLIER$(C, Exp1) \not\sqsubseteq Exp2$, alors
DIRE(OUBLIER$(C, Exp1)$, $Exp2$) $= \forall r.(A \sqcap C) \sqcap \exists r.(B \sqcap C \sqcap A) \sqcap \exists r.(A \sqcap C)$,
et donc, DIRE(OUBLIER$(C, Exp1)$, $Exp2$) $\equiv C$
Cela implique que la terminaison de la procédure d'extension de la base de connaissances effectuée par r_1, r_2 n'est pas assurée.

L'Exemple 4.5.4 suscite une idée sur la condition sous laquelle la terminaison de la procédure d'inférence est assurée. Effectivement, la non-terminaison dans l'Exemple 4.5.4 vient du fait $Exp1 \sqsubseteq Exp2$. Cette condition sur l'expression de requête de révision est généralisée dans la définition suivante.

Définition 4.5.5. Soient r_1, r_2 des règles de révision.

(1) Supposons que $Cons(r_1) = Cons(r_2)$. La notation $Ex(r_1) \not\gtrsim Ex(r_2)$ désigne un des faits suivants :

(a) $Ex(r_1)$ et $Ex(r_2)$ sont les restrictions existentielles et $Ex(r_1) \not\sqsubseteq Ex(r_2)$, $Ex(r_2) \not\sqsubseteq Ex(r_1)$.

(b) $Ex(r_1)$ est une restriction existentielle, $Ex(r_2) = \underbrace{\forall r...\forall r.}_{n} E'$ où $E' \in \{P, \exists r.D'\}$, $P \in N_C$; D' est une $\mathcal{FLE}$-description de concept, et $E' \not\sqsubseteq E''$, $E'' \not\sqsubseteq E'$ pour tout E'' où l'arbre de description $\mathcal{G}_{E''}$ est un sous-arbre au niveau n de l'arbre de description $\mathcal{G}_{Ex(r_1)}$.

(c) $Ex(r_1)$ et $Ex(r_2)$ sont des restrictions universelles, $Ex(r_1) = \underbrace{\forall r...\forall r.}_{m} E_1$, $Ex(r_2) = \underbrace{\forall r...\forall r.}_{n} E_2$ où $E_1 \in \{P, \exists r.D_1\}$, $E_2 \in \{Q, \exists r.D_2\}$, $P, Q \in N_C$, $m \leq n$, D_1, D_2 sont des $\mathcal{FLE}$-descriptions de concept et

$\exists r.D_1 \not\geqslant \underbrace{\forall r...\forall r.}_{n-m} E_2$. Si $m = n$ ou $m < n$, ces cas correspondent respectivement à 1.(a) et à 1.(b).

(2) Supposons que $Cons(r_1)$ dépend de $Cons(r_2)$ et $Cons(r_2)$ apparaît au niveau n dans $Cons(r_1)$ *i.e.* l'arbre de description $\mathcal{G}_{Cons(r_2)}$ est un sous-arbre au niveau n dans l'arbre de description $\mathcal{G}_{Cons(r_1)}$. On désigne par $(c_1, ..., c_n)$, $c_i \in \{\forall r., \exists r.\}$ un n-chemin de la racine du $\mathcal{G}_{Cons(r_1)}$ vers la racine du $\mathcal{G}_{Cons(r_2)}$. La notation $Ex(r_1) \not\geqslant Ex(r_2)$ désigne le fait que :

$$Ex(r_1) \not\geqslant (c_1, ..., c_n).Ex(r_2) \text{ pour tout } n\text{-chemin } (c_1, ..., c_n)$$

Notons que $Ex(r_1) \not\geqslant Ex(r_2)$ implique $Ex(r_1) \not\sqsubseteq Ex(r_2)$ et $Ex(r_2) \not\sqsubseteq Ex(r_1)$ mais le contraire n'est pas vérifié. Nous avons besoins de la notion "$\not\geqslant$" qui est plus stricte que "$\not\sqsubseteq$" car cette dernière ne prend pas en compte la propagation de la restriction universelle sur la restriction existentielle. C'est à dire qu'un ajout de $\forall r.D'$ par DIRE peut neutraliser une suppression de $\exists r.D'$ par OUBLIER malgré $\exists r D' \not\sqsubseteq \forall r.D'$ et $\forall r.D' \not\sqsubseteq \exists r.D'$.

Proposition 4.5.6. *Soit C une $\mathcal{FLE}$-description. On a les affirmations suivantes :*

(1) OUBLIER(DIRE($C, Exp1$), $Exp2$) $\sqsubseteq Exp1$ *si* $Exp1 \not\geqslant Exp2$

(2) *Soient r_1, r_2 des règles de révision telles que $REV(r_1)$=$REV(r_1)$=*OUBLIER*, $Cons(r_1)$ dépende de $Cons(r_2)$, $Cons(r_2)$ apparaîte au niveau $n \geq 0$ dans $Cons(r_1)$ et $Ex(r_1) \not\geqslant Ex(r_2)$.*

Si $Cons(r_1) \sqsubseteq Ex(r_1)$ alors $Cons_{[r_2]}(r_1) \sqsubseteq Ex(r_1)$ où $Cons_{[r_2]}(r_1)$ désigne la description de concept $Cons(r_1)$ obtenue de l'application de la règle r_2.

DÉMONSTRATION. La preuve s'appuie sur les procédures de calcul de OUBLIER et de DIRE.

(1) D'après la Définition DIRE **??**, on a : DIRE(C, Exp_1) $\sqsubseteq Exp_1$.
Si DIRE(C, Exp_1) $\not\sqsubseteq Exp_2$, alors
OUBLIER(DIRE($C, Exp1$), $Exp2$) $\equiv$ DIRE($C, Exp1$) $\sqsubseteq Exp_1$.
Supposons que DIRE(C, Exp_1) $\sqsubseteq Exp_2$. Ce cas correspond au cas exclus dans la partie (3) de la Remarque 4.3.21. Pourtant, la condition $Exp1 \not\geqslant Exp2$ qui est plus forte que $Exp1 \not\sqsubseteq Exp2$ et $Exp2 \not\sqsubseteq Exp1$, permet d'assurer OUBLIER(DIRE($C, Exp1$), $Exp2$) $\sqsubseteq Exp_1$. En effet,

Si $Exp1$ et $Exp2$ sont des restrictions existentielles, par la Définition OUBLIER *4.3.20*, on a : OUBLIER(DIRE(C, Exp_1), Exp_2) $\sqsubseteq Exp_1$ car DIRE(C, Exp_1) $\sqsubseteq Exp_1$ et $Exp1 \not\sqsubseteq Exp2$.

Si $Exp1$ est une restriction existentielle et $Exp2$ est une restriction universelle, alors par la Définition OUBLIER 4.3.20 les restrictions existentielles au top-level de DIRE(C, Exp_1) ne changent pas suivant l'exécution de

OUBLIER(DIRE(C, Exp_1), Exp_2). Donc, OUBLIER(DIRE(C, Exp_1), Exp_2) $\sqsubseteq Exp_1$ est déduite de DIRE(C, Exp_1) $\sqsubseteq Exp_1$.

Maintenant, on considère le cas où $Exp1$ est une restriction universelle et $Exp2$ est une restriction existentielle. On doit montrer que :

$lcs\{\{var(\text{DIRE}(C, Exp1))\ \}\cup$

$\{C'|C' \in ex(\text{OUBLIER}(\text{DIRE}(C, Exp1), Exp2))\ \}\} \sqsubseteq var(Exp1)$

La condition $Exp1 \not\geqslant Exp2$ implique que la description de concept $Exp1$ est de la forme : $\underbrace{\forall r...\forall r.}_{n} E'$ où $E' \in \{P, \exists r.D'\}$, $P \in N_C$ et D' est une $\mathcal{FLE}$-description de concept et $E' \not\sqsubseteq E''$, $E'' \not\sqsubseteq E'$ pour tout E'' où l'arbre de description $\mathcal{G}_{E''}$ est un sous-arbre au niveau n de l'arbre de description $\mathcal{G}_{Exp2}$. De plus, puisque DIRE(C, Exp_1) $\sqsubseteq Exp_1$ et $Exp1$ est une restriction universelle, alors par la Définition DIRE **??** (notamment par la propagation de la restriction universelle sur les restrictions existentielles), tous les sous-arbres $\mathcal{G}_F$ au niveau n de l'arbre de description de DIRE(C, Exp_1) doivent "contenir" le sous-arbre $\mathcal{G}_{E'}$(*i.e.* il existe un homomorphisme du $\mathcal{G}_{E'}$ dans chaque sous-arbre $\mathcal{G}_F$). Selon la Définition OUBLIER 4.3.20, tous les sous-arbres $\mathcal{G}_F$ soit sont conservés dans l'arbre de description de OUBLIER(DIRE(C, Exp_1), Exp_2), soit sont transformés en OUBLIER(F, E''). Puisque E'' et E' sont les restrictions existentielles, $E' \not\sqsubseteq E''$, $F \sqsubseteq E'$, alors OUBLIER(F, E'') $\sqsubseteq E'$. Par conséquent, on obtient $C' \sqsubseteq var(Exp1)$ pour tout $C' \in ex(\text{OUBLIER}(\text{DIRE}(C, Exp1), Exp2))$.

Le cas où $Exp1$ et $Exp2$ sont des restrictions universelles peut se traduire en les cas considérés.

(2) Supposons que $n = 0$ *i.e.* $Cons(r_2)$ apparaît au top-level de $Cons(r_1)$. Si $Cons(r_2) \not\sqsubseteq Exp1$, alors il existe un homomorphisme φ de $Exp1$ dans $Cons(r_1)$ tel que $\varphi(Exp1) \cap Cons(r_2) = \emptyset$ car $Cons(r_1) \sqsubseteq Exp1$ et $Exp1$ est une restriction existentielle ou universelle (description de concept simple). Donc, $Cons_{[r_2]}(r_1) \sqsubseteq Exp1$.

Supposons que $Cons(r_2) \sqsubseteq Exp1$. La partie (1) de cette proposition permet d'obtenir :

OUBLIER($Cons(r_2)$, $Exp2$) $\equiv$ OUBLIER(DIRE($Cons(r_2), Exp1$), $Exp2$) $\sqsubseteq Exp1$. Donc, $Cons_{[r_2]}(r_1) \sqsubseteq Exp1$.

Nous considérons le cas $n > 0$. Supposons que $Cons(r_2)$ apparaisse au niveau n de $Cons(r_1)$ *i.e.* il existe dans l'arbre de description $\mathcal{G}_{Cons(r_1)}$ un n-chemin $(c_1, ..., c_n)$, $c_i \in \{\forall r., \exists r.\}$, de la racine du $\mathcal{G}_{Cons(r_1)}$ vers la racine du $\mathcal{G}_{Cons(r_2)}$. Puisque $Cons(r_1) \sqsubseteq Exp1$, soit $\mathcal{G}_{E_1}$ un sous-arbre du $\mathcal{G}_{Exp1}$ au niveau n tel que $Cons(r_2) \sqsubseteq E_1'$ et $E_1' \not\geqslant Exp2$ où E_1' est un conjoint de E_1 (s'il n'existe pas de tel sous-arbre $\mathcal{G}_{E_1}$ du $\mathcal{G}_{Exp1}$, alors $Cons_{[r_2]}(r_1) \sqsubseteq Exp1$) . La partie (1) de cette proposition permet d'obtenir : $Cons_{[r_2]}(r_2) \sqsubseteq E_1'$, et donc $Cons_{[r_2]}(r_2) \sqsubseteq E_1$. Cela implique que $Cons_{[r_2]}(r_1) \sqsubseteq Exp1$.

□

Maintenant, nous formulons et montrons une condition de terminaison de l'inférence sur le formalisme hybride. La démonstration du théorème suivant est subordonnée à l'existence des propriétés des opérations de révision.

Théorème 4.5.7. *Soit* $\Delta := (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ *une base de connaissances où* $\Delta_{\mathcal{T}}$ *et* $\Delta_{\mathcal{R}}$ *sont acycliques. Pour chaque couple de règles* $r_i, r_j \in \Delta_{\mathcal{R}}$ *telles que* $Cons(r_i) = Cons(r_j)$*, supposons que :* $Ex(r_i) \not\geqslant Ex(r_j)$*. Alors, pour toute suite de transformations de* Δ *effectuée par les règles* $r_1, ..., r_i, ... \in \Delta_{\mathcal{R}}$ *:*

$$\Delta_1 \rightarrow \Delta_2 \rightarrow, ..., \rightarrow \Delta_i \rightarrow ...$$

, il existe n *fini tel que* $\Delta_n = \Delta_{n+1} = ...$

DÉMONSTRATION. Puisque $\Delta_{\mathcal{T}}$ et $\Delta_{\mathcal{R}}$ sont acycliques, il existe un ensemble de règles $R_1 \subseteq \Delta_{\mathcal{R}}$ tel que pour toute règle $r_1 \in R_1$, les concepts $Cons(r_1), Ex(r_1)$ ne sont pas dépendants des concepts à réviser. De la même manière, nous pouvons définir l'ensemble des règles $R_m \subseteq \Delta_{\mathcal{R}} \setminus (R_1 \cup ... \cup R_{m-1})$ tel que pour toute règle $r_m \in R_m$, les concepts $Cons(r_m), Ex(r_m)$ ne dépendent que des concepts $Cons(r_{m-1})$ où $r_{m-1} \in R_{m-1}$. On a : $\Delta_{\mathcal{R}}$=$(R_1 \cup ... \cup R_m)$.
Il est évident que la terminaison de modification des concepts $Cons(r_k)$ pour $k \in \{1, ..., m\}$ dépend seulement de la terminaison de modification des concepts $Cons(r_i)$ pour tout $i \in \{1, ..., k\}$. Afin de démontrer la proposition, nous n'avons besoin de démontrer que chaque $Cons(r_m)$ ne change plus après un nombre fini d'applications des règles où $r_m \in R_m$ car la démonstration pour R_k, $k < m$ ressemble à celle pour R_m. Donc, on doit démontrer que pour toute suite (peut-être infinie) de transformations effectuée par la suite des règles : $S = r^1, ..., r^n, ... \in R_m$ où $Cons(r^1)$=...=$Cons(r^m)$=... = E, alors E ne change plus après un nombre fini d'applications de ces règles.

(1) Si $r^i, r^j \in S$ telles que $r^i = r^j$, $j > i$ et $REV(r^i) = REV(r^j)$ = DIRE (*i.e.* $Ant(r^i) = Ant(r^j)$, $Cons(r^i) = Cons(r^j)$ et $Ex(r^i) = Ex(r^j)$), alors $S \simeq S \setminus \{r^j\}$ *i.e.* les deux bases de connaissances obtenues Δ par la suite S et par la suite $S \setminus \{r^j\}$ sont équivalentes. Effectivement, si l'on désigne par E_l le concept E obtenu après l'application d'une règle r^l pour tout $l \geq 1$, alors on doit démontrer que $E_{j-1} \sqsubseteq Ex(r^i)$ où $Cons(r^j)$=E_{j-1}. D'abord, on a : $E_i \sqsubseteq Ex(r^i)$ car E_i = DIRE(E_{i-1}, $Ex(r^i)$). De plus, pour tout k où $i < k < j$, on a $E_k \sqsubseteq Ex(r^i)$ si $E_{k-1} \sqsubseteq Ex(r^i)$ et $REV(r^k)$ = DIRE. D'autre part, puisque $Ex(r^i) \not\geqslant Ex(r^k)$, alors par la Proposition 4.5.6, on a : $E_k \sqsubseteq Ex(r^i)$ si $E_{k-1} \sqsubseteq Ex(r^i)$ et $REV(r^k)$ = OUBLIER pour tout k où $i < k < j$. Par conséquent, par la Définition DIRE **??**, on obtient : $E_{j-1} = E_j$ car $E_{j-1} \sqsubseteq Ex(r^i) = Ex(r^j)$.

(2) A partir de l'argument dans (1) , chaque r^i où $REV(r^i)$ = DIRE apparaît au plus une fois dans la suite S. Cela implique qu'il existe q tel que $REV(r^q) = ... = REV(r^{p+p})...$ = OUBLIER. On a : $E_q \not\sqsubseteq Ex(r^q)$. De plus, pour tout k où $k > q$, $E_k \not\sqsubseteq Ex(r^q)$ car $E_q \sqsubseteq ... \sqsubseteq E_k$ et $E_q \not\sqsubseteq Ex(r^q)$. Donc, si $r^k = r^q$ *i.e.* $Ex(r^k)$=$Ex(r^q)$ alors $E_{k-1} \not\sqsubseteq Ex(r^k)$ et $E_k = E_{k-1}$. Cela implique que chaque r^j où $REV(r^j)$ = OUBLIER, $j > q$, apparaît au plus une fois dans la suite S.

En bref, on obtient que chaque r^i où $REV(r^i)$ = DIRE apparaît au plus une fois dans la suite S et il existe $q \geq 1$ tel que chaque r^i où $REV(r^i)$ = OUBLIER apparaît au plus une fois dans la suite S où $i > q$. Par conséquent, on obtient *cqfd.*

□

Le Théorème 4.5.7 affirme que la transformation de la base de connaissances $\Delta := (\Delta_\mathcal{T}, \Delta_\mathcal{A}, \Delta_\mathcal{R})$ effectuée par le composant $\Delta_\mathcal{R}$ sous la restriction indiquée se termine après un nombre fini d'application des règles selon un ordre d'application. Cette restriction qui s'applique aux expressions de requête de révision (Exp) pour le même concept à réviser, semble raisonnable. La raison pour cela est que si le composant $\Delta_\mathcal{R}$ contient deux règles dont l'une ajoute une connaissance qui sera supprimée par l'autre, et le composant $\Delta_\mathcal{A}$ active ces règles, la B.C. peut impliquer une incompatibilité. En particulier, si les deux expressions de requête appartenant à deux règles DIRE et OUBLIER sont équivalentes, la restriction permet de limiter de telle sorte de lacune de conception.

Par ailleurs, nous nous rendons compte que la B.C. obtenue peut dépendre de l'ordre d'application de règles. Pour que la B.C. soit uniquement déterminée après l'application des règles, certains critères doivent être pris en compte. La première idée pour ces critères est de restreindre la partie des règles $\Delta_\mathcal{R}$ plutôt que la partie $\Delta_\mathcal{A}$. Cette restriction doit s'appuyer sur le nombre de règles appliquées ou sur la différence entre les composants $\Delta_\mathcal{T}$ générés avant et après l'application des règles. Tout d'abord, nous remarquons que l'ordre d'application des règles dans lequel le nombre des règles appliquées est minimal, ne coïncide pas avec l'ordre dans lequel le nombre des concepts modifiés est minimal. L'exemple suivant montre cette possibilité.

Exemple 4.5.8.

$\Delta_\mathcal{T}$:={
Équipe := ∀*membre*. **Personne** ⊓ (≥ 2 *membre*) ;
PetiteEquipe := **Équipe** ⊓ (≤ 4 *membre*) ;
GrandeEquipe := **Équipe** ⊓ (≥ 8 *membre*) ;
EquipeModerne := **Équipe** ⊓ ∃ *membre*. **Femme** ;
Epreuve := **ActivitéPhysique** ⊓ **ActivitéChorégraphique** ;
EquipeDeFoot := **Equipe** ⊓ ∀ *membre*. **Homme** ;
Année
} ;
$\Delta_\mathcal{A}$:={ **Année**(2004), **Epreuve**(danse), **EquipeDeFoot**(equipeDeFrance) ;
ActivitéPhysique(danse), **ActivitéChorégraphique**(danse) }
$\Delta_\mathcal{R}$:={
r1 : **Epreuve**(danse) ⇒ DIRE(**Equipe**, ∀*membre*. **Femme**) ;
r2 : **EquipeDeFoot**(equipeDeFrance) ⇒ OUBLIER(**PetiteEquipe**, ≤ 5 *membre*) ;
r3 : **EquipeDeFoot**(equipeDeFrance) ⇒ OUBLIER(**GrandeEquipe**, ≥ 10 *membre*) ;
r4 : **Année**(2004) ⇒ OUBLIER(**Epreuve**, **ActivitéChorégraphique**) ;
r5 : **Année**(2004) ⇒ DIRE(**Epreuve**, **ActivitéIntellectuelle**)

}

A partir du $\Delta_{\mathcal{A}}$, on a toutes les règles qui sont activées. Pourtant, si l'on choisit l'ordre (r1,r4,r5), alors les 3 opérations suivantes sont effectuées :
DIRE(**Équipe**, $\forall$*membre*. **Femme**),
DIRE(**Epreuve**, **ActivitéIntellectuelle**),
OUBLIER(**Epreuve**, **ActivitéChorégraphique**) et
les 5 concepts suivants sont modifiés : **Epreuve**, **Équipe**, **PetiteEquipe**, **GrandeEquipe**, **EquipeModerne**. En effet, après l'application de la règle r1, l'assertion **EquipeDeFoot**(equipeDefrance) n'est plus valable, donc les règles r2,r3 ne sont plus activées.
Par contre, si l'on choisit l'ordre (r4,r5,r2,r3), alors les 4 opérations suivantes sont effectuées :
OUBLIER(**Epreuve**, **ActivitéChorégraphique**),
DIRE(**Epreuve**, **ActivitéIntellectuelle**),
OUBLIER(**PetiteEquipe**, ≤ 5 *membre*),
OUBLIER(**GrandeEquipe**, ≥ 10 *membre*) et
les 3 concepts suivants sont modifiés : **Epreuve** , **PetiteEquipe**, **GrandeEquipe**.

Si le composant $\Delta_{\mathcal{R}}$ assure les modifications nécessaires qui permettent de partager la compréhension entre les acteurs, une restriction sur le nombre d'application des règles semble plus raisonnable que sur le nombre des concepts modifiés. Par la suite, nous considérons deux politiques opposées par le nombre des règles appliquées.

Définition 4.5.9. Soit $\Delta := (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ une base de connaissances où $\Delta_{\mathcal{T}}$ est acyclique et $\Delta_{\mathcal{R}}$ est non-récursif. Un sous-ensemble des règles $R \subseteq \Delta_{\mathcal{R}}$ est appelé *complétion de règles* ou *complétion* de Δ si Δ ne change plus après l'application des règles dans R selon un ordre.
Une complétion R est appelée *maximal* (*minimal*) s'il y a un ordre d'application des règles dans R , appelé *ordre maximal* (*ordre minimal*) tel que :

(1) Toute règle déclenchable (applicable) appartient à R après cette application des règles.

(2) Le nombre des règles de R est maximal (minimal).

4.5.1. Nombre minimal des règles appliquées. Cette politique reflète l'idée de conservation de la B.C selon laquelle la B.C est modifiée au minimum par l'application des règles.
D'abord nous remarquons qu'il existe plusieurs ensembles minimaux des règles qui génèrent deux B.C différentes après les applications de deux ensembles de règles. On considère l'exemple suivant :

Exemple. $\Delta_{\mathcal{A}}$:=$\{D_1(d), D_2(d), C_1(d)\}$;
$\Delta_{\mathcal{R}} := \{$
$r_1 : C_1(d) \Rightarrow \text{DIRE}(D_1, Exp1)$,
$r_2 : D_1(d) \Rightarrow \text{DIRE}(D_2, Exp2)$,

$r_3 : D_2(d) \Rightarrow \text{DIRE}(X, \mathit{Exp}3)$ }
Il est évident que $\Delta_{\mathcal{R}}$ est non-récursif. Sans perte de généralité, nous supposons que l'application de la règle r_1 désactive r_2 et l'application de la règle r_2 désactive r_3. Donc, nous avons deux ensembles minimaux des règles (r_1,r_3) et (r_2,r_1).

Cela montre que les conditions citées sur le composant des règles $\Delta_{\mathcal{R}}$ ne sont pas suffisantes. Nous avons besoin de restreindre encore plus le composant $\Delta_{\mathcal{R}}$ pour obtenir uniquement la B.C. après un nombre minimal des règles appliquées. Cependant, ce problème reste encore ouvert.

4.5.2. Nombre maximal des règles appliquées. Cette politique reflète l'idée d'exploitation des connaissances du composant $\Delta_{\mathcal{R}}$ selon laquelle une règle doit être appliquée si c'est possible. Cette idée résulte de la maximisation de la compréhension partagée par plusieurs acteurs à travers l'application des règles.

Définition 4.5.10. Soit $\Delta := (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ une base de connaissances où $\Delta_{\mathcal{T}}$ est acyclique et $\Delta_{\mathcal{R}}$ est non-récursif. Un *graphe de dépendance* $\mathcal{G}_R$ peut être construit du composant $\Delta_{\mathcal{R}}$ comme suit :
- Chaque nœud de $\mathcal{G}_R$ correspond à une règle $r \in \Delta_{\mathcal{R}}$.
- Un arc de $\mathcal{G}_R$ relie un nœud n' à un autre nœud n'' de $\mathcal{G}_R$ où n', n'' correspondent à deux règles r', r'' si $\mathit{Ant}(r'')$ est défini *directement* ou *indirectement* via $\mathit{Cons}(r')$ *i.e.* $\mathit{Ant}(r'')$ dépend de $\mathit{Cons}(r')$.

Les nœuds sans arc d'entrée sont appelés *nœuds initiaux*. Les nœuds sans arc de sortie sont appelés *nœuds terminaux*.

La Figure 4.5.1 représente le graphe de dépendance des règles dans l'Exemple 4.5.8.

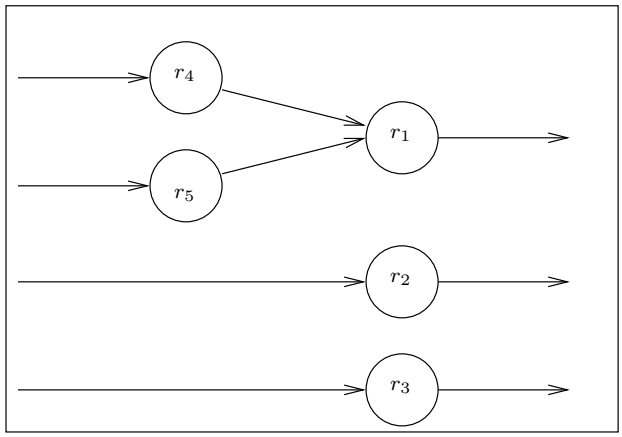

FIG. 4.5.1. Graphe de Dépendance

Remarque 4.5.11. Le graphe de dépendance a les propriétés suivantes :

- $\mathcal{G}_R$ peut être modifié par les opérations OUBLIER et DIRE *i.e.* certains arcs (r', r'') où $Ant(r'')$ dépend *indirectement* de $Cons(r')$, peuvent être supprimés (cassés) suivant l'application d'une règle OUBLIER ou DIRE. Par contre, si $Ant(r'')$ dépend *directement* de $Cons(r')$ *i.e.* $Cons(r') \in Ant(r'')$, alors l'arc (r', r'') est préservé.
- L'application d'une règle r peut casser un arc (r'', r') s'il existe un arc (r, r') et $Cons(r)$ dépend de $Cons(r'')$. En effet, lorsque l'application d'une règle r est effectuée, la définition du concept $Cont(r)$ est modifiée *i.e.* certains noms de concept dans la définition $Cont(r)$ peuvent être substitués par d'autres noms. Cela implique que certaines dépendances entre le concept $Cont(r)$ et d'autres concepts C' n'existent plus. Donc, s'il existe un arc représentant la dépendance entre $Cont(r)$ et $Ant(r')$, alors il existe également les arcs représentant les dépendances *indirectes* entre C' et $Ant(r')$. Les arcs qui représentent ces dépendances indirectes peuvent être cassés par l'application de la règle r. Par conséquent, s'il y a un arc qui représente une dépendance indirecte entre $Cons(r'')$ et $Ant(r')$ où $Cont(r)$ dépend de $Cons(r'')$, alors l'arc (r'', r') peut être cassé par l'application de la règle r.
- $\mathcal{G}_R$ est acyclique si $\Delta_{\mathcal{R}}$ est non-récursif.

Grâce au graphe de dépendance correspondant au composant $\Delta_{\mathcal{R}}$, nous pouvons proposer un algorithme qui permet de déterminer un ordre d'application des règles dans lequel le nombre des règles appliquées est maximal. De plus, cet ensemble maximal est unique.

Algorithme 4.5.12. *(Ordre maximal)*
Entrée *:*
- *Base de connaissances $\Delta := (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ où le composant Δ_R satisfait à la condition de terminaison présentée dans la Théorème 4.5.7.*

Sortie *: Ordre maximal O*

(1) *$O := \emptyset$;*

(2) *Construire le graphe de dépendance $\mathcal{G}_R$;*

(3) *Dans $\mathcal{G}_R$, pour chaque chemin d'un nœud terminal vers un nœud initial, on peut déterminer le premier nœud rencontré* n *qui correspond à une règle activée et* n $\notin O$*. Désigner par C l'ensemble des nœuds déterminés ;*

(4) *Enlever les nœuds* n' *de C correspondant à une règle DIRE s'il existe un chemin d'un nœud terminal vers le nœud* n' *contenant un nœud* n" *qui appartient à C. Désigner par D l'ensemble des nœuds restants de C ;*

(5) *Dans l'ensemble D, affecter à chaque nœud une priorité selon les principes suivants :*

   (a) *Un nœud correspondant à une règle OUBLIER est plus prioritaire qu'un nœud correspondant à une règle DIRE.*

(b) *Les nœuds correspondant à des règles OUBLIER ont la même priorité et les nœuds correspondant à des règles DIRE ont la même priorité. On obtient deux groupes disjoints de nœuds tels que tous les nœuds dans le même groupe ont la même priorité et la priorité du groupe des nœuds correspondant aux règles OUBLIER est supérieure à celle du groupe des nœuds correspondant aux règles DIRE.*

(c) *Dans chaque groupe des nœuds, s'il existe deux arcs* (r_1, r), (r_2, r) *et trois nœuds correspondant aux règles* r, r_1, r_2 *tels que* $Cons(r_1)$ *dépend de* $Cons(r_2)$*, alors* r_2 *et les prédécesseurs de* r_2 *sont plus prioritaires que* r_1 *;*

(6) $O := O \cup \{n\}$ *où* n $\in D$ *et* n *est le plus prioritaire (non-déterministe) ;*

(7) *Saturer* O *i.e. les règles dans* O *sont appliquées jusqu'à ce que* $\Delta_{\mathcal{T}}$ *ne change plus. Répéter 2. ;*

L'algorithme s'arrête lorsque $\mathcal{G}_R$ *n'a plus de nœud* n $\notin O$ *qui corresponde à une règle activée.*

Exemple 4.5.13. Le graphe de dépendance correspondant aux règles r_1, r_2, r_3 dans l'exemple 4.5.1 comporte les arcs suivants : (r_1, r_2), (r_2, r_3). L'application de l'Algorithme 4.5.12 à ce graphe donne l'ordre maximal comme suit : (r_3, r_2, r_1).

Dans la suite, s'il n'y a pas de confusion, on peut se référer à une règle au lieu du nœud correspondant dans un graphe de dépendance. La proposition suivante assure la correction de l'Algorithme 4.5.12 et l'existence unique de l'ensemble maximal de règles.

Proposition 4.5.14. *Soit* $\Delta := (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$ *une base de connaissances où le composant* Δ_R *satisfait à la condition de terminaison présentée dans le Théorème 4.5.7. L'ensemble des règles* O *retourné par l'Algorithme 4.5.12 est une complétion et cette complétion contient le nombre maximal des règles appliquées. De plus, s'il existe une complétion* O' *tel que* $|O| = |O'|$*, alors* $O = O'$*.*

Pour préparer la démonstration de la proposition, nous formulons et montrons le lemme suivant.

Lemme 4.5.15. *Soit* O *l'ensemble des règles retourné par l'Algorithme 4.5.12. Si une règle* $r \in \Delta_R$ *est déclenchable alors* $r \in O$*.*

Démonstration. Nous distinguons les deux cas suivants :

(1) La règle r est initialement activée. En effet, supposons que $r \notin O$. Alors à une itération de l'Algorithme 4.5.12, il existe un arc (r', r) dans le graphe $\mathcal{G}_R$ où r' est une règle DIRE qui désactive r. Donc, r' est un prédécesseur de r et r' est choisie dans O lorsque r est activée et $r \notin O$. Cela contredit la construction de l'étape 4. de l'Algorithme 4.5.12.

(2) La règle r est activée par un ensemble R de règles OUBLIER déclenchables dans l'ordre $R = \{r_1, ..., r_n\}$ où $R \subseteq \Delta_R$. Cela implique que les règles $r_1, ..., r_n$ sont les prédécesseurs de la règle r dans le graphe de dépendance. Supposons que $\{r_{I(1)}, ..., r_{I(n)}\} \subseteq O$ où I qui est une permutation de $\{1, ..., n\}$ est l'ordre des règles dans O.

(a) Il n'existe pas une règle DIRE $r_d \notin R$ qui est un prédécesseur de r et est choisie avant une règle $r_i \in \{r_{I(1)}, ..., r_{I(n)}\}$. Nous le montrons par l'absurde.
En effet, si la règle DIRE r_d est choisie à une itération de l'Algorithme 4.5.12, alors l'ensemble D qui est construit par l'étape 4. à cette itération ne contient que les règles DIRE car selon le critère de priorité 5.(a) de l'Algorithme 4.5.12, s'il existe une règle OUBLIER r'' dans D, alors cette règle r'' sera choisie. Cela implique qu'à cette itération soit *i)* il n'y a plus une règle OUBLIER activée dans le graphe de dépendance *i.e.* la règle OUBLIER r_i n'est jamais choisie, ce qui contredit $r_i \in \{r_{I(1)}, ..., r_{I(n)}\}$, soit *ii)* il existe une règle OUBLIER activée r_x. Alors, dans ce cas, tout chemin d'un nœud terminal vers la règle OUBLIER r_x doit passer par une règle DIRE activée qui n'est pas encore choisie car, sinon par le critère de priorité (a) de l'étape 5. de l'Algorithme 4.5.12, la règle OUBLIER r_x sera choisie au lieu de la règle DIRE r_d.
On a : $r_x \notin \{r_{I(1)}, ..., r_{I(n)}\}$ car si $r_x \in \{r_{I(1)}, ..., r_{I(n)}\}$, alors tout chemin d'un nœud terminal vers la règle r contient une règle activée qui n'est pas encore choisie (tout chemin d'un nœud terminal vers la règle r peut être allongé pour atteindre r_x qui est un prédécesseur de r) *i.e.* il existe un chemin d'un nœud terminal vers la règle DIRE r_d contient une règle activée qui n'est pas encore choisie. Alors, la règle DIRE r_d ne peut pas être choisie d'après l'étape 4. de l'Algorithme 4.5.12. C'est une contradiction.
Donc, tout chemin de la règle OUBLIER r_x vers une règle $r_j \in \{r_{I(1)}, ..., r_{I(n)}\}$ (le chemin est dans le sens inverse) doit passer par une règle DIRE (activée). Cela ne permet pas à la règle r_x d'activer les règles $\{r_{I(1)}, ..., r_{I(n)}\}$. Cela signifie que la règle $r_i \in \{r_{I(1)}, ..., r_{I(n)}\}$ n'est jamais activée par n'importe quelle règle OUBLIER r_x. C'est une contradiction car il y a certaines règles $r' \in R'$ qui ne seront jamais activées.

(b) S'il existe une nouvelle règle OUBLIER $r_o \notin R$ qui est un prédécesseur de r et est choisie avant une règle $r_i \in \{r_{I(1)}, ..., r_{I(n)}\}$, alors par le critère de priorité (c) de l'étape 5. de l'Algorithme 4.5.12, $Cont(r_o)$ ne dépend d'aucune règle $r_i \in \{r_{I(1)}, ..., r_{I(n)}\}$, soit *ii)* $Cont(r_o)$ est indépendante de toute $Cont(r_i)$ où $r_i \in \{r_{I(1)}, ..., r_{I(n)}\}$. Donc, par la Remarque 4.5.11, l'application de la règle OUBLIER r_o ne casse aucun arc entre r' et r pour toute règle $r' \in \{r_{I(1)}, ..., r_{I(n)}\}$ qui n'est pas encore choisie. De plus, on a : $Ant(r) \sqsubseteq Ant_{r_0}(r)$ où $Ant_{r_0}(r)$ est le concept $Ant(r)$ obtenu juste après l'application de la règle r_0. Par conséquent, si l'application de toutes les règles $r' \in \{r_{I(1)}, ..., r_{I(n)}\}$ active la règle

r, alors l'application de toutes les règles $\{r_{I(1)}, ..., r_{I(n)}\} \cup \{r_0\}$ active également la règle r.

(c) Si $Cons(r_{I(i)})$ et $Cons(r_{I(j)})$ sont indépendantes où $1 \leq I(i) < I(j) \leq n$, alors $A_{r_{I(i)},r_{I(j)}} \equiv A_{r_{I(j)},r_{I(i)}}$ où $A_{u,v}$ est obtenue de l'application des règles u, v au concept $A \in Ant(r)$.

(d) Si $Cons(r_{I(j)})$ dépend de $Cons(r_{I(i)})$ où $1 \leq I(i) \neq I(j) \leq n$, alors $I(i) < I(j)$. En effet, i) si $r_{I(i)}$ est activée et $r_{I(j)}$ n'est pas encore activée, alors il est évident que $I(i) < I(j)$. ii) Si $r_{I(j)}$ et $r_{I(i)}$ sont toutes activées auparavant, alors selon le critère de priorité (c) de l'étape 5. de l'Algorithme 4.5.12, la règle $r_{I(i)}$ est choisie avant $r_{I(j)}$ *i.e.* $I(i) < I(j)$. iii) Si $r_{I(j)}$ est activée et $r_{I(i)}$ n'est pas encore activée, alors selon le critère de priorité (c) de l'étape 5. de l'Algorithme 4.5.12, les prédécesseurs de $r_{I(i)}$ sont plus prioritaires que $r_{I(j)}$ dans le graphe de dépendance. Puisque $r_{I(i)}$ est déclenchable, alors $r_{I(i)}$ devient activée après que toutes les prédécesseurs déclenchables de $r_{I(i)}$ sont choisies. Cela nous conduit au cas ii).

Nous montrons que $Cons_{[r_{I(j)},r_{I(i)}]}(r_{I(j)}) \sqsubseteq Cons_{[r_{I(i)},r_{I(j)}]}(r_{I(j)})$ où $Cons_{[u,v]}(r)$ est désigné pour la description de concept $Cons(r)$ obtenue de l'ordre d'application des règles $[u, v]$. En effet, puisque $Cons(r_{I(i)})$ ne dépend pas de $Cons(r_{I(j)})$ (car $Cons(r_{I(j)})$ dépend de $Cons(r_{I(i)})$ et $\Delta_{\mathcal{T}}$ est acyclique) alors les $Cons(r_{I(i)})$ obtenues des deux ordres d'application : $[r_{I(i)}, r_{I(j)}]$ et $[r_{I(j)}, r_{I(i)}]$ sont équivalentes.
Si $Cons(r_{I(j)}) \not\sqsubseteq Ex(r_{I(j)})$ et donc, $Cons_{[r_{I(i)}]}(r_{I(j)}) \not\sqsubseteq Ex(r_{I(j)})$ (car $Cons(r_{I(j)}) \sqsubseteq Cons_{[r_{I(i)}]}(r_{I(j)})$), alors par la Définition OUBLIER 4.3.20, on obtient $Cons_{[r_{I(j)}]}(r_{I(j)}) \sqsubseteq Cons_{[r_{I(i)},r_{I(j)}]}(r_{I(j)})$.
Supposons que $Cons(r_{I(j)}) \sqsubseteq Ex(r_{I(j)})$. Puisque $Ex(r_{I(i)}) \not\gtrless Ex(r_{I(j)})$, alors par la partie (2) de la Proposition 4.5.6, on obtient : $Cons_{[r_{I(i)}]}(r_{I(j)}) \sqsubseteq Ex(r_{I(j)})$. Donc, selon la partie (3) de la Remarque 4.3.21, on obtient également : $Cons_{[r_{I(j)}]}(r_{I(j)}) \sqsubseteq Cons_{[r_{I(i)},r_{I(j)}]}(r_{I(j)})$ car $Cons(r_{I(j)}) \sqsubseteq Cons_{[r_{I(i)}]}(r_{I(j)})$.
De plus, si $Cons_{[r_{I(j)}]}(r_{I(j)})$ dépend de $Cons(r_{I(i)})$ ou non, alors $Cons_{[r_{I(i)},r_{I(j)}]}(r_{I(j)})$ dépend de $Cons(r_{I(i)})$ ou non respectivement (la relation de subsomption est préservée lors de la substitution de $Cons(r_{I(i)})$ par $Cons_{[r_{I(i)}]}(r_{I(i)})$). Donc, dans tous les cas, on a :

$$Cons_{[r_{I(j)},r_{I(i)}]}(r_{I(j)}) \sqsubseteq Cons_{[r_{I(i)},r_{I(j)},r_{I(i)}]}(r_{I(j)})$$

D'autre part, puisque $Cons_{[r_{I(i)}]}(r_{I(i)}) \not\sqsubseteq Ex(r_{I(i)})$,
$REV(r_{I(i)})$=$REV(r_{I(j)})$ = OUBLIER et donc,
$Cons_{[r_{I(i)}]}(r_{I(i)}) = Cons_{[r_{I(i)},r_{I(i)}]}(r_{I(i)})$, alors
$Cons_{[r_{I(i)},r_{I(j)},r_{I(i)}]}(r_{I(j)}) = Cons_{[r_{I(i)},r_{I(j)}]}(r_{I(j)})$. On obtient :
$Cons_{[r_{I(j)},r_{I(i)}]}(r_{I(j)}) \sqsubseteq Cons_{[r_{I(i)},r_{I(j)}]}(r_{I(j)})$.

Par ailleurs, l'ordre des règles $[r_1, ..., r_n]$ peut être transformé en l'ordre $[r_{I(1)}, ..., r_{I(n)}]$ par plusieurs échanges $[r_u, r_v]$. Cela implique qu'un concept $A \in Ant(r)$ obtenu de l'ordre d'application des règles $[r_1, ..., r_n]$ est subsumé par ce concept obtenu de l'ordre d'application des règles $[r_{I(1)}, ..., r_{I(n)}]$ *i.e.* si la règle r est activée par l'ordre d'application des règles $[r_1, ..., r_n]$, alors la règle r est également activée par l'ordre d'application des règles $[r_{I(1)}, ..., r_{I(n)}]$.

Maintenant, nous montrons que $\{r_1, ..., r_n\} \subseteq O$. Pour chaque r_i où $i \in \{1, ..., n\}$, si r_i est initialement activée alors $r_i \in O$ selon la démonstration ci-dessus (1.). Sinon, puisque r_i est déclenchable, il existe un ensemble R_i de règles OUBLIER déclenchables qui activent r_i. Cet argument est appliqué jusqu'à ce que R_i ne contient que les règles initialement activées. A partir de 1. et de 2., on peut conclure que $r_i \in O$ pour tout $i \in \{1, ..., n\}$.

En bref, nous avons prouvé que :

(1) Si une règle r est initialement activée alors $r \in O$.

(2) Si une règle r est activée par un ensemble R de règles OUBLIER déclenchables dans l'ordre $R = \{r_1, ..., r_n\}$ où $R \subseteq \Delta_R$ alors,

   (a) $R \subseteq O$

   (b) Soit I l'ordre d'application des règles $\{r_1, ..., r_n\}$ dans O, et donc $\{r_{I(1)}, ..., r_{I(n)}\} \subseteq O$. Il n'existe pas une règle $r' \in O$ où $REV(r')$=DIRE est choisie avant une règle $r_{I(k)}$ et $1 \leq k \leq n$.

   (c) S'il existe pas une règle $r' \in O$ où $REV(r')$=OUBLIER est choisie avant une règle $r_{I(k)}$ et $1 \leq k \leq n$, alors cela ne change pas l'activation de la règle r par l'ensemble R des règles.

   (d) L'application de l'ensemble des règles $\{r_{I(1)}, ..., r_{I(n)}\}$ dans l'ordre I active la règle r.

Tout cela nous permet d'obtenir *cqfd.*

□

La démonstration de la Proposition 4.5.14 est directement établie à l'aide du Théorème 4.5.7 et du Lemme 4.5.15.

DÉMONSTRATION. D'abord nous montrons que l'Algorithme 4.5.12 se termine. En effet, d'une part l'ensemble D qui est déterminé par l'étape 4. de l'Algorithme est non vide à chaque itération car selon la Définition 4.5.10 (le graphe $\mathcal{G}_R$ est acyclique), pour chaque nœud n du graphe $\mathcal{G}_R$, il existe toujours un chemin d'un nœud terminal vers n. De plus, $\Delta_{\mathcal{R}}$ est fini et à chaque itération une règle est choisie de l'ensemble D, donc l'algorithme se termine si l'étape 7. de l'Algorithme 4.5.12 pour la saturation de O se termine. Et cela est également assuré par le Théorème 4.5.7. On obtient que O est une complétion.

(1) La maximalité de O est déduite directement du Lemme 4.5.15.

(2) L'unicité de O. Supposons qu'il existe une complétion O' où $|O'| = |O|$ et $O' \setminus O \neq \emptyset$. Cela contredit $O' \subseteq O$ que l'on a démontré.

(3) L'unicité de $\Delta_{\mathcal{T}}$ (TBox) c'est à dire que le TBox $\Delta_{\mathcal{T}}$ obtenu ne dépend pas du choix non-déterministe qui est fait dans l'étape 6. de l'Algorithme 4.5.12. En effet, si deux règles r_i, r_j ont la même priorité et, $Cons(r_i)$ et $Cons(r_j)$ sont indépendantes ($REV(r_i) = REV(r_i)$ et r_i, r_j sont activées), l'affirmation est évidente. Si $Cons(r_j)$ dépend de$Cons(r_i)$, alors selon l'étape 5.(c) r_i est choisie avant r_j.

□

Exemple 4.5.16. (Exemple d'Application).

Maintenant, nous présentons une extension de l'exemple d'introduction qui montre l'utilisation de règles de révision et la nécessité de l'inférence sur le formalisme hybride.
Supposons que l'on ait l'ontologie partagée par les acteurs :

$\Delta_{\mathcal{T}} := \{$
Produit, **Polluant**, **Chaleur**, **Carbonisation**, **Bruit**, **Toxicité**, **EntrepriseDeConst**,
EntrepriseEurop := $\exists$*associer*.**Européen**,
EntrepriseAmér := $\exists$*associer*.**Américain**,
Feu := **Chaleur** $\sqcap$ **Carbonisation**
$\}$

Les concepts **Produit**, **Polluant**, **Bruit**, **Chaleur, Carbonisation**, **Toxicité**, **Européen**, **Américain**, **EntrepriseDeConst** sont primitifs. Les rôles *associer*, *résister, acheté-par*, *vendu-par* sont primitifs. Le concept **Feu** possède deux propriétés : la **Chaleur** et la **Carbonisation**. Le concept **EntrepriseEurop** (respectivement, **EntrepriseAmér**) est défini comme un ensemble d'individus qui possède un *associé* **Européen** (respectivement, **Américain**). Le concept **ProdRésAuPolluant** est défini comme un **Produit** capable de *résister* au **Polluant**.
Il y a deux acteurs qui possèdent les deux ontologies dérivées suivantes :

$\Delta_{\mathcal{T}_1} := \{$
ProdRésAuPolluant := **Produit** $\sqcap$ $\exists$*résister*.**Bruit** ,
ProdDImport := **ProdRésAuPolluant** $\sqcap\exists$*contrôler*.**Toxicité** $\sqcap$ $\exists$*résister*.**Chaleur**

$\}$
et
$\Delta_{\mathcal{T}_2}$ (construction) := $\{$
ProdRésAuPolluant := **Produit** $\sqcap$ $\exists$*résister*.**Feu** $\sqcap$ $\exists$*résister*.**Bruit** ,
ProdDImport := **ProdRésAuPolluant** $\sqcap\exists$*contrôler*.**Toxicité**
$\}$

Le rôle *contrôler* qui est primitif, permet de déterminer si un produit est contrôlé.

Il y a deux versions différentes du concept défini **ProdRésAuPolluant**. Pour un acteur dans le secteur de la construction, ce concept est défini comme un concept capable de résister à la fois au **Feu** et au **Bruit**. Par contre, pour un acteur dans les autres secteurs, le concept **ProdRésAuPolluant** est défini comme un concept qui ne peut que résister au **Bruit**.

De même, le concept défini **ProdDImport** (produit d'importation) est différemment défini dans les deux ontologies. Pour un acteur dans le secteur de la construction, ce concept est capable de résister à la **Chaleur** car il est subsumé par le concept **ProdRésAuPolluant** qui peut résister au **Feu**. Cependant, cette propriété est exprimée dans la définition du concept dans l'ontologie des autres acteurs.
Supposons que les deux acteurs utilisent le même ensemble des fait $\Delta_{\mathcal{A}}$ comme suit :

$\Delta_{\mathcal{A}}$:={*vendu-par*(*a*,*b*), *acheté-par*(*a*,*c*), **ProdRésPolluant***(a),* **EntrepriseEurop**(*b*), **EntrepriseAmér**(*c*), **Polluant**(*Bruit*) }.

Notons que le concept **Bruit** a uniquement un individu.
Le fait *vendu-par*(*a*,*b*) signifie que le produit *a* est vendu par l'entreprise *b*. Le fait *acheté-par*(*a*,*c*) signifie que le produit *a* est acheté par l'entreprise *c*.
Soit $\Delta_{\mathcal{R}}$ comporte les deux règles de révision suivantes :

$\Delta_{\mathcal{R}}$:= {
r_1 : *vendu-par*(X, Y_1) $\wedge$ *acheté-par*(X, Z_1) $\wedge$
associer(Y_1, Y_2) $\wedge$ *associer*(Z_1, Z_2) $\wedge$
Européen(Y_2) $\wedge$ **Américain**(Z_2)
$\Rightarrow$ DIRE(**ProdRésPolluant**, $\exists$ *résister*.**Feu**) ;
r_2 : **EntrepriseDeConst**(Y_1) $\wedge$ *vendu-par*(X, Y_1) $\wedge$
ProdRésPolluant(X)
$\Rightarrow$ OUBLIER(**ProdDImport**, $\exists$ *résister*.**Chaleur**)
}

La règle r_1 dit que : si un produit X est vendu par une entreprise Y_1 qui a un associé **Européen**, le produit X est acheté par une entreprise Z_1 qui a un associé **Américain**, alors le concept **ProdRésPolluant** doit être interprété comme un produit capable de résister au ***Feu***. Cette règle permet de traiter le *contexte géographique i.e.* deux acteurs qui viennent des deux continents différents, et donc qui ont deux normes différentes, peuvent partager la définition du concept **ProdRésPolluant**.

Les faits **EntrepriseEurop**(b) et **EntrepriseEurop** := $\exists$*associer*.**Européen** impliquent *associer*(b, b') et **Européen**(b') .
Les faits **EntrepriseAmér**(c) et **EntrepriseAmér** := $\exists$*associer*.**Américain** impliquent *associer*(c, c') et **Américain**(c') . Donc, la condition de la règle r_1 est vérifiée.

La règle r_2 dit que : si une **EntrepriseDeConst** (entreprise de la construction) Z_1 vend un produit X qui est un produit **ProdRésPolluant** , alors le concept **ProdDImport** n'a pas besoin de la propriété de résistance à la **Chaleur** qui est explicitement défini. Cette règle traite le *contexte industriel i.e.* deux acteurs appartiennent

à deux industries différentes. La satisfaction de la condition de la règle r_2 est déduite directement du $\Delta_{\mathcal{A}}$.

Maintenant, supposons que l'acteur qui utilise l'ontologie $\Delta_{\mathcal{T}_1}$ reçoit deux concepts **ProdDImport** et **ProdRésPolluant**. Cet acteur doit appliquer les règles r_1, r_2 afin de partager la compréhension de ces deux concepts. Bien que les deux règles r_1 et r_2 soient initialement activées, nous pouvons obtenir deux ontologies différentes suivant l'application des règles. Cela dépend de l'ordre d'application de ces règles.

En effet, si l'ordre (r_2,r_1) est choisi, toutes les deux règles sont exécutées et on obtient les nouvelles définitions des deux concepts **ProdDImport** et **ProdRésPolluant** comme suit :

Selon l'Algorithme 4.3.17, on obtient :

OUBLIER (**ProdRésAuPolluant** ⊓∃*contrôler*.**Toxicité** ⊓ ∃*résister*.**Chaleur**, ∃ *résister*.**Chaleur**)
= **ProdRésAuPolluant** ⊓∃*contrôler*.**Toxicité**

Selon l'Algorithme 4.3.29, on obtient :

DIRE (**Produit** ⊓ ∃*résister*.**Bruit** , ∃ *résister*.**Feu**)
= **Produit** ⊓ ∃*résister*.**Bruit** ⊓∃ *résister*.**Feu**

Si l'ordre (r_1,r_2) est choisi, la seule règle r_1 est exécutée, car suivant l'application de la règle r_1, la révision sur $\Delta_{\mathcal{A}}$ élimine l'assertion **ProdRésPolluant***(a)*. Cela rend la règle r_2 inactivée. Donc, on obtient la nouvelle définition du concept **ProdRésPolluant** comme suit :

Selon l'Algorithme 4.3.29, on obtient :

DIRE (**Produit** ⊓ ∃*résister*.**Bruit** ,∃ *résister*.**Feu**)
= **Produit** ⊓ ∃*résister*.**Bruit** ⊓∃ *résister*.**Feu**

A partir de cet exemple, le point de vue dans lequel on choisit l'*ordre maximal* (Section 4.5.2) est renforcé car la différence entre les ontologies obtenues par l'ordre d'application des règles (r_2,r_1) est moins importante que celle entre les ontologies obtenues par l'ordre d'application des règles (r_1, r_2).

4.6. Conclusion

Dans ce chapitre, nous avons introduit la théorie de la révision de croyances : canevas AGM. A partir de cela, l'ensemble de principes pour la révision d'une base de connaissances générale est construit. Nous avons également montré comment projeter ces principes à la terminologie. Par la suite, certaines approches de révision et un canevas pour les opérations de révision sur la terminologie sont présentés. Dans l'approche conservatrice, la préservation des assertions sur l'ABox, malgré les

modifications de définition de concept sur le TBox, attire notre attention sur des inférences avec l'opérateur épistémique dans un monde fermé. Toutefois, l'interaction entre l'opérateur épistémique et les opérateurs de révision n'a pas été étudiée dans ce mémoire. Ensuite, nous avons montré comment définir les opérations de révision pour le langage $\mathcal{FLE}$ dans l'approche structurelle. Cette dernière permet aux opérations de révision définies au niveau de la syntaxe de satisfaire les principes de révision au niveau de la sémantique. Une étude profonde sur la complexité de ces opérateurs est nécessaire pour que l'on puisse améliorer la performance des algorithmes.
Le chapitre se termine par un service d'inférence pour le formalisme hybride avec la présence du composant de règles de révision. Ce service nous permet d'implémenter un système dans lequel plusieurs acteurs correspondant à plusieurs ontologies dérivées effectuent les échanges en partageant les connaissances formalisées dans chaque ontologie dérivée. Par la suite, un algorithme, qui est conçu, permet de calculer un sous-ensemble des règles maximal dont l'application assure la terminaison et le partage des connaissances le plus possible entre les ontologies en question.
Une question qui se pose concerne l'extension des opérations de révision à des langages avec la restriction de cardinalité et la disjonction. Cette extension exige une caractérisation de subsomption structurelle pour ces langages. D'autre part, si nous enrichissons le formalisme hybride actuel en ajoutant les règles de Horn au composant de la règle, alors quelles interactions se produiront dans le nouveau formalisme ? Dans ce cas, le formalisme obtenu sera une extension du langage CARIN. Donc, une extension des services d'inférences de CARIN au formalisme hybride mérite d'être étudiée.

Troisième partie

Mise en œuvre

CHAPITRE 5

Échange de Données dans le Commerce Électronique et Problème de Transparence Sémantique

Dans ce chapitre, nous présentons d'abord le commerce électronique (C.E) et décrivons brièvement certains points intéressants de ce domaine. Ensuite, les modèles d'échanges de données importants utilisés dans le C.E sont étudiés dans le but d'identifier la problématique de chacun. Ces études montrent qu'un des obstacles les plus importants à l'interopérabilité de communication entre les acteurs est le problème de transparence sémantique. En particulier, nous investiguons les parties concernant la représentation de connaissances dans ebXML afin d'indiquer à quelle mesure les solutions proposées dans les Chapitres 3,4 peuvent résoudre des questions posées. Enfin, nous montrons la correspondance entre les deux instances du problème de transparence sémantique présentés dans le Chapitre 2, et deux scénarios d'échanges de documents spécifiés dans ebXML.

5.1. Introduction au Commerce Électronique

Un des aspects importants du Commerce Électronique est d'intégrer toutes les parties du domaine ensemble. Chaque système correspondant à une partie doit communiquer avec un autre système. La pression et la compétition du marché exige que les entreprises s'intègrent complètement. Cependant, les entreprises actuelles, notamment les PME[1] font du commerce sans avoir une vision large de collaboration. Dans ce contexte, une idée très répandue est d'utiliser les outils d'informatique pour supporter les activités quotidiennes. Ces outils permettraient d'automatiser certains processus communs (par exemple, les transactions entre le fournisseur et le consommateur), de rendre l'entreprise plus approche du consommateur, *etc.* L'infrastructure du commerce électronique devrait comporter les champs suivants **[KR99 (44)]** :

(1) *Progiciel de gestion intégré* (PGI). Il regroupe en principe toutes les applications de gestions nécessaires à l'entreprise, que ce soit les applications de gestion dites horizontales (*e.g* comptabilité) ou verticales (*e.g* gestion de production). Les PGI disposent d'une infrastructure unique commune (bases de données, mécanismes d'échange, *etc.*) et incluent généralement des outils de coopération.

(2) *Gestion de la chaîne logistique globale.* Au niveau opérationnel, elle met en correspondance les informations de demande et de capacité de production pour l'établissement des plans de production et des paramètres de livraison.

[1]Petite et Moyenne Entreprise

(3) *Achat réalisé via un support électronique* (E-procurement). Un système EP d'une entreprise permet d'envoyer un message de commande d'un produit à un serveur supportant EP. Ce serveur transmet cette commande sous une forme définie à un fournisseur de ce produit. Le fournisseur suppose que l'entreprise prend le catalogue et la condition la plus récente.

(4) *Marché électronique*. Il est une plate-forme où le fournisseur et le consommateur se rencontrent pour effectuer les transactions commerciales.

Dans la section suivante, nous investiguons les modèles d'échanges de données existants dans le C.E.

5.2. Modèle d'échange de données EDI

5.2.1. Introduction. L'EDI est un outil au service de l'échange électronique consistant à transporter automatiquement de l'application informatique d'une entreprise à l'application informatique d'une autre entreprise, par des moyens de télécommunication, des données structurées selon des messages types convenus à l'avance [**CHI01 (20)**].

L'objectif de l'EDI est de fournir à des entreprises une plate-forme pour échanger automatiquement des données administratives et commerciales entre les ordinateurs. De plus en plus il devient un outil de communication standardisé. En effet, dans le monde EDI, les deux normes les plus importantes : ANSI X12 pour l'Amérique du Nord et EDIFACT pour le reste. Les organisations OASIS et ONU ont réuni tous les groupes professionnels qui ont développé des solutions entièrement propriétaires sur des réseaux privés, tous les acteurs d'Internet, pour concevoir un standard qui comprend à la fois une syntaxe, des dictionnaires de données et de regroupements de ces données en segments et messages. Ces mêmes groupes s'occupent des méthodes, des accords d'échange, des listes de codes, des modèles d'affaires. C'est tout un monde EDIFACT qui s'est construit. Cette méthode assure l'existence de solutions complètes, acceptées au niveau international mais elle engendre des difficultés de communication entre les spécialistes EDI et les autres acteurs des systèmes d'information.

5.2.2. EDIFACT et ses répertoires. Le standard EDIFACT comporte deux blocs principaux. Le premier est un vocabulaire qui est inclus dans les répertoires EDIFACT. Chaque élément du vocabulaire comporte : i) un nom ; ii) un indicatif numérique ; iii) une description de la notion qu'il représente (il explique sa signification conventionnelle et aide à définir le contenu de l'information fournie) ; iv) une spécification du mode de représentation de l'information (*e.g* l'espace disponible). Une partie importante de la sémantique EDIFACT s'y inclut. Ce vocabulaire est établi par les spécialistes et recouvre différentes activités : transactions commerciales, opérations logistiques, formalités administratives.

Le deuxième bloc correspond aux définitions syntaxiques comparables aux règles de grammaire. Il permet de structurer les éléments afin de pouvoir construire des segments types, puis, de structurer des segments types afin de construire des messages types. Par exemple, un segment commence par un en-tête ; le séparateur ' :' sert à séparer des éléments simples ; le séparateur '+' sert à séparer des éléments composites ; les diagrammes de branchement permettent d'organiser des groupes de segments selon leur fonction. Voici un exemple du segment PRI (en-tête) qui se compose des éléments :

PRI+AAA :24.99 : :SRP'

où "AAA" signifie "prix net" ; "SRP" signifie "prix proposé".

A partir de la syntaxe et du vocabulaire, l'EDIFACT introduit le concept de segments types. Ils sont la combinaison de mots du vocabulaire et des éléments constitutifs des messages types. Un segment comporte un en-tête, des données, des séparateurs et la fin du segment. Les segments sont identifiés par un code alphabétique à trois caractères. Il y a deux catégories de segments comme suit : i) les segments de données qui sont composés uniquement de données : les noms, les adresses, les parties de transactions, les lieux, la désignation de marchandises, les valeurs, *etc.* ii) les segments de services qui sont définis par les règles de syntaxe servant à l'échange interactif : les types de règles de syntaxe, l'émetteur du message, *etc.*

5.2.3. Accord d'échange préalable : sous-ensemble du standard. Les messages EDIFACT sont des unités finales avec une signification entière de la communication. Ils peuvent remplacer les documents en papier dans la méthode traditionnelle. Toutefois, la structuration des messages est très variée en fonction des domaines d'activité. Elle est l'un des objectifs de la négociation entre des acteurs avant d'arriver au premier échange. On dit également l'accord d'échange préalable.

L'accord d'échange préalable fournit à chaque message un diagramme de structure, une table de segments comportant le statut, la répétition de chaque segment (un sous-ensemble des segments définis), une spécification des segments : le statut, la longueur, la forme de la représentation des segments. De plus, l'accord d'échange préalable précise les méthodes à utiliser pour la transmission physique de données dans le cadre de l'application considérée. Il définit également les méthodes de codage communes qui devraient être utilisées par les acteurs, les problèmes juridiques et les problèmes de sécurité liés au transfert de l'information, *etc.*

Le standard EDIFACT fournit une base de langage pour faire de l'échange de données dans plusieurs domaines. En réalité, il est très fréquent que les échanges n'aient lieu que entre les membres d'un même groupe d'un domaine. La négociation est indispensable pour sélectionner les structures des messages conformes à son domaine, pour détailler les standards logiciels utilisés, comme par exemple le traducteur. Une quantité importante de connaissances ne peut être découverte à partir des messages échangés mais elle est décrite dans les manuels d'utilisateurs.

Exemple 5.2.1. Un message "Order" peut être écrit en EDIFACT comme suit :

(1) UNH+1+ORDERS :D :96A :UN'
// indiquer que le document est celui de l'EDIFACT et identifier le type de document : ORDERS
(2) BGM+220+AGL153+9+AB'
// le début du message
(3) DTM+137 :20000310 :102'
// indiquer la Date ou l'Heure de la commande
(4) DTM+61 :20000410 :102'
// indiquer la Date ou l'Heure du dernier jour pour la livraison
(5) NAD+BY+++PLAYFIELD BOOKS+34 FOUNTAIN SQUARE PLAZA+ CINCINNATI+OH+45202+US'
// indiquer le nom et l'adresse de l'acheteur
(6) NAD+SE+++QUE+201 WEST 103RD STREET+INDIANAPOLIS+IN+ 46290+US'
// indiquer le nom et l'adresse du vendeur
(7) LIN+1'
// la première ligne de la commande
(8) PIA+5+0789722429 :IB'QTY+21 :5'
// indiquer l'identifiant et la quantité du produit
(9) PRI+AAA :24.99 : :SRP'
// indiquer le prix
(10) LIN+2'
(11) PIA+5+0789724308 :IB'
(12) QTY+21 :10'
(13) PRI+AAA :42.50 : :SRP'
(14) UNS+S'
// indiquer que les messages suivants sont le résumé du message
(15) CNT+3 :2'
// indiquer le nombre de lignes
(16) UNT+17+1'
// indiquer le nombre du segments et de messages (répétition)

Note 5.2.2. La structure d'un message varie selon le secteur industriel. Dans certains cas, le concept du groupe de segment et la répétition de segments sont introduits. Les définitions de ces concepts dépendent aussi fortement du secteur industriel.

5.2.4. Avantages.

– Automatisation des traitements et des échanges entre applications hétérogènes. L'EDI permet les échanges entre applications hétérogènes, avec automatisation des traitements. Ainsi, il s'agit des échanges non seulement de machine à machine mais aussi de système d'information à système d'information, le tout sans que l'utilisateur ne modifie les applications, ne fournisse ou ne reçoive des informations.

– Échange d'information structurée. Il ne suffit pas d'échanger des fichiers textes ou des images, même automatiquement, pour faire de l'EDI. Encore faut-il que ces documents soient suffisamment "auto-renseignés" et structurés, ou au moins contenus dans des enveloppes permettant d'identifier clairement le contenu et les conditions de l'échange. Il faut en effet pouvoir adresser le document à l'application à laquelle il correspond, et garder une trace analysable des échanges.

5.2.5. Limites.
– Documents "fichiers plats" et illisibles pour l'homme. La complexité de ses messages vient du format très compressé à cause des contraintes de réseaux de communication.
– Coûts élevés. Ils sont dus aux coûts de développement de traducteurs, de l'intégration des traducteurs au système, de l'installation des réseaux de communication, de modifications pour de nouveaux acteurs et de négociation pour parvenir à des accords d'échange.
– Manque d'interopérabilité sémantique. Il est très difficile d'établir les relations EDI entre entreprises n'appartenant pas à un organisme commun. En effet, l'interopérabilité n'est pas assurée par la conformité à une même syntaxe, ni aux mêmes protocoles de communication. La véritable difficulté est de s'accorder réellement sur les définitions et sur les attributs d'une donnée. Les interprétations des concepts peuvent varier selon les pays, les langues et les secteurs industriels. Tout échange suppose l'accord sur des processus, des sémantiques, des règles en cas de différents, *etc.*, c'est-à-dire un accord d'échange entre les parties.
– Peu de technologies supportent l'EDI. Le monde EDI-EDIFACT reste isolé.

5.3. Modèle de langage formel

5.3.1. Motivation. Le modèle de langage formel est inspiré des lacunes du modèle EDI. Afin d'établir l'échange de données, deux acteurs du modèle EDI doivent négocier pour atteindre un accord d'échange qui inclut la manière d'interpréter les messages. Dans ce cas, la majorité de connaissances servant à interpréter les messages sont prédéfinies dans les manuels d'utilisateurs. Ainsi, la sémantique des messages n'est pas véhiculée avec ceux-ci. Cela empêche un nouvel acteur de participer au système d'échange.

Pour que tous les acteurs se comprennent, il est nécessaire qu'ils parlent le même langage. Par ailleurs, pour valider les messages et effectuer des inférences, c'est-à-dire, automatiser le traitement et la communication entre les machines, il faut formaliser ce langage. L'objectif de ce modèle est donc d'établir la transparence sémantique pour les transactions commerciales communes, et de faciliter la participation d'un nouvel acteur au système d'échange.

Un langage typique de cette direction est FLBC[2], développé par Steven O. Kimbrough et Scott A.Moore (1998). Les auteurs sont convaincus que le langage formel pour la communication doit utiliser un lexique unique et publique car cela permet d'éviter des accords spécifiques sur la structure ou sur l'interprétation propre des messages échangés. De plus, ils choisissent la logique des prédicats du premier ordre (LPPO) comme un formalisme de base pour exprimer les phases de ce langage.

5.3.2. Représentation de connaissances commerciales dans FLBC. Le langage FLBC se fonde sur un lexique et un ensemble de règles de grammaire. La partie importante de la sémantique est représentée dans ce lexique. Ce dernier présente les opérations traditionnelles comme "payer", "livrer", "créer", *etc.* ; les locutions verbales comme "promise", "assert", "declare", "invoice", *etc.* La logique modale peut être utilisée pour formaliser ces locutions.

Pour illustrer comment décrire un document commercial en utilisant ce lexique et ces règles, on considère l'exemple suivant. Un message EDI est simulé comme suit :

1. promesse : 12345
2. date-heure : 2004-02-13
3. de : s
4. a : r
5. livrer
   a) produit : g
   b) à : r
   c) par : s
   d) date-heure : 2004-02-13 + jour (2004-02-13 + 30)

On peut convertir ce message en celui de FLBC :

$$\mathsf{Promise}(12345) \wedge \mathsf{Speaker}(12345, s) \wedge \mathsf{Addressee}(12345, r) \wedge \mathsf{Cul}(12345, \text{2004-02-13}) \wedge \Box\,(\,\top \Rightarrow (\,\mathsf{K}(12345) \leftrightarrow (\mathsf{deliver}(e) \wedge \mathsf{Agent}(e, s) \wedge \mathsf{Benefactive}(e, r) \wedge \mathsf{Sake}(e, 12345) \wedge \mathsf{Theme}(e, g) \wedge \mathsf{Cul}(e, t) \wedge \leq (t, +(\text{2004-02-13}, \mathsf{day}(\text{2004-02-13}+ 30))))))$$

Ce message peut être interprété comme suit : "12345 est une promesse de s pour r qui est faite le 13/02/2004 (*cul* est une abréviation de *culminating*). La promesse concerne une délivraison du type e. Cette promesse est tenue (opérateur K) ssi s effectue une livraison du produit g du type e à r. De plus, cette livraison doit être effectuée dans 30 jours. $\Box$ est un opérateur de nécessité de la logique modale et $\top$ est la tautologie logique.

Le détail du langage FLBC peut être trouvé dans [**Kim98 (45), Kim00 (46)**].

[2]Formal Language for Business Communication

5.3.3. Avantages et Limites.

- La formalisation avec LPPO du FLBC permet de faciliter la validation du document en FLBC.
- Le lexique unique du FLBC est une grande difficulté pour le commerce actuel. En effet, le commerce électronique est un domaine hétérogène décrit par un grand nombre de concepts. De plus, la définition d'un concept peut varier en fonction de la culture et de la géographie.
- Des connaissances représentées par la logique modale dans le FLBC ne sont pas encore suffisamment exploitées. Autrement dit, ces connaissances ne sont pas formalisées dans le but de les traiter automatiquement.

5.4. Initiative ebXML

Le développement du Commerce Électronique exige une plate-forme flexible, dynamique et ouverte facilitant l'échange de données entre les partenaires en tenant compte de l'hétérogénéité de leurs activités et de leur taille. L'échange de données de façon traditionnelle avec l'EDI montre des difficultés infranchissables pour les acteurs dont les moyens ne sont pas suffisants. De plus, l'apparition de XML avec la capacité de structurer des données de ce langage nous permet de penser à des échanges de données dont la sémantique est plus riche sur Internet. Dans ce contexte, l'ebXML a été créé et il semble qu'il répondrait à ces besoins. Pourtant, nous n'avons pas tous les outils nécessaires pour réaliser les spécifications de l'ebXML. Il manque notamment d'un mécanisme capable de représenter la sémantique impliquée par les formalismes utilisés. Un tel mécanisme devrait faciliter la réalisation de ces spécifications, par exemple la composition des documents commerciaux et les manipulations des processus commerciaux.

En effet, les connaissances commerciales sont constituées de facteurs divers que nous ne réussissons pas toujours à capturer par les outils actuels. Typiquement, l'UML est considéré comme un outil puissant pour la modélisation mais il ne peut que modéliser les connaissances des processus prévisibles, déterministes et sans répudiation. Pour faciliter la capture et la modélisation des connaissances commerciales, l'UN/CEFACT propose un outil, appelé UMM[3], qui est considéré comme une extension de l'UML pour le commerce. Puisque l'ebXML utilise cet outil pour la modélisation des processus commerciaux et de la composition des documents d'échange, la sémantique capturée est caractérisée par la prévisibilité, c'est-à-dire que toutes les transactions ont des rôles, des bornes de temps, des succès et échec bien déterminés ; par la capacité de créer des connexions légales ; et par la non-répudiation, c'est-à-dire que les conduites commerciales légales s'imposent. Bien que cet outil facilite déjà la représentation de connaissances, la sémantique informelle de l'UML dont l'ebXML hérite nous empêche d'échanger des données de façon transparente sémantique. En outre, l'ebXML se sert également du formalisme des règles de production pour représenter une partie des connaissances, une définition de la sémantique formelle, commune et compatible à celles des formalismes utilisés n'est pas établie automatiquement. Il existe des efforts de l'ebXML dans le but de surmonter ces difficultés, par

[3]UN/CEFACT Modeling Methodology

exemple le mécanisme de contexte, mais ils ne peuvent que répondre partiellement au problème. Par ailleurs, si l'on rend le scénario d'échange plus flexible en acceptant des connaissances sans les partager, les difficultés se multiplieront.

En sachant que nous n'arrivons jamais à capturer *complètement* la sémantique impliquée dans un langage (langage naturel, langage de modélisation, *etc.*), la formalisation de la sémantique vise toujours à un ensemble de services d'inférences dont les spécifications d'ebXML ont besoin.

Les sections suivantes décrivent les parties les plus importantes de l'ebXML dans lesquelles les connaissances commerciales sont impliquées. Une analyse détaillée sur ces parties nous permet d'identifier certains problèmes du modèle de la représentation de connaissances dans ebXML. Nous allons montrer que la version simplifiée de ces problèmes correspond aux instances du problème de transparence sémantique étudiées dans le Chapitre 2. Par conséquent, les solutions proposées dans les Chapitres 3 et 4 sont applicables aux problèmes identifiés dans ebXML.

5.4.1. Introduction à XML et ebXML. XML est un langage de balise permettant d'introduire à la fois ce que l'on communique (données) et les informations sur ce qui est communiqué (description de données). Il permet également une séparation stricte entre la structure (la définition de la structure d'un document), le contenu (la description du contenu d'une occurrence du document) et la présentation (la "mise en page") du document. Le langage XSLT crée un pont entre XML et le monde WEB en permettant de transformer les documents XML en HTML. Grâce aux principes de conception simples et efficaces, on trouve de plus en plus de domaines d'applications de XML, par exemple, le format de documents pour l'échange d'inter-applications, le publication sur le WEB (création de pages dynamiques, transmission et mise à jour de données, présentation sur de multiples supports). L'ebXML est une initiative profitant des capacités du XML pour développer une plate-forme eCommerce.

L'ebXML propose un ensemble des spécifications permettant à l'entreprise, quelle que soit sa taille, quelle que soit l'industrie, de faire ses affaires sur Internet. Il tire les leçons de l'expérience de l'EDI et suit une approche plus méthodologique.

La Figure 5.4.1 qui illustre les transactions de l'ebXML est adaptée de la "Spécification de l'Architecture Technique de ebXML" [**ebX01h (33)**].

(1) L'entreprise A examine le Référentiel d'ebXML (Repository), notamment la bibliothèque des processus commerciaux, pour déterminer si l'ebXML est approprié à son business ou quels sont les exigences d'implémentation de l'ebXML. Dans le cas où il n'existe pas de processus commercial commun, l'entreprise A peut définit un nouveau processus commercial selon [**ebX01d (29), ebX01b (27)**].

(2) En se basant sur les informations disponibles dans le Référentiel d'ebXML, A peut construire ou acheter l'implémentation ebXML adaptée à ses transactions ebXML.

(3) A crée et enregistre un CPP (Collaboration Protocol Profile) dans le Référentiel. CPP contient les informations nécessaires pour qu'un partenaire

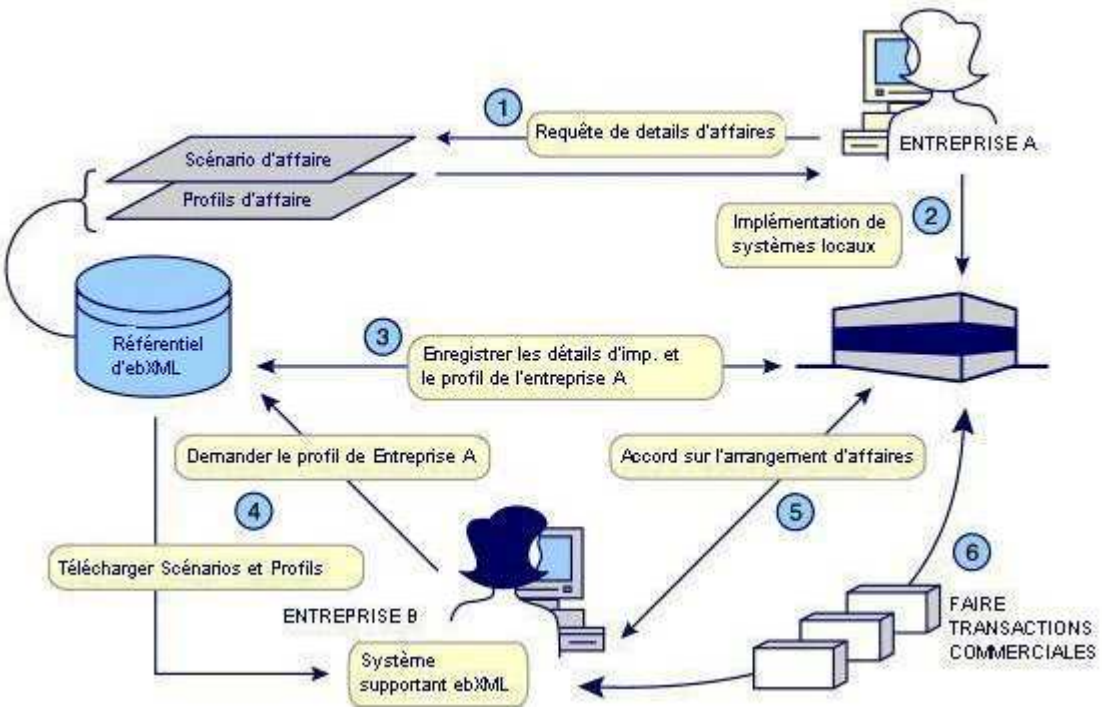

FIG. 5.4.1. Interaction entre deux entreprises ebXML - une vue de haut niveau

potentiel puisse déterminer le rôle commercial de A et le type de protocole pour ce rôle.

(4) Après l'enregistrement du CPP de A, B peut rechercher et examiner le CPP de A pour savoir si celui-ci est compatible avec le CPP de B. Pour ce faire, B essaie de déterminer l'intersection entre les deux CPP afin d'obtenir un CPA (Collaboration Protocol Agreement).

(5) Si les deux CPP sont compatibles, B prend contact avec A pour obtenir un acceptation de A sur le CPA créé. Si les deux entreprises aboutissent à un accord commun, ils configurent leurs système en se basant sur le CPA obtenu.

(6) A, B effectuent les transactions commerciales selon le CPA obtenu. C'est-à-dire que les deux entreprises utilisent les Processus Commerciaux et les Composants de Bases (Core Components) définis dans le CPA.

5.4.2. Représentation de connaissances commerciales dans ebXML. Dans cette section nous montrons comment l'ebXML capture et représente les connaissances commerciales. Notre objectif est d'indiquer quelles sont des informations nécessaires dont chaque partenaire doit disposer avant le premier échange, quel point atteint l'automatisation de la capture de ces informations.

Processus Commerciaux. L'ebXML utilise le mécanisme, qui s'appelle UMM, pour capturer les détails d'un scénario commercial spécifique. Un processus commercial (Business Process) décrit comment un partenaire joue son rôle, fait partie d'une relation avec un autre partenaire, prend la responsabilité de la collaboration d'échange, afin de faciliter l'interaction. L'interaction entre des rôles est considérée comme un ensemble de transactions commerciales. Chaque transaction s'exprime par un échange de documents commerciaux. Un document commercial se compose des objets d'informations commerciales (Business Information Objects). Un objet d'informations commerciales, à son tour, peut se composer des composants de base (Core Components) (*e. g* les concepts documents commerciaux, objets d'informations commerciales). De plus, l'ebXML propose les schémas de la spécification des Processus Commerciaux (Business Process Specification Schemas) comme un sous-ensemble sémantique d'UMM ayant pour but de configurer le système afin d'exécuter les transactions commerciales.

Le schéma de la spécification des processus commerciaux représente la sémantique d'un processus commercial sous une forme qui implique une collaboration commerciale avec un autre partenaire (chaque schéma qui est décrit par deux versions UML et XML permet de configurer le système ebXML en exécution). Une collaboration consiste en un ensemble de rôles collaborants à travers des transactions chorégraphiques effectuées par des échanges de documents (une collaboration ne fournit que le contexte pour composer des documents commerciaux au lieu d'en définir et elle peut également utiliser ces documents). Chaque transaction commerciale peut être implantée en utilisant des patterns standard. La spécification des processus commerciaux servent à former les CPP et les CPA.

Un processus d'analyse pour capturer la sémantique d'un Processus Commercial spécifique s'effectue par une équipe de spécialistes. Cette équipe est encouragée à utiliser les outils UML et BPAW (ebXML Business Process Analysis WorkSheets) fourni par ebXML. L'analyse des informations commerciales identifie les documents commerciaux qui sont inclus dans les transactions. Les spécifications du Processus Commercial et les définitions de documents commerciaux résultent des cette analyse. Ces spécifications sont enregistrées dans une bibliothèque commerciale (Business Library). Cette dernière est un référentiel ebXML des spécifications des processus commerciaux et des objets d'informations commerciales. Cela implique qu'une bibliothèque commerciale publique supporte fortement l'interopérabilité commerciale.

Composants de Base. Le Processus Commercial détermine les caractéristiques du charge final (payload) du document commercial. Parmi ces caractéristiques, il y en a qui varient dramatiquement selon les industries alors que les autres caractéristiques ne changent pas. Si l'on considère un document comme un ensemble de composants où chacun est un "bloc" contenant des informations commerciales, alors les Composants de Base sont des composants capable d'apparaître dans plusieurs circonstances et dans différents domaines commerciaux. Un Composant de Base, s'appelle CCT (Core Component Type) dans le dictionnaire de composants de base, est un bloc commun et s'utilise à travers des industries, donc il est libre de contexte. Les composants de domaine, appelés BIE (Basic Information Entity) et ABIE (Aggregate Basic Information Entity), dans le dictionnaire de Composants de Base sont spécifiques à

un contexte commercial donné. Ainsi, les composants de domaine portent une sémantique. Par exemple, "*quantity* " n'a aucun sens commercial, mais "*quantity shipped*" porte une signification. L'objectif final des Composants de Base est de supporter la réutilisation des composants même si les Processus Commerciaux avec les définitions des documents et les informations contextuelles sont variés.

Les informations suivantes doivent être définies pour chaque Composant de Base : nom, description de la nature et du sens, identifiant unique, synonymes (les mots ou phrases ont le même sens que le nom du composant de base. Ils servent à capturer des noms communs d'un Composant de Base), les composant réutilisés, type de données, remarques (exemples, références), CCT et finalement, la convention de nomination.

On peut constater une relation étroite entre les Processus Commerciaux et les composants de base. En effet, la définition d'un document commercial utilise des informations contextuelles que les processus commerciaux fournissent. La combinaison de ces informations contextuelles définit le but commercial du processus correspondant. Un composant commun qui est construit pour un but commercial spécifique permet la réutilisation. D'autre part, le document commercial est toujours construit à partir des objets d'informations commerciales, composants de domaine, Composants de Base. En analysant les processus commerciaux, les objets d'informations commerciales appropriés sont découvertes. Ces objets peuvent être extraits de la bibliothèque commerciale, bibliothèque de domaine (composant de domaine) ou bibliothèque de base (composant de base).

La Figure 5.4.2 montre l'utilisation de l'UMM et de l'UML dans ebXML pour capturer et représenter les connaissances commerciales.

L'ebXML propose un ensemble de fiches descriptives, les diagrammes d'état et d'activité conformément à l'UMM permettant de représenter plus pratiquement les connaissances commerciales (1). A partir de ces fiches, l'utilisateur peut définir les Processus Commerciaux décrivant ses affaires. L'ebXML consiste en deux parties importantes portant la sémantique. La première partie, appelée BPSS (Business Process Specification Schema), décrit un sous-ensemble de la sémantique (3). Le BPSS a pour objectif de fournir des informations de configuration du système en exécution (8). L'utilisateur peut extraire des fiches descriptives les éléments et relations (processus commerciaux, modèle d'information) conformément au BPSS afin de les introduire aux BSP (Business Process Specification). La deuxième partie, appelée Composants de Base (Core Component), décrit les composants libres de contexte permettant la réutilisation pour la composition de documents (5). Ces composants sont dérivés du lexique et du méta-modèle en UMM (4). Le document commercial se compose de Composants de Base et d'informations contextuelles extraites de fiches descriptives (5,6,7). Les connaissances suivant les (6) et (7) sont représentées par des règles de contexte.

Une contribution de l'ebXML est de définir le BPSS, un sous-ensemble sémantique de l'UMM, permettant aux utilisateurs à la fois de configurer des interfaces, logiciels du système d'échange en exécution (*e.g* chorégraphie des transactions commerciales) et de définir le BPS conformément à ses affaires. En outre, une autre contribution de l'ebXML est de construire les composants de base réutilisables. Ces Composants de Base avec les règles de contexte portant les informations contextuelles fournissent les

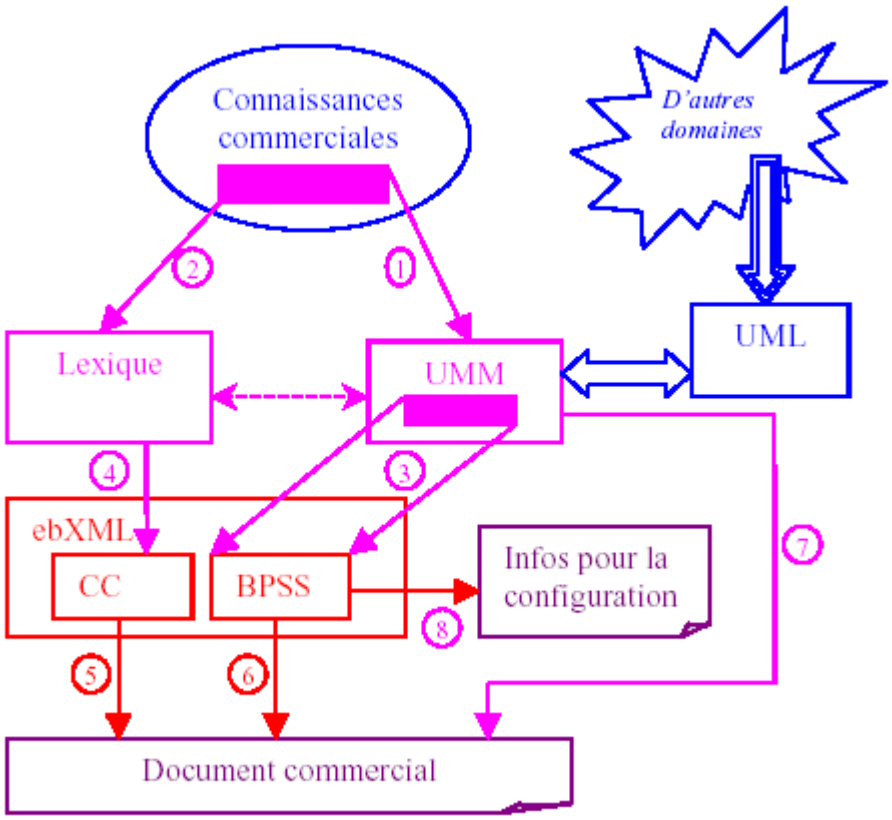

FIG. 5.4.2. Utilisation de UMM et UML dans ebXML

éléments pour composer les documents commerciaux. Ces règles résultent du modèle d'information UMM (fiches descriptives).

Mécanisme de contexte. La réutilisation des composants de base est un objectif conceptuel important de l'ebXML car elle décide du succès de l'interopérabilité commerciale. La variation dramatique des caractéristiques des processus commerciaux à travers des industries nous empêche de construire une bibliothèque suffisante de composants communs. C'est pourquoi l'idée sur une telle bibliothèque prédéfinie ne s'adapte pas à cette situation. Une nouvelle approche au problème est de construire dynamiquement des composants communs demandés en modifiant des composants de base. Les modifications se basent sur les contextes qui viennent des processus commerciaux en question. Pour éclaircir ce mécanisme important, nous montrons comment ebXML définit le contexte et comment il applique le contexte aux composants de base.

Le contexte dans lequel un Processus Commercial a lieu est défini comme un ensemble de catégories contextuelles et des valeurs associées. Par exemple, un fabricant

de colle vend ses produits à un fabricant de chaussures, les valeurs contextuelles sont les suivantes :

Catégorie contextuelle	Valeur
Processus	Approvisionnement
Classification de produit	Colle
Région d'acheteur	France
Région de vendeur	Etats-Unis

Grâce aux informations contextuelles définies comme ci-dessus, le système est capable d'éviter des conflits de données, exprimer les relations commerciales entre des données, découvrir des composants de base, *etc.*

Par ailleurs, l'ebXML définit un modèle hiérarchique pour les composants de base avec le contrôle contextuel. La racine de la hiérarchie, appelé "Core Component Type", ne contient aucune information sur la sémantique. Par contre, "Basic Information Entity" et "Aggregate Information Entity", qui sont les descendants de "Core Component Type", décrivent des facteurs sémantiques. L'application des règles de contexte modifie "Basic Information Entity" et "Aggregate Information Entity" pour obtenir les composants appropriés au Processus Commercial. Ces composant obtenus constituent effectivement les documents commerciaux.

Notons que l'application des règles de contexte aux Composants de Base peut engendrer des incompatibilités, par exemple, un conflit de résultats. En effet, on considère les règles suivantes :

(1) Si un acheteur est de la région Etats-Unis, les descriptions des produits n'incluront pas les lignes d'articles dans la commande.

(2) Si un vendeur est en France, la description des produits sera incluse dans la commande.

(3) Si l'industrie d'un acheteur est l'automobile, la catégorie de produit sera ajoutée aux lignes d'articles de la commande.

(4) Si une entité d'informations catégorique d'un produit existe et l'industrie d'un vendeur est chimique, un attribut sera ajouté à la catégorie du produit pour indiquer la toxicité du produit dans cette catégorie.

Si les conditions des règles (1) et (2) sont vérifiées, ces règles sont appliquées à une ligne de produit dans le document. Dans ce cas, on ne sait pas si la description des produits sera incluse dans la commande. De plus, le résultat obtenu dépend de l'ordre dans lequel les règles sont appliqué. En effet, si les conditions des règles (3) et (4) sont vérifiées, ces règles sont appliquées à une ligne de produit dans le document. Si la règle (4) est appliquée avant, l'attribut de toxicité ne sera pas introduit car la catégorie de produit n'existe pas encore. Par contre, si la règle (3) est appliquée avant, l'attribut de toxicité y sera introduit.

Composition du document commercial. Maintenant, nous essayons d'examiner comment composer de façon automatique un document commercial en s'appuyant sur les éléments présentés dans les sections ci-dessus. Un canevas de l'ebXML présente un mécanisme fondé sur les règles de contexte, qui permet d'assembler les Composants de Base pour obtenir un document attendu.

La composition/interprétation des documents commerciaux à l'aide du mécanisme de contexte est capable de répondre à la différence de vocabulaires des acteurs. Et pourtant, cette différence est limitée par la spécification ebXML aux catégories contextuelles autorisées. Rien n'assure que ces catégories soient exhaustives.

Le processus d'assemblage d'un document utilise des règles pour extraire les Composants de Base appropriés du Référentiel ebXML. L'idée principale de ce processus s'appuie sur le contexte. Ce dernier est établi à l'aide de l'analyse du processus commercial. Cela fournit un canevas permettant d'adapter des Composants de Base aux besoins commerciaux spécifiques en conservant la transparence du processus de transformation. Le composant de base spécifique dépendant du contexte est utilisé pour la composition d'un document alors que la définition du composant générique reste utile pour l'interaction avec un partenaire dans un contexte différent (*i.e* industrie ou région différente). Comme mentionné, une règle de contexte associe chaque catégorie contextuelle à des valeurs différentes. Elle indique également quelles actions sont effectuées lorsque les informations contextuelles remplissent la condition de la règle. Les actions peuvent ajouter l'information à la définition d'un composant de base ou en éliminer.

L'ebXML propose deux ensembles de règles : règles d'assemblage et règles de contexte. Les deux sont représentés en XML (avec DTD). Les règles d'assemblage sont, par exemple, "CreateGroup", "CreateElement", "Rename", *etc.* Les règles de contexte sont, par exemple, "Condition", "Add", "Subtract", *etc.* Le résultat d'un processus d'assemblage est le document dont l'en-tête est de la forme suivante :

<ELEMENT Document (Taxonomy+, Assembly, ContextRules?, Component+)>

où "Taxonomy" pointe vers le contexte spécifique qui se combine avec les règles d'assemblage et de contexte.

Exemple 5.4.1. Les règles de contextes peuvent être exprimées comme suit :

```
<!ELEMENT   ContextRules   (Rule+)>
<!ATTLIST   ContextRules
            version        CDATA              #IMPLIED
            id             ID                 #IMPLIED
            idref          IDREF              #IMPLIED
>
<!ELEMENT   Rule           (Taxonomy+,        Condition+)>
<!ELEMENT   Apply          (exact|hierarcal)  "exact"
            Order          CDATA              #IMPLIED
            id             ID                 #IMPLIED
            idref          IDREF              #IMPLIED
>
<!ELEMENT   Taxonomy       EMPTY>
            Context        CDATA              #REQUIRED
            ref            CDATA              #REQUIRED
            id             ID                 #IMPLIED
            idref          IDREF              #IMPLIED
>
<!ELEMENT   Condition      (Action,Condition  ,Occurs)+>
```

```
<?xml version="1.0" ?>
<!DOCTYPE ContextRules SYSTEM "contextrules.dtd">
<ContextRules id="CalAer">
<Rule apply="hierarchical">
<Taxonomy context="Geopolitical" ref="http ://ebxml.org/classification/ISO3166"/>
<Taxonomy context="Industry"
   ref="http ://ebxml.org/classification/industry/aviation"/>
   <Condition test="$Geopolitical='United States'">
      <Action applyTo="//Buyer/Address">
      <Occurs>
            <Element >
            <Name>State</Name>
            </Element>
      </Occurs>
      <Add after="@id='fred'">
            <CreateGroup type="choice">
            <Element >
                  <Name>Floor</Name>
                  <Type>string</Type>
            </Element>
            <Element >
                  <Name>Suite</Name>
                  <Type>string</Type>
```

```
            </Element>
            </CreateGroup>
        </Add>
        ...
        </Action>
        </Condition>
</Rule>
</ContextRules>
```

L'étude sur la représentation des connaissances dans les composants de l'ebXML montre que la sémantique de l'ebXML est impliquée dans celle de diagrammes UML et de Composants de Base. Autrement dit, le formalisme UML avec les explications des termes situés dans l'ontologie est capable de représenter toutes les connaissances dont l'ebXML a besoin. En effet, le diagramme de classe UML est utilisé pour décrire le méta-modèle du Composant de Base permettant d'y introduire les informations contextuelles. Les Composants de Base avec la sémantique sont instanciés de ces diagrammes. Le BPSS est également représenté par un diagramme d'UML dans lequel la sémantique des collaborations et des transactions d'un processus commercial est capturée. Les diagrammes d'état et d'activité d'UML sont également utilisés pour exprimer les comportements d'objets et les interactions entre eux. En outre, la composition de documents commerciaux basée sur les connaissances représentées et sur le mécanisme de contexte exigerait sans doute un autre formalisme pour représenter les règles de contexte. Chaque règle est représentée par deux parties dont le conséquent implique la modification d'un composant de base. Le formalisme souhaité pour ces règles devrait être capable de franchir les problèmes cités dans l'exemple ci-dessus (conflit de résultats, dépendance de l'ordre appliqué).

Avantages.

(1) Utiliser l'approche orientée-objet avec l'outil d'UML dans la modélisation des données et des transactions. Cette approche permettra de capturer une partie importante des connaissances commerciales, y inclus la dynamique, qui sera modélisée comme les Processus Commerciaux. Cela permet également de faciliter la réutilisation en construisant une bibliothèque des Processus Commerciaux communs. Si cette bibliothèque s'installe dans le Référentiel ebXML, les PMEs peuvent y accéder pour rechercher les processus appropriés à leur affaire. Les processus trouvés peuvent subir quelques modifications afin de tenir compte des spécificités de leur entreprise.

(2) Améliorer l'interopérabilité sémantique grâce à la conception des Composants de Base et le mécanisme de contexte. Les composants de base permettent également la réutilisation mais l'objectif principal est toujours l'interopérabilité. Le mécanisme de contexte est un moyen capable de résoudre les problèmes traditionnels d'un acteur qui doit faire du commerce avec plusieurs partenaires appartenant à différentes industries. Les problèmes sont les suivants : la même donnée (le même concept) porte différents noms ou se trouve à des places différentes dans le document. La solution exige souvent

une analyse des vocabulaires supportés par les acteurs pour arriver à des transformations entre eux. Il est évident que le mécanisme de contexte permet une réduction considérable sur le coût d'analyse des vocabulaires. Le deuxième type de problèmes traditionnel est que la même donnée (le même concept) se situe dans différents processus différents ou est interprétée par des cultures différentes. Pour ce type de problèmes, on peut également trouver une solution grâce au mécanisme de contexte avec la flexibilité du XML mais parfois la performance doit être sacrifiée (la complexité du document XML s'accroît si l'on veut éviter l'ambiguïté).

(3) Utiliser le langage de balise XML comme un format de transport des documents échangés. Cela permet de profiter des outils du XML (*e.g* XSLT) pour faciliter la traduction entre XML et un autre format utilisé par les applications commerciales.

Limites.

(1) Par nature, l'approche orientée-objet ne permet pas de capturer suffisamment les connaissances qui sont impliquées dans le commerce. La relation de généralisation-spécialisation entre des composants de base ne suffit pas d'exprimer la relation entre les occurrences du même concept dans un document. Par exemple, un concept (Taxe) se situant dans une ligne d'article aurait potentiellement une sémantique différente de celle du même concept se situant dans la tête du document. Dans ce cas, il exige une "connaissance" supplémentaire sur la relation généralisation-spécialisation. De plus, l'absence d'un formalisme du modèle limite la performance des algorithmes d'inférences. Par exemple, dans le processus d'évolution des composants communs (composants de base + contexte), il est possible que les composants modifiés deviennent très similaires malgré la différence des Composants de Base dont ils dérivent. Dans ce cas, la similarité ne peut pas être détectée.

(2) Le mécanisme de contexte résout l'interopérabilité sémantique en utilisant les règles de contexte ou d'assemblage, qui sont indépendantes aux Composants de Base et viennent des processus commerciaux. Cela peut engendrer les incompatibilités cités dans la Section 1.4.2.2. Tant que ces règles s'appliquent "naïvement" aux Composants de Base, ces incompatibilités persistent. La solution a besoin d'une intégration cohérente des informations contextuelles dans les Composants de Base en conservant la réutilisation.

(3) Malgré les efforts pour organiser les Composants de Base et introduire le mécanisme de contexte, l'ebXML manque toujours d'un modèle formel sur lequel la représentation sémantique se fonde. Seul un modèle formel peut permettre de valider des documents échangés et d'améliorer le traitement automatique de composition et d'interprétation de documents "en ligne". Pour cette raison, l'ebXML est toujours loin d'un modèle d'échange de données où la transparence sémantique est établie.

5.4.3. Synthèse sur les modèles. Les trois modèles étudiés montrent les enjeux auxquels est confronté l'échange de données sous forme de document. L'établissement de la transparence sémantique pour l'échange de données nécessite une représentation formelle des connaissances du domaine. Une telle représentation facilite l'intégration, sémantiquement, d'une base de connaissances à une autre. La composition et l'interprétation des documents échangés dans l'EDI s'appuient sur la sémantique prédéfinie dans les dictionnaires et accords d'échange préalables. D'une part un acteur ne peut introduire une modification d'un concept existant ou un nouveau concept sans accord préalable, d'autre part le système d'échange n'est accessible que pour les membres du groupe qui utilisent le même dictionnaire, le même accord d'échange.

L'ebXML a tiré les leçons de l'EDI et facilite la participation d'un nouvel acteur au système d'échange en rendant les spécifications, les définitions accessibles à tous. De plus, l'ebXML fournit un mécanisme, malgré ses lacunes, permettant aux acteurs de spécialiser ces spécifications pour obtenir celles qui sont adaptées à leurs affaires. Cependant, il manque d'une représentation formelle de la sémantique et un mécanisme d'intégration des sémantiques différentes. La représentation formelle manquante facilitera la validation des documents échangés et le traitement automatique des connaissances. Celle-ci avec le mécanisme d'intégration permettra d'introduire de façon "en ligne" un nouveau concept dans un document, c'est-à-dire la possibilité d'introduction et d'interprétation de nouvelles connaissances.

Le modèle du langage formel FLBC a bien conscience de ces aspects mais la conception fondée sur un lexique unique n'est pas appropriée à l'hétérogénéité des concepts et des cultures commerciaux. En outre, certains éléments de la sémantique du langage naturel qui sont impliqués dans le FLBC ne sont suffisamment pas traités.

5.5. Deux scénarios d'échange de données dans ebXML

Comme mentionné, un système qui supporte complètement l'ebXML sera très complexe. Ainsi, dans le cadre de ce travail, nous nous concentrons sur deux scénarios d'échanges qui sont inspirés de la spécification de ebXML présentée dans les sections précédentes. Un système complet supportant ebXML doit englober les deux scénarios comme des services de base. Le premier scénario d'échange correspond à un modèle dont la sémantique est statique, c'est-à-dire que toutes les définitions des concepts ne changent pas tout au long du processus d'échange. Le deuxième scénario d'échange est une extension du premier en prenant en compte le contexte d'échange. Si l'on considère qu'un document commercial est constitué par des termes (ou concepts), la définition de ces termes peut être changée en fonction du contexte d'échange. Cette section décrit ces deux scénarios et la correspondance entre ces scénarios et les instances du problème de transparence sémantique (PTS).

5.5.1. Échange de données sans contexte : première instance du PTS. Les Processus Commerciaux et Composants de Base couvrant la sémantique commerciale dans ebXML sont représentés comme une ontologie basée sur la Logique

de Description. Cette ontologie partagée par les acteurs qui participent au système d'échange, est appelée *ontologie partagée*. Cette dernière qui définit les concepts primitifs et les concepts de base, correspond au Composant de Base et à une partie des Processus Commerciaux. Chaque acteur peut définir une ontologie propre à lui qui dérive de l'ontologie partagée. Les concepts spécifiques d'un sous-domaine peuvent être introduits directement dans une ontologie dérivée ou définis via les concepts dans l'ontologie partagée. Les ontologies dérivées contiennent les BIE (Basic Information Entity) et ABIE (Aggregate Basic Information Entity).

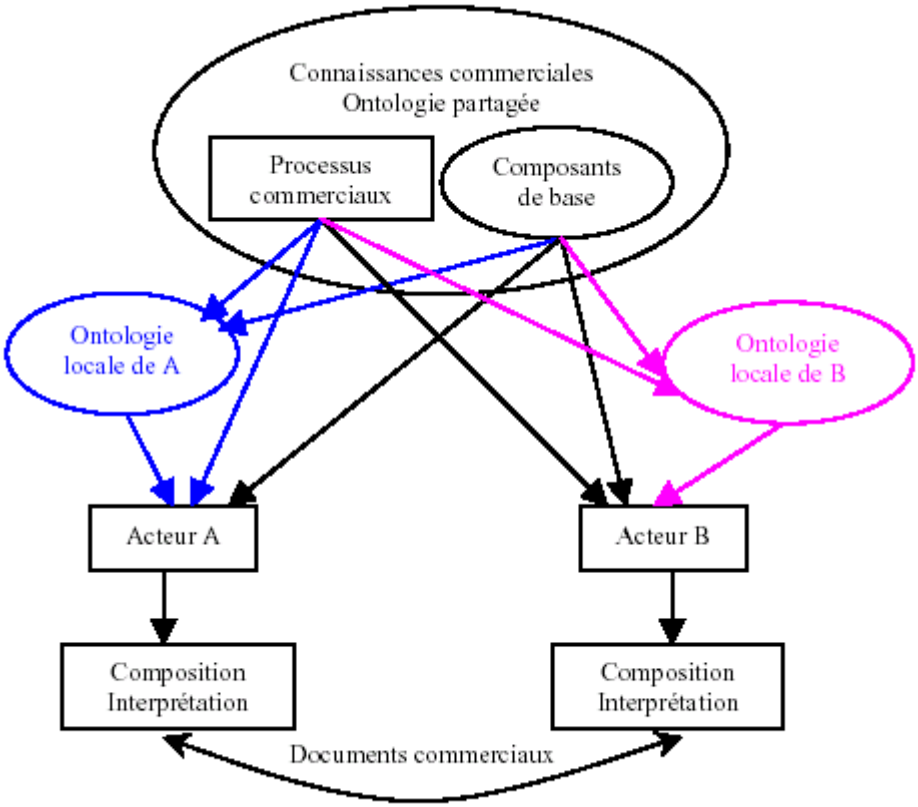

FIG. 5.5.1. Scénario d'échange de données sans contexte

Les éléments constitutifs du scénario sont illustrés dans la Figure 5.5.1. Notons que dans ce scénario l'introduction de la sémantique dans les BIE et ABIE doit être effectuée manuellement par les acteurs. Si toutes les ontologies dérivées utilisent le même langage L.D et un document à échanger est considéré comme une liste de concepts, la composition d'un document à envoyer ou l'interprétation d'un document reçu peut être automatisée. En effet, l'équivalence de deux concepts portant différents noms dans différentes ontologies dérivées peut être découverte à l'aide des services d'inférence standard. Si deux ontologies dont les langages L.D sont différents *i.e* il est possible qu'un acteur puisse recevoir un concept qui n'est pas exprimable dans son

ontologie dérivée. Dans ce cas, l'approximation d'un concept dans un langage L.D L_1 par un concept dans un langage moins expressif L_2 est nécessaire.

En bref, afin de supporter ce scénario dans un système conforme à ebXML, d'une part l'introduction du langage $\mathcal{DLR}$, qui est présenté dans la Section 2.2.1, est nécessaire. Ce langage expressif permet de formaliser tout le diagramme de classes UML. Toutefois, les services d'inférence non-standard développés dans le cadre de ce travail ne supportent pas le langage $\mathcal{DLR}$. Un langage L.D qui est moins expressif que $\mathcal{DLR}$ peut être utilisé pour capturer partiellement les Processus Commerciaux et Composants de Base à partir des diagrammes UML. D'autre part, ce scénario nécessite également les services d'inférence standard et d'approximation. Ces services d'inférences sont étudiés dans les Chapitres 2 et 3. Un exemple de ce scénario est illustré dans la cadre du prototype ONDIL, la Section 6.4.

5.5.2. Échange de données avec le contexte : deuxième instance du PTS. Le commerce électronique est considéré comme *i)* un marché énorme et ouvert (plusieurs entreprises sont capables d'y entrer et d'en sortir librement) ; *ii)* un marché avec des relations dynamiques (des associations sont formées et annulées facilement et fréquemment) ; *iii)* un marché avec l'hétérogénéité des activités d'acteurs. L'ambition de l'ebXML est de répondre aux besoins du marché avec tels caractéristiques. Il sera infaisable si tous les acteurs se restreignent à utiliser le même vocabulaire avec les compréhensions identiques sur les termes de ce vocabulaire pour échanger des documents. De plus, il est très fréquent que la résolution des différences sémantiques entre les partenaires hétérogènes s'effectue dynamiquement pendant l'échange, c'est - à - dire de façon "en ligne". C'est pourquoi le fait de supporter un scénario d'échange avec la sémantique dynamique est indispensable pour l'ebXML.

La spécification d'ebXML propose le mécanisme de contexte pour résoudre le problème dans lequel un Composant de Base est interprété différemment en fonction du sous-domaine commercial. Afin d'échanger des données, les acteurs doivent partager la compréhension d'un Composant de Base à l'aide des règles de contexte. Cela implique que les règles de contexte imposent les modifications nécessaires sur les Composants de Base pour harmoniser les significations de ces composants. Ainsi, le deuxième scénario étend le premier en autorisant le changement d'une définition de concept dans une ontologie dérivée. L'essentiel de ce scénario est d'atteindre en exécution la harmonisation des concepts portant le même nom dans différentes ontologies dérivées. L'élément important est que chaque ontologie du système doit comporter un nouveau formalisme qui représente les règles de contexte. De plus, une intégration sémantique du nouveau formalisme dans l'ontologie basée sur la L.D doit être étudiée.

Il est évident que les problèmes posés dans le scénario d'échange de données avec le contexte correspondent à la deuxième instance du problème de transparence sémantique qui est formulée dans le Chapitre 2. En effet, si la modification d'une ontologie peut se traduire en la révision de l'ontologie, alors une règle de contexte est représentée par une règle de révision qui est définie dans la Section4.4. D'autre part, l'intégration des règles de révision dans l'ontologie exigerait un formalisme hybride dont la sémantique se fonde sur la Logique de Description. La construction d'un tel

formalisme hybride est également présentée dans la Section 4.4. Un exemple de ce scénario est illustré dans la cadre du prototype ONDIL, Section 6.4.

5.6. bcXML : une spécialisation d'ebXML

5.6.1. Contexte et Objectif. Même s'il existe plusieurs canevas qui apportent de nouvelles technologies au Commerce Électronique, l'utilisation de ces technologies au niveau de la conception ou la construction d'un projet émerge lentement dans le secteur de la construction. La raison importante pour cela est que les technologies existantes ne sont pas très bien adaptées à la nature du secteur de la construction. Elles nécessitent souvent des normes internationales complémentaires. Dans ce contexte, le projet eConstruct[4] ou bcXML[5] [**VTL$^+$01 (68)**] se propose de développer un langage de communication basé sur XML et sur les technologies associées pour le secteur de la construction.

Une difficulté majeure de ce secteur est la communication à travers des barrières organisationnelles, due au problème d'échange de connaissance entre différents systèmes d'informations. L'objectif du projet bcXML est de remplir la lacune que constitue ce manque de communication pour que tous les acteurs du secteur puissent profiter des avantages des nouvelles technologies.

5.6.2. Approche du projet eConstruct et langage bcXML. Techniquement, le projet s'appuie sur le langage XML pour développer le langage bcXML permettant aux applications d'échanger des informations spécifiques au domaine. Tous les concepts définis dans bcXML seront disponibles et traduisibles dans différents langages et selon différentes vues spécifiques à chaque organisation.

Méthodologiquement, le langage bcXML peut être considéré comme une projection de l'ebXML sur le secteur de la construction. En effet, le bcXML comporte deux parties correspondant aux parties de l'ebXML. Le méta-schéma bcXML ou XTD (XML Taxonomy Definition) est indépendant de la sémantique du secteur de la construction. Le format de XTD est harmonisé avec les autres spécifications reconnues dans le secteur, *e. g* LexiCon. Le XTD sert à définir les objets et propriétés d'objets. En se basant sur le XTD générique, un acteur (*e.g* un vendeur) peut publier les informations sur ses produits sous forme d'un catalogue en spécialisant ou étendant ce XTD (*e.g* LexiCon développé par STABU). On peut se rendre compte que le XTD correspond aux Composants de Base dans l'ebXML et le catalogue correspond aux BIE (Basic Information Entity) et ABIE (Aggregate Basic Information Entity). L'échange de données éventuel entre les acteurs qui possèdent différents catalogues conforme à bcXML pose le même problème que celui dans l'ebXML.

Dans le cadre de cette thèse, nous implémentons un système, appelé ONDIL, permettant de modéliser une partie de la Taxonomie LexiCon comme une ontologie

[4]Electronic Construction

[5]Building and Construction XML

partagée et certains catalogues comme des ontologies dérivées. Le prototype ONDIL sera le sujet du Chapitre 6.

5.7. Conclusion

Dans ce chapitre, nous avons présenté les modèles d'échange de données existant dans le commerce électronique. Notamment, nous avons décrit le spécification de l'ebXML. A partir de ces analyses, nous avons identifié la problématique de chaque modèle d'échange de données et montré que l'ebXML définit un canevas assez générique et ouvert pour que les PME puissent en profiter. Nous avons indiqué dans l'ebXML les formalismes utilisés pour représenter les connaissances commerciales. Il s'agit du formalisme UML et une sorte de règles de production. Nous avons montré que ces formalismes peuvent être encapsulés dans la Logique de Description. C'est-à-dire qu'une partie importante de la sémantique d'ebXML peut être formalisée en utilisant la Logique de Description.

Par ailleurs, puisque l'ebXML se conforme à un grand nombre d'acteurs dont l'objectif est différent, nous nous limitons aux situations dans lesquelles certaines tâches spécifiées peuvent être automatisées. Ces tâches sont généralisées dans le but d'obtenir deux scénarios d'échange de données importants qu'un système complet conforme à l'ebXML peut réutiliser. Une des motivations de ce travail est d'établir la correspondance entre ces scénarios et les deux instances du problème de transparence sémantique présentées dans le Chapitre 2. Cela implique que les résultats obtenus dans le Chapitres 3 et 4 nous permettent de construire un système conforme à l'ebXML dans lequel certaines tâches sont considérées comme des services d'inférences développés. Finalement, nous avons présenté une spécialisation de l'ebXML pour le secteur de la construction : le langage bcXML. Un prototype de cette thèse est consacré à modéliser la Taxonomie bcXML (LexiCon) comme une ontologie basée sur le formalisme hybride introduit dans le Chapitre 4.

CHAPITRE 6

Mise en œuvre dans ONDIL

Le projet ONDIL a commencé au milieu de l'année 2003. Ce projet fournit un outil dont le but est d'aider à concevoir et de maintenir des ontologies basées sur la Logique de Description. Ce chapitre est organisé en six sections. Dans la première section, nous passons en revue les projets les plus représentatifs et nous présentons les services d'inférence implémentés et offerts par ces projets. Nous décrivons le modèle formel pour la représentation de connaissances dans ONDIL dans la deuxième section. La Section 6.3 discute de l'implémentation des nouveaux services d'inférences développés. Dans la Section 6.4, nous présentons l'application de ONDIL à la conception et à la maintenance des ontologies issues des taxonomies de bcXML.

6.1. Systèmes d'aide à la conception et à la gestion des ontologies

6.1.1. FaCT[1]. Le système FaCT est développé à *University of Manchester* par *Ian Horrocks* [**Hor97 (43)**]. L'objectif de FaCT est d'évaluer la faisabilité de l'utilisation des algorithmes de subsomption optimaux basés sur la technique de tableaux. Le système FaCT permet de raisonner sur des descriptions de concept, de rôles et d'attributs, et de maintenir une hiérarchie de concept basée sur les relations de subsomption. FaCT fournit également les services servant à gérer des B.C terminologiques (TBox), à ajouter l'axiome à une B.C, et à procéder aux inférences.

Par ailleurs, comme les autres classificateurs basés sur L.D, le système FaCT utilise l'algorithme de subsomption pour calculer l'ordre partiel de l'ensemble des noms de concept dans la B.C. Cet ordre partiel s'exprime dans la hiérarchie de concepts. Cette hiérarchie peut être considérée comme un graphe orienté acyclique dans lequel chaque concept est connecté à leurs subsumeurs et subsumés directs. Les algorithmes pour la classification utilisés dans système FaCT bénéficient des propriétés de transitivité par rapport à la relation de subsomption.

Le pouvoir d'inférence du système FaCT résulte des aspects suivants. Premièrement, FaCT se base sur un langage L.D très expressif $\mathcal{SHIQ} = \{\mathcal{ALC}$ + restriction de cardinalité + hiérarchie de rôles + rôle inverse + rôle transitif$\}$ capable de capturer la majorité de la sémantique des modèles industriels, par exemple, le modèle IFC connu dans le secteur de la construction. Deuxièmement, ce système utilise plusieurs techniques d'optimisation récentes [**Hor97 (43)**] mais les plus importantes sont :

[1]Fast Classification of Terminologies

- Normalisation et codage. Cette technique se base sur la syntaxe de descriptions de concept pour détecter toutes les insatisfiabilités évidentes et les structures répétitives dans la B.C. De plus, la technique de codage permet d'investiguer la possibilité dans laquelle une structure de données efficace peut être utilisée pour stocker les concepts.
- Embranchement sémantique. Les algorithmes de tableaux standard sont inefficaces à cause de la technique d'embranchement syntaxique. Par exemple, lorsque l'algorithme standard rencontre une disjonction sur son parcours de l'arbre de recherche, il va rechercher tous les systèmes de contraintes possibles obtenus de l'ajout des contraintes correspondantes à chaque conjonct. Par contre, la technique d'embranchement sémantique permet de diminuer l'espace de recherche dans le cas où deux branches du point d'embranchement dans l'arbre de recherche sont strictement disjointes.
- Rétro-parcours de dépendance (backjumping). Sur le parcours de l'arbre de recherche, cette technique étiquète les contraintes indiquant les affectations qui mène à l'introduction de ces contraintes. Lorsqu'un clash est découvert, l'algorithme peut retourner en ignorant les affectations inutiles. Cela permet d'éviter d'explorer les autres branches de l'arbre de recherche.
- Technique de cache. La détection d'une insatisfiabilité est moins coûteuse que celle d'une satisfiabilité dans les algorithmes de tableaux standard. Pour cette raison, la technique de cache peut faciliter à démontrer la satisfiabilité d'une description de concept constitués des parties dont les résultats de test de satisfiabilité sont sauvegardés dans les étapes précédentes.

Sur le plan d'implémentation, le système FaCT fournit deux interfaces de programmation différentes. En effet, la première interface, appelée CORBA-FaCT, utilise l'architecture CORBA pour la communication client-serveur. Avec cette interface, un client peut communiquer avec le serveur comme un moteur d'inférence via un API Java. Le serveur FaCT offre les groupes de services suivants :

- Gestion de base de connaissances (par exemple, chargement d'un fichier TBox)
- Définition d'un TBox. Ce groupe comporte les fonctions servant à définir un concept, un attribut ou un rôle et à imposer les relations disjointe et d'implication entre des concepts.
- Inférence sur le TBox. Ce groupe comporte les fonctions servant à procéder aux inférences sur le TBox défini. Les inférences importantes sont comme suit : subsomption, satisfiabilité, classification, *etc.*
- Requêtes de TBox.

La deuxième interface de FaCT, appelée DIG[2], est conçu comme un Servlet chez serveur. La communication client-serveur est assurée par le protocole HTTP. Cette interface suit la tendance dans laquelle les applications utilisent une architecture basée sur services. Un client peut communiquer avec le serveurs FaCT via les messages en XML. Cette idée est empruntée des initiatives comme SOAP, XML-RPC qui construit

[2]Description Logic Implementation Group

la spécification du protocole d'échange de messages en utilisant le langage XML audessus le le protocole HTTP. La communication basée sur le protocole HTTP permet au client de réutiliser les outils riches existants pour le développement d'applications.

En effet, dans DIG le client communique avec le serveur via la requête HTTP POST. Le corps d'une requête doit être un message encodé en XML. Le serveur DIG utilise l'élément de racine d'un message pour déterminer le type du message. Les requêtes Ask et Tell sont envoyées par le client et le requête Reponse est retournée par serveur.

6.1.2. RACER[3]. Le système RACER est développé à *University of Hamburg* (2001). Comme FaCT, le moteur d'inférence de RACER utilise les algorithmes de tableaux optimisés. Le système RACER autorise le langage L.D très expressif $\mathcal{SHIQ}$. En dehors des services offerts par FaCT, le système RACER supporte le ABox, les requêtes sur le ABox et peut gérer plusieurs TBox en même temps. De plus, le système RACER autorise à modifier un TBox soumis. C'est-à-dire que l'utilisateur n'a plus besoin de retransmettre un TBox modifié au serveur RACER.

Sur le plan de la représentation de connaissances, le système RACER autorise le TBox qui comporte l' inclusion de concept générale (implies C_1 C_2), l'équation de concept (equivalent C_1 C_2) et l'axiome de disjointness de concept (disjoint C_1,..., C_n). En outre, il supporte également les inférences sur les domaines concrets suivants : $\mathbb{Z}$, $\mathbb{Q}$ en combinant les inéquations linéaires. Plus précisément, les noms des valeurs d'un domaine concret s'appellent *objets*. Les individus sont associés aux objets via les *attributs*. Un attribut peut être déclaré dans une entrée d'un TBox comme suit [**HM02 (40)**] :

```
(signature
    :atomic-concepts (... teenage)
    :roles (...)
    :attributes ((integer age)))
   ...
(equivalent teenager (and human (min age 16)))
(equivalent old-teenager (and human (min age 18)))
   ...
(equivalent human-with-feaver (and human (>= temperature-celsius 38.5)))
```

Sur le plan de l'implémentation, le système RACER offre du côté client plusieurs interfaces. L'interface de fichier permet d'intérroger le serveur via un shell. Il accepte également la syntaxe XML pour les TBox du système FaCT. En particulier, le système RACER peut communiquer avec les applications via l'interface DIG qui est présentée dans le système FaCT. Finalement, le système RACER peut lire les fichier RDF ou DAML+OIL. Les informations (les triplets) dans un fichier RDF sont représentées dans un ABox de telle sorte que le sujet, la propriété et l'objet d'un triplet soient représentés respectivement par un individu, un rôle et un individu.

[3]Renamed ABox and Concept Expression Reasoner

6.2. Représentation de connaissances dans ONDIL

Le premier objectif du système ONDIL est de permettre d'échanger entre des acteurs en tant que des clients des documents contenant les concepts définis dans différentes ontologies. Cette fonctionnalité de ONDIL a pour but de réaliser les scénarios d'échange décrits dans le Chapitre 5. En dehors de plusieurs nouveaux services d'inférences ajoutés, la capacité de communication entre les clients de cette manière fait la différence du système ONDIL des autres systèmes existants.

La base de connaissances dans ONDIL est le triplet : $\Delta = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$. Le composant $\Delta_{\mathcal{T}}$ est le TBox (base terminologique) basé sur le langage L.D $\mathcal{FLE}$. Les inclusions générales ne sont pas acceptées dans le TBox. Le composant$\Delta_{\mathcal{A}}$ est le ABox (assertions sur les individus). Le troisième composant $\Delta_{\mathcal{R}}$ est l'ensemble de règles de révision.
Le triplet Δ, appelé *formalisme hybride*, est construit dans le cadre de ce travail.

6.2.1. Base terminologique. La base terminologique $\Delta_{\mathcal{T}}$ dans ONDIL comporte des définitions de concept. Une entrée de la base qui correspond à une définition se compose de deux parties. Le côté gauche d'une entrée est un nom de concept défini appartenant à un ensemble N_C. Le côté droit d'une entrée est une $\mathcal{FLE}$-description de concept composée de noms de concepts (N_C), de rôles primitifs (N_R) et les constructeurs : conjonction, restriction universelle, restriction existentielle. La base terminologique $\Delta_{\mathcal{T}}$ doit être acyclique dans ONDIL. Un TBox est appelé *acyclique* s'il n'existe pas un nom de concept qui est *directement* ou *indirectement* défini via lui-même.

Dans ce chapitre, on utilise la notation comme présentée dans FaCT pour exprimer la $\mathcal{FLE}$-description de concept. La table suivante fournit la correspondance entre cette notation et celle utilisée dans les chapitres précédents.

Notation logique	Notation de FaCT
$\top$	`*TOP*`
$C_1 \sqcap ... \sqcap C_n$	(`and` $C_1 ... C_n$)
$\forall R.C$	(`some` R C)
$\exists R.C$	(`all` R C)

Exemple 6.2.1. Dans le TBox de ONDIL, le produit résistant aux polluants est défini comme suit :

ProdRésAuPolluant :=(`and` **Produit** (`some` *résister* **Feu**) (`some` *résister* **Bruit**))

Cela signifie que **ProdRésAuPolluant** est un **Produit** qui peut résister au **Feu** et au **Bruit**.

6.2.2. Base d'assertions (ABox). La base d'assertion $\Delta_{\mathcal{A}}$ ou ABox dans ONDIL contient des assertions sur les individus. L'ensemble des noms d'individus satisfait l'*hypothèse du nom unique* c'est -à-dire que tous les noms utilisés dans ABox doivent se référer aux objets (constantes) distingués dans le domaine. Une assertion dans ONDIL est sous forme $C(d)$ ou $R(d_1, d_2)$ où d, d_1, d_2 sont les noms d'individus , C est une $\mathcal{FLE}$-description de concept et R appartient à l'ensemble de rôles primitifs N_R.

Dans ONDIL, chaque ABox se réfère à un TBox. Les assertions dans le ABox doivent être interprétées selon les définitions dans le TBox référé. Lorsqu' un ABox est créé, le TBox référé doit exister.

Exemple 6.2.2. Dans le ABox de ONDIL, le produit d est un **ProdRésPolluant** est défini comme suit :

$$\textbf{ProdRésPolluant}(d)$$

ou la relation qui dit que le produit d vendu par l'acteur a est défini par :

$$\textit{vendu-par}(d,a)$$

6.2.3. Base de règles de révision. Afin de présenter le troisième composant, nous introduisons la notion *hiérarchie de TBox* ou *hiérarchie d'ontologies* dans ONDIL. Effectivement, les TBox dans ONDIL sont organisées selon une hiérarchie dont la racine est un TBox de base partagée. Chaque nœud de cette hiérarchie correspond à un TBox qui dérive du TBox de base. C'est-à-dire que le TBox de base définit les concepts, les rôles primitifs et les concepts de base dans un domaine d'application. De plus, un concept dans un TBox dérivé peut être défini via les concepts du TBox de base (y inclus les noms de concept N_C et les rôles primitifs N_R). Chaque TBox dérivé représente les connaissances spécifiques. Ainsi, un TBox dérivé est définie indépendemment des autres TBox. Par conséquent, il est possible qu'il existe deux concepts de deux TBox dérivés qui portent différents noms mais qui sont sémantiquement équivalents. En revanche, il est possible également qu'il existe deux concepts de deux TBox dérivés qui portent le même nom mais qui ne sont pas équivalents.

Pour partager la compréhension des concepts qui sont définis dans deux TBox dérivés, nous avons besoin de la révision de ces TBox. Supposons que les assertions dans les ABox dérivés et le ABox de base (ces ABox correspondent respectivement aux TBox dérivés et au TBox de base) décident le déclenchement de la révision. C'est-à-dire que le partage de la compréhension n'a lieu que dans un *contexte* déterminé par les *valeurs de contexte*. Ces valeurs sont impliquées *directement* ou *indirectement* des assertions de ABox. A partir de cela, un *contexte d'échange* ou *contexte* est défini par les assertions explicites ou implicites dans les ABox en question. Ainsi, le processus de révision de deux TBox dérivés est déclenché par les assertions explicites ou implicites dans les ABox en question. Lorsque le processus de révision de deux TBox dérivés se termine, on obtient les TBox révisés qui sont *moins* différents.

La base de règles $\Delta_{\mathcal{R}}$ introduite dans ONDIL a pour objectif de représenter les dépendances entre les assertions de ABox (valeurs de contexte) et les déclenchements d'opérations de révision. Une règle de révision $r \in \Delta_R$ est de la forme :

$$r : Ant(r) \Rightarrow Rev(Cons(r),\, Exp(r))$$

où $Rev \in \{\text{OUBLIER, DIRE}\}$, $Ant(r)$ est la conjonction des assertions dans le ABox, $Cons(r)$ est un nom de concept défini dans le Tbox et $Exp(r)$ est une $\mathcal{FLE}$-description de concept sans conjonction au top-level (concept simple).

De plus, la base de règle $\Delta_{\mathcal{R}}$ dans ONDIL doit satisfaire quelques conditions supplémentaires pour que le processus de révision effectué par l'application des règles dans $\Delta_{\mathcal{R}}$ se termine et les ontologies révisées se soient les plus proches possibles. Effectivement, pour assurer la terminaison du processus de révision, chaque règle $r \in \Delta_R$ doit être acyclique (chaque$C_i \in Ant(r)$ ne dépend pas de $Cons(r)$) et les descriptions de concept $Exp(r_i)$, $Exp(r_j)$ ne se sont pas subsumées réciproquement où $Cons(r_i) = Cons(r_j)$. La différence minimale entre les ontologies révisées sont garantie par la non-récursivité de l'ensemble Δ_R. (l'ensemble Δ_R est non-récursif s'il n'existe pas de cycle de règles dépendantes 4.5.3).

Exemple 6.2.3. Supposons qu'il y ait deux acteurs qui possèdent les deux TBox dérivés suivants :
$\Delta_{\mathcal{T}_1} := \{$
ProdRésAuPolluant := **Produit** $\sqcap \exists$*résister*.**Bruit** ,
$\}$
et
$\Delta_{\mathcal{T}_2}$ (construction) := $\{$
ProdRésAuPolluant := **Produit** $\sqcap \exists$*résister*.**Feu** $\sqcap \exists$*résister*.**Bruit** ,
$\}$
Il y a deux versions différentes du concept défini **ProdRésAuPolluant**. Pour un acteur dans le secteur de la construction par exemple, ce concept est défini comme un concept capable de résister à la fois au **Feu** et au **Bruit**. Par contre, pour un acteur dans les autres secteurs, le concept **ProdRésAuPolluant** est défini comme un concept qui ne peut que résister au **Bruit**.
Supposons que les deux acteurs partagent le même ensemble des assertions $\Delta_{\mathcal{A}}$ comme suit :
$\Delta_{\mathcal{A}}$:=$\{$*vendu-par*(a,b), *acheté-par*(a,c), **EntrepriseEurop**(b), **EntrepriseAmér**(c), **Polluant**(*Bruit*) $\}$.
Notons que le concept **Bruit** a uniquement un individu.
L'assertion *vendu-par*(a,b) signifie que le produit a est vendu par l'entreprise b. L'assertion *acheté-par*(a,c) signifie que le produit a est acheté par l'entreprise c.
Soit $\Delta_{\mathcal{R}}$ contenant la règle de révision suivante :

$\Delta_{\mathcal{R}} := \{$

r_1 : *vendu-par*(X, Y_1) $\wedge$ *acheté-par*(X, Z_1) $\wedge$
associer(Y_1, Y_2) $\wedge$ *associer*(Z_1, Z_2) $\wedge$
Européen(Y_2) $\wedge$ **Américain**(Z_2)
$\Rightarrow$ DIRE(**ProdRésPolluant**, $\exists$ *résister*.**Feu**) ;
}

La règle r_1 dit que : si un produit X est vendu par une entreprise Y_1 qui a un associé **Européen**, le produit X est acheté par une entreprise Z_1 qui a un associé **Américain**, alors le concept **ProdRésPolluant** doit être interprété comme un produit capable de résister au ***Feu***. Cette règle permet de traiter le *contexte géographique i.e.* deux acteurs qui viennent des deux continents différents, et donc qui ont deux normes différentes, peuvent partager la définition du concept **ProdRésPolluant**.
L'assertion **EntrepriseEurop**(b) et la définition
EntrepriseEurop := $\exists$*associer*.**Européen** impliquent *associer*(b, b') et **Européen**(b').
L'assertion **EntrepriseAmér**(c) et la définition
EntrepriseAmér := $\exists$*associer*.**Américain** impliquent *associer*(c, c') et **Américain**(c') . Donc, la condition de la règle r_1 est vérifiée.

6.3. Services d'inférences dans ONDIL

6.3.1. De FaCT à ONDIL. Comme mentionné dans la section précédente, le système FaCT offre les services d'inférence standard efficaces sur le TBox. Cependant, la construction et la maintenance des ontologies nécessitent de nouveaux services d'inférences qui sont capables de calculer des descriptions de concept à partir d'informations complètes ou partielles. D'autre part, le système FaCT ne supporte pas la gestion de plusieurs TBox correspondant à différents clients connectés. Cette manque nous empêche de développer un système d'échanges de données basé *directement* sur le système FaCT. C'est pourquoi afin d'implémenter un système d'échanges de données entre plusieurs acteurs (Section 5.5), nous devons revêtir le serveur FaCT d'une couche qui permet d'une part de gérer plusieurs TBox en même temps, d'autre part de fournir les services d'inférences nécessaires pour la maintenance des ontologies.

Dans ce contexte, nous développons le système ONDIL qui a pour objectif à la fois de remplir ces lacunes de FaCT et de réutiliser son pouvoir d'inférence.

6.3.2. Le Plus Spécifique Subsumeur Commun (lcs). ONDIL permet non seulement d'ajouter un nouveau concept avec la définition bien déterminée mais aussi d'y ajouter un concept dont la définition est partiellement décrite. Ce besoin résulte de la maintenance d'ontologies scientifiques où un concept à ajouter est caractérisé via d'autres concepts existants. En effet, il n'est pas toujours facile d'arriver à construire une définition formelle d'un nouveau concept à partir de certaines propriétés données. Ainsi, la tâche moins difficile est d'indiquer des relations (*e.g* relation de subsomption) entre le nouveau concept et des concepts définis dans l'ontologie. A partir de cela, on peut obtenir la définition du concept "la plus spécifique" qui satisfait à toutes

les relations à l'aide d'un service d'inférence permettant de calculer la définition (représentation formelle) à partir de ces informations partielles. La définition formelle obtenue de ce service peut être considérée comme une proposition. L'utilisateur peut la modifier de telle sorte que la définition modifiée convienne mieux à la signification du concept dans l'application en question.

ONDIL offre un service, appelé *lcs*, permettant de répondre à ce besoin. Ce service utilise les algorithmes développés dans le Chapitre 3. Comme mentionné, ces algorithmes ne fonctionnent qu'avec les TBox qui ne contiennent pas les inclusions générales et sont basés sur le langage $\mathcal{ALE}$. Récemment, le travail dans [**VTL$^+$01 (68)**] étend l' algorithme de calcul *lcs* au langage $\mathcal{ALEN}$. Notons que le *lcs* de descriptions de concept en langage comportant le constructeur de disjonction est trivialement calculé. Malgré ces extensions, l'expressivité de tels TBox est bien inférieure à celle du TBox dans FaCT. Pour cette raison, le service d'inférence *lcs* implémenté dans ONDIL autorise seulement aux TBox basés sur le langage $\mathcal{ALE}$.

6.3.3. Échange de documents sans contexte : approximation. Si tous les acteurs dans le système multi-ontologies utilisent des langages L.D avec la même expressivité, l'interprétation d'un concept dans un document reçu peut se traduire en dépliage de la définition de ce concept. En effet, un récepteur reçoit un document avec les termes définis en fonction de l'ontologie de l'émetteur. Pour interpréter ces termes, le récepteur devrait déplier les définitions jusqu'où ces définitions ne contiennent que les termes de l'ontologie partagée ou de son ontologie locale. Grâce à la similarité des deux langages utilisés par deux acteurs, les définitions dépliées sont interprétables par le récepteur. Dans certains cas, la taille de définitions dépilées serait très grande. C'est pourquoi le système nécessite un service d'inférences, appelé *réécriture*, pour simplifier de telles définitions. Le fonctionnement de ce service se base sur l'idée suivante : on cherche à remplacer au fur et à mesure des parties de la description de concept par des noms de concept qui sont définis dans l'ontologie. Le résultat obtenu dans [**BKM00 (6)**] montre qu'il existe une substitution minimale pour le langage $\mathcal{ALE}$. C'est-à-dire qu'une description de concept peut être recouverte par un nombre minimal de noms de concepts définis dans l'ontologie. Le calcul d'une telle substitution est un problème NP-complet dans le pire des cas. Dans la pratique, la complexité peut être diminuée par des techniques heuristiques. Ces algorithmes heuristiques sont également présentés dans [**BKM00 (6)**].

Cependant, dans certains cas les concepteurs des ontologies locales doivent chercher un compromis entre la complexité et l'expressivité du langage utilisé dans une ontologie. Si un acteur utilise une ontologie "simple" qui satisfait la majorité de ses applications, il n'aura pas besoins d'un langage avec l'expressivité élevée. Dans ce cas, un langage simple avec des services d'inférences moins complexes sera son choix. En revanche, si un acteur a besoin d'une ontologie expressive pour capturer la sémantique sophistiquée de ses applications, alors un langage expressif est évidemment exigé. C'est pourquoi les acteurs font face à l'échange de documents dans lequel les langages L.D utilisés dans deux ontologies dérivées sont différents. Pour interpréter les termes d'un document, le récepteur devrait déplier, comme dans le cas précédent,

les définitions jusqu'où ces définitions ne contiennent que les noms de termes définis dans l'ontologie partagée ou dans son ontologie locale. Malheureusement, il est probable que la définition du terme déplié contient encore des constructeurs qui ne sont pas autorisés dans le langage de l'ontologie du récepteur.

Ce problème est connu comme la première instance du problème de transparence sémantique qui est formulé dans le Chapitre 2 lorsque le langage de l'expéditeur $\mathcal{L}_1$ est plus expressif que celui du récepteur $\mathcal{L}_2$. Les résultats présentés dans le Chapitre 3 permettent à un récepteur d'interpréter un concept d'un expéditeur où $\mathcal{L}_1 = \mathcal{ALC}$ et $\mathcal{L}_2 = \mathcal{ALE}$. L'idée de la solution est issue du calcul d'approximation d'une $\mathcal{ALC}$-description de concept par une $\mathcal{ALE}$-description de concept. Le calcul d'approximation est interprété dans le sens où on cherche la description de concept la plus proche de l'originale et cette description ne contient que les constructeurs du langage L.D du récepteur. Il faut noter qu'une telle transformation fait perdre de la sémantique *i.e* il y a une différence entre la définition originale et celle obtenue.

Le service du calcul d'approximation qui est implémenté dans ONDIL utilise l'algorithme d'approximation simplifié *3.4.2.* Ce service est intégré dans un autre service où deux acteurs connectés au serveur effectuent l'échange d'un document. Plus précisément, supposons que l'ontologie récepteur $O_{\mathcal{ALE}}$ et l'ontologie expéditeur $O_{\mathcal{ALC}}$ utilisent les langages $\mathcal{ALE}$ et $\mathcal{ALC}$ respectivement, alors chez récepteur la fonction suivante est insérée dans la procédure d'interprétation de document reçu :

$$approx_{\mathcal{ALE}}(C)$$

où C est une description de concept reçu de l'expéditeur.

Les deux services d'inférence abordés dans cette section montre que le système est capable d'interpréter un concept reçu de façon déterministe *i.e* il permet de calculer (s'il n'existe pas) ou d'identifier (s'il existe) dans l'ontologie du récepteur la description de concept qui est équivalente ou la plus proche du concept reçu. En particulier, le système peut découvrir si deux concepts différents (ils portent deux noms différents) définis dans deux ontologies dérivées sont équivalents. En d'autres termes, les calculs nécessaires pour cette interprétation se basent uniquement sur les TBox *i.e* sur les définitions de concept. Par conséquent, ce scénario d'échange ne pose pas de problème d'inconsistance car il peut se traduire en problème de l'expansion d'un TBox. Cependant, le scénario d'échange où un concept défini différemment dans deux ontologies dérivées (deux concepts portent le même nom) nécessite certaines tâches supplémentaires. En effet, la première tâche est que la définition du concept dans l'ontologie récepteur doit être modifiée pour assurer la compatibilité entre les deux définitions du concept. La deuxième est de réviser l'ontologie récepteur pour prendre en compte la modification de la définition. La section suivante va présenter comment ce scénario d'échange est intégré dans ONDIL.

6.3.4. Échange de documents avec contexte : Révision et Règles de Révision. Comme mentionné dans le Chapitre 5, les ontologies spécifiques aux sous-domaines peuvent dériver de l'ontologie de base. Une fois le processus d'échange est défini avec la disponibilité des informations contextuelles, les termes de l'ontologie de base sont projetés dans les ontologies spécifiques selon ces informations contextuelles.

Ainsi, les vocabulaires appropriés à ce contexte sont créés. L'approche proposée dans ONDIL inspirée de l'ebXML utilise les règles de contexte comme un formalisme supplémentaire pour représenter les informations contextuelles elles-mêmes et pour appliquer ces informations à une ontologie dérivée existante. L'antécédent de la règle de contexte est la conjonction de valeurs de contexte. Ces valeurs sont déterminées lorsque le processus d'échange est défini. Le conséquent de la règle de contexte est une action permettant de modifier la définition d'un terme.

Nous nous rendons compte que ce modèle d'échange correspond à la deuxième instance du problème de transparence sémantique formulée dans le Chapitre 2. En effet, l'application des règles de contexte permet aux acteurs de partager la compréhension d'un concept qui est différemment défini dans des ontologies dérivées. L'action de modification d'un concept correspondant au conséquent d'une règle de contexte peut amener à une révision de l'ontologie. De plus, il existe un ensemble de règles de contexte qui s'appliquent à un concept. Cela nécessite un ordre d'application de règle de telle sorte que l'application des règles dans cet ordre fournit chez récepteur l'ontologie révisée qui est "la plus proche" de l'ontologie expéditeur. Cela implique que la compréhension des concepts est partagée au mieux.

L'ONDIL propose les services de révision basées sur les algorithmes développés dans le Chapitre 4. D'une part, ces services sont implémentés indépendamment et ils peuvent être appelés depuis chaque ontologie dérivée. Ils servent à réviser l'ontologie dans la phase de maintenance. Plus précisément, les deux services de révision suivants sont intégrés dans ONDIL :

$$\text{OUBLIER}(C, Exp) \text{ et } \text{DIRE}(C, Exp)$$

où C est un nom de concept défini dans un $\mathcal{FLE}$-TBox et Exp est une $\mathcal{FLE}$-description de concept sans conjonction au top-level.

D'autre part, pour modéliser les règles de contexte, on a besoin d'insérer dans la base de connaissance le composant de règles de révision $\Delta_{\mathcal{R}}$ (Section 4.4). Notons que le composant de règles $\Delta_{\mathcal{R}}$ devrait être global *i.e* les ontologies dérivées partagent ce composant de règles.

L'ONDIL propose également un service d'inférence spécifique pour le formalisme hybride basé sur l'Algorithme 4.5.12 développé dans le Chapitre 4. Ce service permet d'identifier, dans un contexte donné, l'ensemble maximal de règles et un ordre sur cet ensemble tels que l'application de cet ensemble de règles dans l'ordre fournit l'ontologie "la plus appropriée" pour échanger des données dans ce contexte. Les informations contextuelles sont exprimées dans le ABox $\Delta_{\mathcal{A}}$ au moment de l'échange de données.

6.4. Modules développés dans ONDIL

Le système ONDIL comporte plusieurs modules indépendants. Nous allons présenter en détail les modules développés dans le cadre de cette thèse. ONDIL fournit une interface graphique basée sur WEB et la communication entre le serveur et le

client est assurée par le protocole HTTP. Le serveur est revêtu par une couche Servlet qui reçoit les requêtes des clients et appelle les services du moteur d'inférence de ONDIL ou de FaCT.

6.4.1. Gestion des Ontologies et Moteur d'Inférence. Le serveur ONDIL peut gérer différentes ontologies et chaque ontologie correspond à un client connecté au serveur ONDIL. Le serveur offre des services d'inférences permettant la transformation d'une ontologie en une autre afin d'assurer les échanges de données (documents) entre les clients. Le moteur d'inférence FaCT est intégré dans ONDIL afin de profiter des services d'inférences standard implémentés autorisant les langages L.D très expressifs. Le moteur d'inférence développé par ONDIL implémente certains services non-standard étudiés dans les Chapitres 3, 4.

6.4.2. Aide à la conception des ontologies. Au niveau de l'aide de la conception des ontologies, ONDIL implémente les services suivants :

- Le serveur ONDIL peut lire un fichier représentant un TBox en XML-FaCT. Le fichier lu est compilé en une ontologie.
- A partir de l'interface graphique de Argo[4], un concepteur peut convertir ses diagrammes de classe UML en un fichier XMI (format XML défini par Argot pour les diagrammes UML). En suite, le fichier XMI obtenu est compilé et géré comme une ontologie. Ce service est développé en collaboration avec l'équipe KRYSTAL-EXECO. Puisque le langage L.D supporté par ONDIL est moins expressif que le langage $\mathcal{DLR}$, les ontologies issues des diagrammes UML complexes peuvent bénéficier seulement du service de satisfiabilité offert par le moteur d'inférence FaCT intégré. Maintenant, nous pouvons utiliser le service de satisfiabilité du FaCT pour vérifier la cohérence du schéma UML. La Figure 6.4.1 montre un schéma UML conçu dans Argo.
  La Figure 6.4.2 visualise l'ontologie obtenue du schéma UML dans la Figure 6.4.1.
- En se basant sur les classes graphiques de FaCT, nous implémentons un service qui permet de visualiser graphiquement la structure hiérarchique d'une ontologie (cf. la Figure 6.4.2).

6.4.3. Aide à la maintenance des ontologies. Au niveau de l'aide à la maintenance des ontologies, ONDIL implémente les services suivants :

- En se basant sur le nouveau algorithme pour le calcul lcs, nous implémentons un service qui permet de construire un concept à partir de descriptions partielles (cf. la Figure 6.4.3).
- Les opérations de révision : DIRE et OUBLIER dans le langage $\mathcal{FLE}$

[4]http ://argotuml.tigris.org

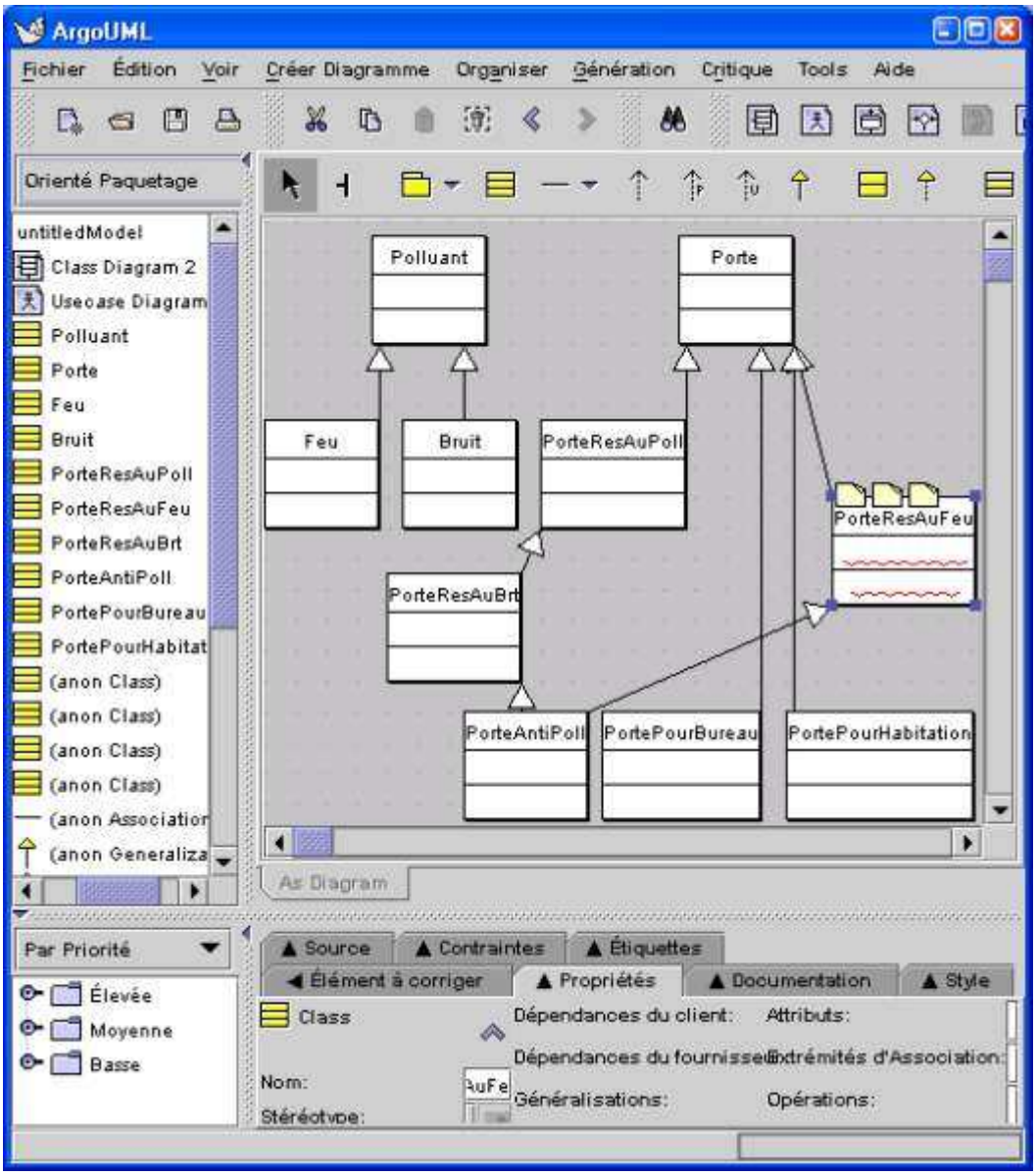

FIG. 6.4.1. Création d'une ontologie à l'aide d'Argo

6.4.4. Échange de documents. Dans le modèle ebXML, un document commercial est créé à partir des deux sources d'informations. Le Processus Commercial définit l'échange de données *i.e* les modèles de documents à échanger. Deuxièmement, la définition des termes constituant un document est déterminée par les Composants de Base. Et pourtant, le premier scénario d'échange de document qui est implémenté dans ONDIL suppose que le modèle de documents existe auparavant et le document se compose des concepts définis dans l'ontologie d'un émetteur. Afin d'interpréter le

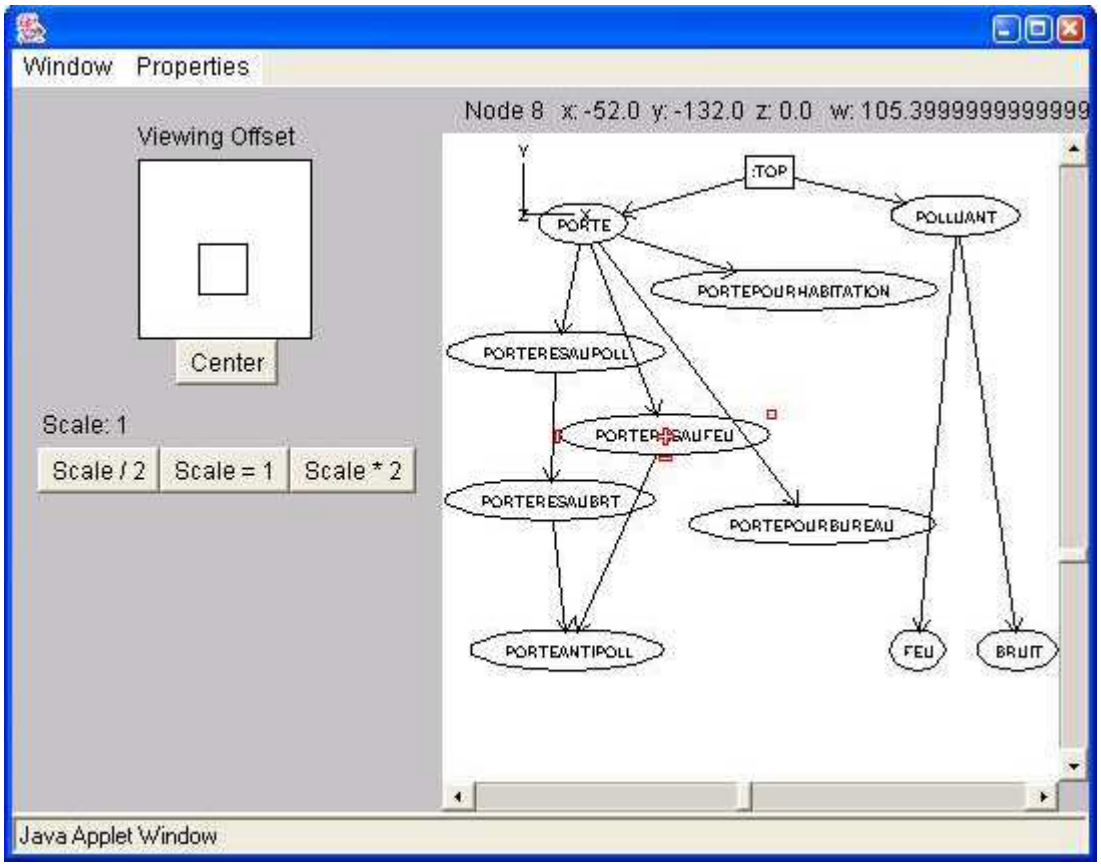

FIG. 6.4.2. Ontologie correspondant au schéma UML

document reçu, un récepteur utilise le service standard d'équivalence ou le service non-standard d'approximation.

Comme mentionné, les services non-standard, y inclus les opérations de révision, ne sont développés que dans les langages L.D sans constructeur de disjonction. C'est pourquoi les services supportant l'échange de documents implémentés dans ONDIL permettent de calculer le concept défini dans une ontologie sans constructeur de disjonction tel que ce concept soit le plus proche d'un concept défini dans une ontologie autorisant le constructeur de disjonction. Plus précisément, on considère les deux cas suivants :

(1) Les ontologies de l'émetteur et du récepteur sont définies respectivement dans le langage $\mathcal{FLE}^+$ et le $\mathcal{FLE}$ où $\mathcal{FLE}^+$ est obtenu du $\mathcal{FLE}$ en autorisant le constructeur de disjonction.

(2) Les ontologies de l'émetteur et du récepteur sont définies respectivement dans le langage $\mathcal{ELN}^+$ et $\mathcal{ELN}^-$ où $\mathcal{ELN}^+$ est obtenu du $\mathcal{ELN}^-$ en autorisant le constructeur de disjonction.

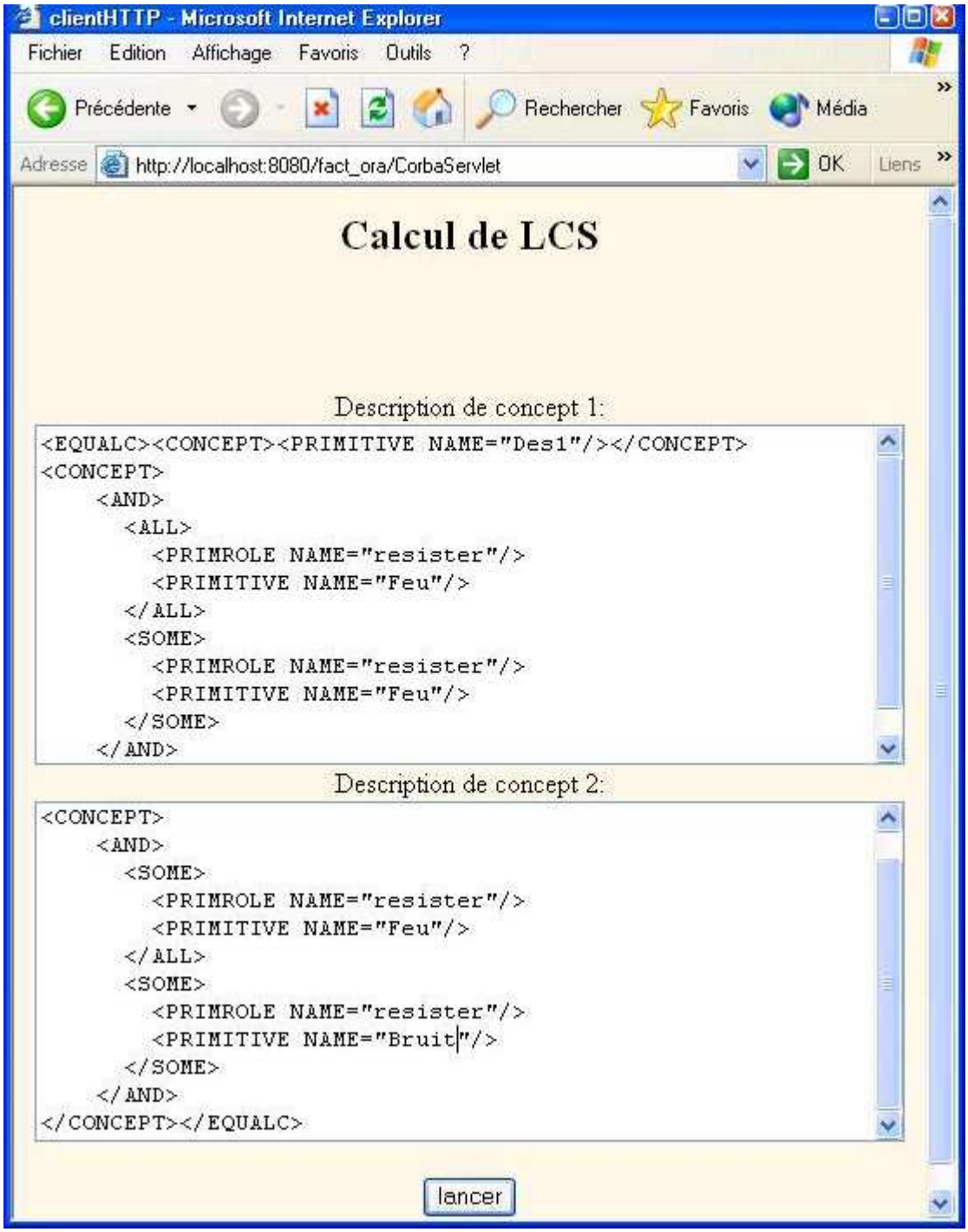

FIG. 6.4.3. Calcul du LCS de deux $\mathcal{FLE}$-descriptions de concept

En sachant que le langages $\mathcal{FLE}$ est le sous-langage de $\mathcal{ALC}$ et le langage $\mathcal{FLE}$ est le sous-langage de $\mathcal{ALE}$, alors nous avons utilisé les algorithmes développés dans

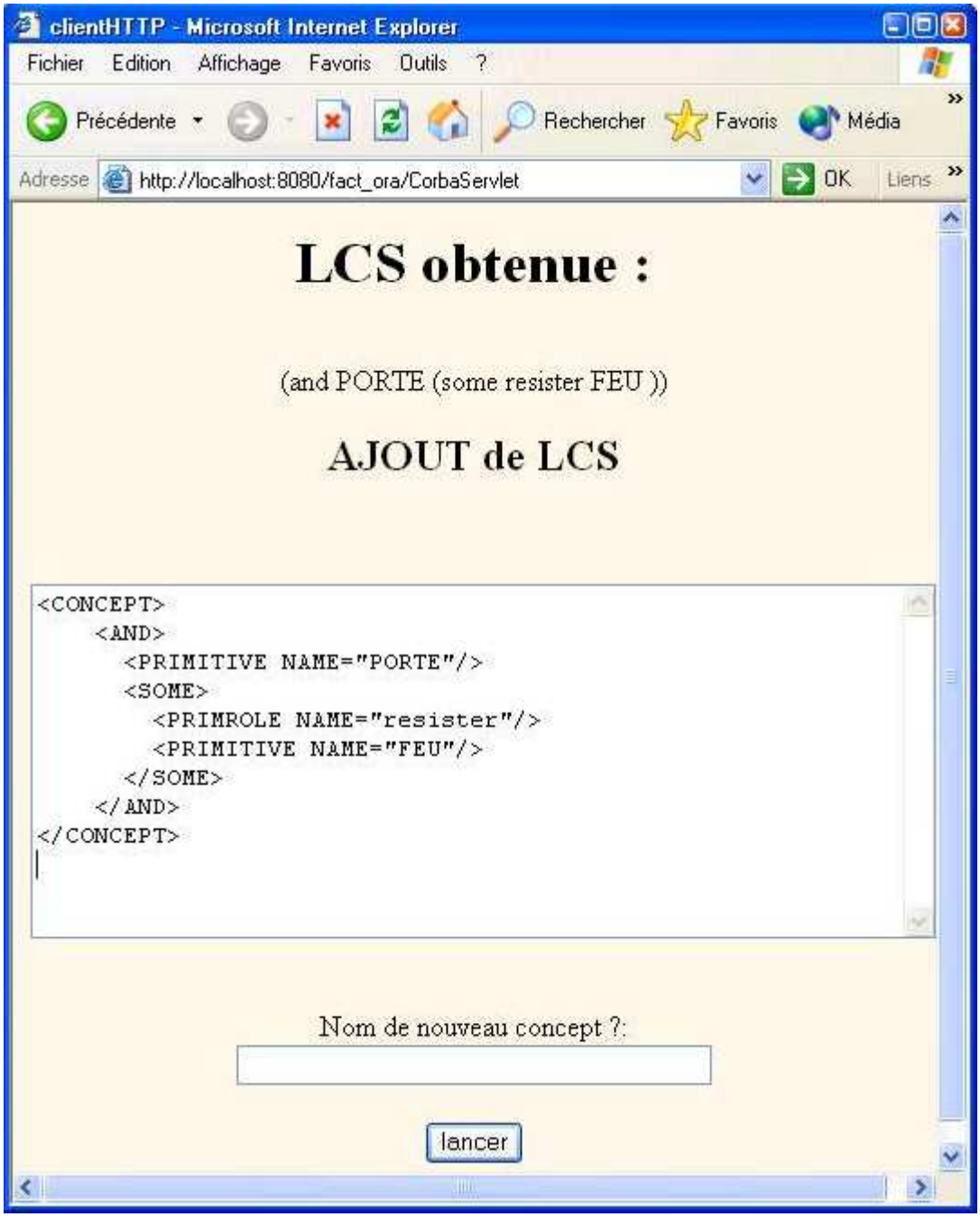

FIG. 6.4.4. Résultat du calcul LCS

le Chapitre 3 pour implémenter le service correspondant au premier cas. En outre, les algorithmes développés dans [**BKT02b (16)**] sont utilisés pour le deuxième cas.

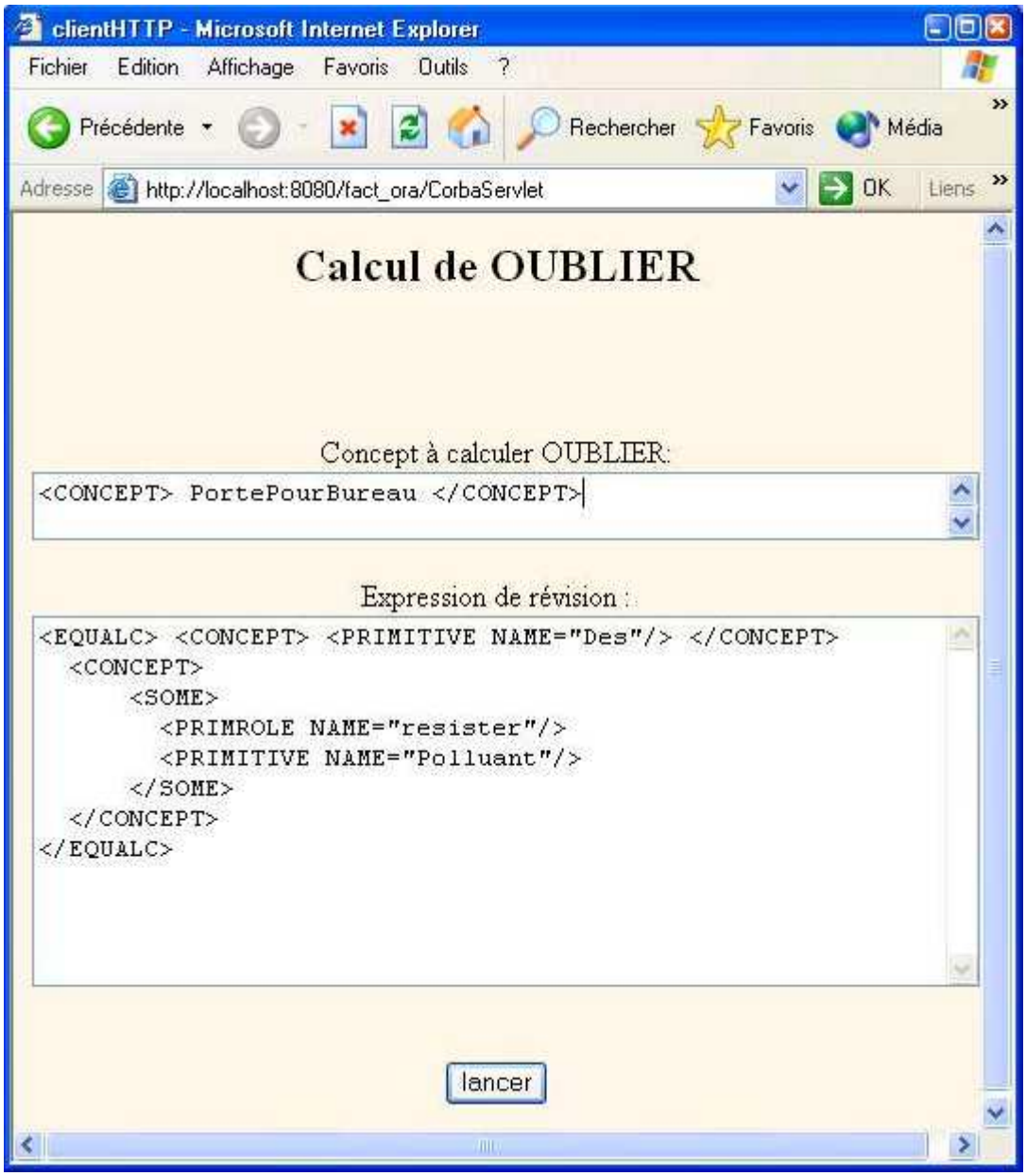

FIG. 6.4.5. Calcul de OUBLIER d'un $\mathcal{FLE}$-concept

Exemple 6.4.1. Soient PorteResAuPolluant, PortePourBureau deux concepts définis dans l'ontologie O_1 basée sur le langage $\mathcal{FLE}^+$ et PorteAntiPolluant un concept défini dans l'ontologie O_2 basée sur le langage $\mathcal{FLE}$. De plus, les deux concepts Feu, Bruit sont définis dans l'ontologie partagée.

```
<EQUALC>
        <CONCEPT>
                <PRIMITIVE NAME="Feu"/>
        </CONCEPT>
        <CONCEPT>
        <AND>
                <PRIMITIVE NAME="Chaleur"/>
                <PRIMITIVE NAME="Polluant"/>
        </AND>
        </CONCEPT>
</EQUALC>


<EQUALC>
        <CONCEPT>
                <PRIMITIVE NAME="Bruit"/>
        </CONCEPT>
        <CONCEPT>
        <AND>
                <TOP/>
                <PRIMITIVE NAME="Polluant"/>
        </AND>
        </CONCEPT>
</EQUALC>
```

Dans l'ontologie O_1, les concepts suivants sont définis :

```
<EQUALC>
        <CONCEPT>
        <PRIMITIVE NAME="PorteResAuPolluant"/>
      </CONCEPT>
      <CONCEPT>
            <AND>
                  <PRIMITIVE NAME="Porte"/>
            <SOME>
                  <PRIMROLE NAME="resister"/> <TOP/>
            </SOME>
            <OR>
            <ALL>
                  <PRIMROLE NAME="resister"/>
                  <PRIMITIVE NAME="Feu"/>
            </ALL>
            <ALL>
                  <PRIMROLE NAME="resister"/>
                  <PRIMITIVE NAME="Bruit"/>
            </ALL>
            </OR>
            </AND>
      </CONCEPT>
</EQUALC>

<EQUALC>
        <CONCEPT>
        <PRIMITIVE NAME="PortePourBureau"/>
        </CONCEPT>
        <CONCEPT>
              <AND>
              <PRIMITIVE NAME="Porte"/>
              <SOME>
                    <PRIMROLE NAME="resister"/>
                    <TOP/>
              </SOME>
              <ALL>
                    <PRIMROLE NAME="resister"/>
                    <PRIMITIVE NAME="Feu"/>
              </ALL>
              </AND>
        </CONCEPT>
</EQUALC>
```

Dans l'ontologie O_2, le concept suivant est défini :

```
<EQUALC>
        <CONCEPT>
        <PRIMITIVE NAME="PorteAntiPolluant"/>
        </CONCEPT>
        <CONCEPT>
                <AND>
                        <PRIMITIVE NAME="Porte"/>
                <SOME>
                        <PRIMROLE NAME="resister"/>
                        <PRIMITIVE NAME="Feu"/>
                </SOME>
                <ALL>
                        <PRIMROLE NAME="resister"/>
                        <PRIMITIVE NAME="Feu"/>
                </ALL>
                </AND>
        </CONCEPT>
</EQUALC>
```

La Figure 6.4.6 illustre la composition d'un document en utilisant l'ontologie O_1. Les termes PorteResAuPolluant, PortePourBureau sont choisis de l'ontologie.

Nous obtenons les résultats qui sont illustrés dans la Figure 6.4.7. Le concept PorteResAuPolluant doit être approximé par une $\mathcal{FLE}$-description de concept. Par contre, le système détermine que le concept PortePourBureau est équivalent au concept PorteAntiPolluant qui est existant dans l'ontologie O_2.

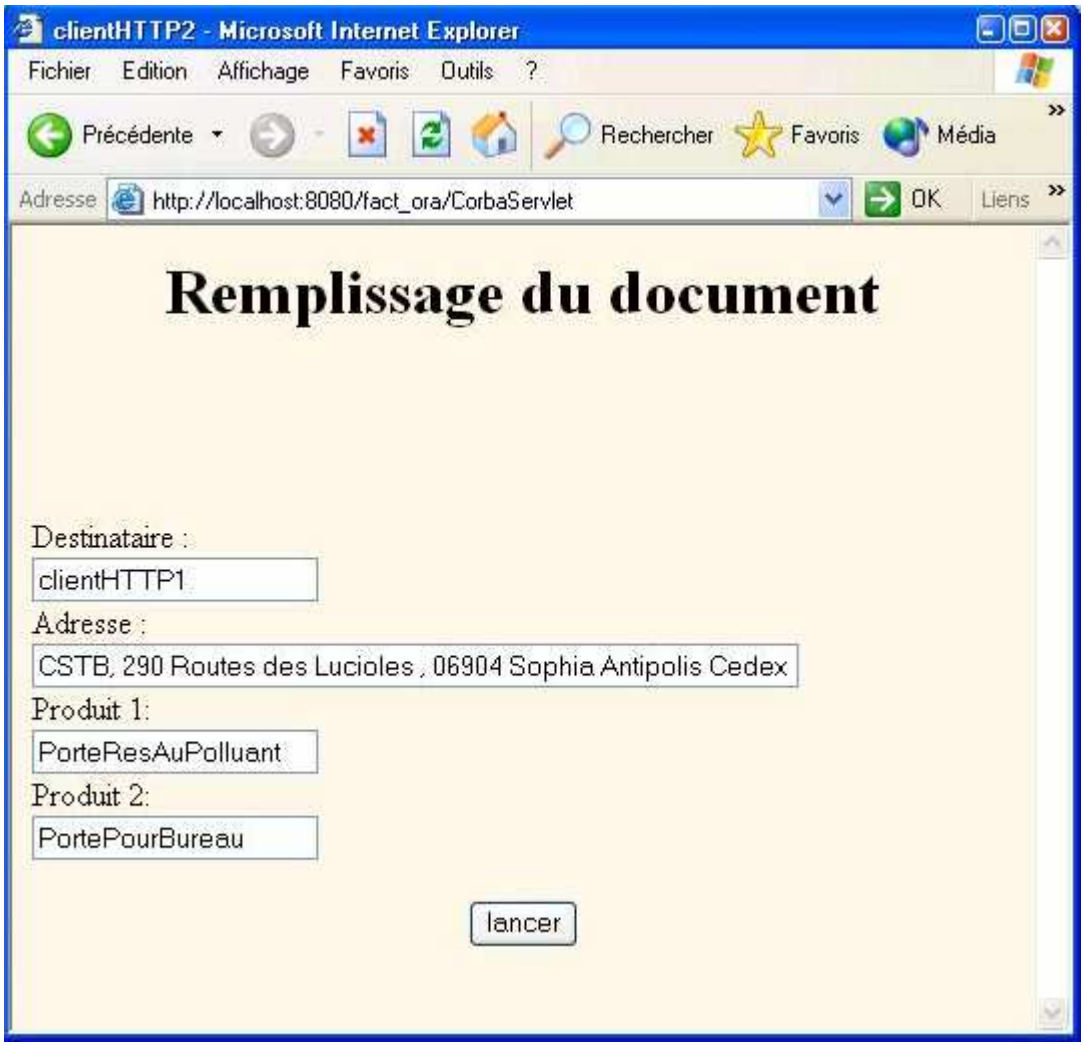

FIG. 6.4.6. Composition d'un document dans $\mathcal{FLE}^+$-ontologie

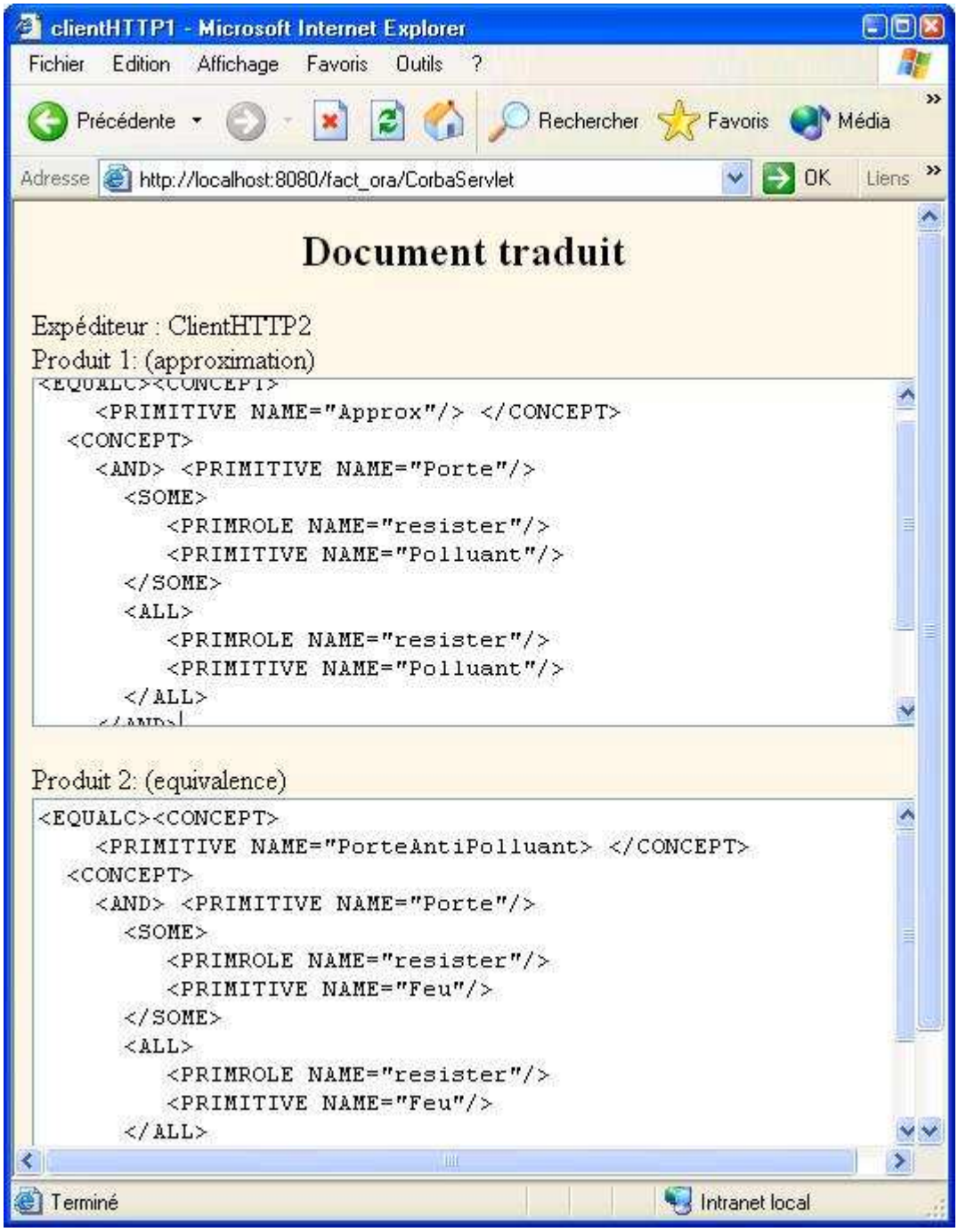

FIG. 6.4.7. Interprétation d'un document dans $\mathcal{FLE}$-ontologie

Conclusion

Un des objectifs de cette thèse a été d'identifier un formalisme approprié permettant de représenter les connaissances commerciales impliquées dans les modèles de données existants dans le Commerce Électronique. Ce formalisme doit être suffisamment expressif et flexible pour qu'il soit capable de modéliser les connaissances très hétérogènes du domaine du Commerce Électronique et d'y intégrer sans difficulté majeure d'autres formalismes compatibles. D'autre part, ce formalisme doit être muni d'une sémantique formelle et déclarative pour que le problème de l'automatisation de certaines tâches résultant du domaine d'application puisse se traduire en problème d'inférences dans ce formalisme. En d'autres termes, tant que la sémantique informelle est impliquée derrière le formalisme utilisé, la cohérence des connaissances n'est pas assurée. A travers les chapitres, nous avons montré dans quelle mesure la Logique de Description satisfait ces critères.

En analysant les modèles d'échange de données existants dans le domaine du Commerce Électronique, nous avons mis en évidence une variante du problème de transparence sémantique qui présente une difficulté majeure des modèles d'échange de données actuels. Le problème de transparence sémantique empêche les applications de communiquer de façon "intelligente", c'est-à-dire la signification des données échangées sur laquelle les applications doivent s'accorder auparavant. Aucune connaissance implicite résultant d'une procédure d'inférence, n'est considérée tant que ce problème est présent.

L'utilisation de la Logique de Description dans le but de formaliser la sémantique impliquée derrière les formalismes utilisés dans le modèle de données de ebXML tente d'enlever la barrière imposée par le problème mentionné. Toutefois, la Logique de Description nécessite certaines extensions pour qu'elle puisse assurer les propriétés attendues.

En effet, la formulation de la sémantique des formalismes présents à l'aide de la Logique de Description nous permet de traduire le problème de transparence sémantique dans le Commerce Électronique en le problème de transformation d'ontologies basées sur cette logique. Nous avons étudié deux instances intéressantes du problème : i) la traduction précise ou approximative d'une ontologie en une autre ontologie, ii) harmonisation des ontologies par les règles de révision. Afin de proposer des solutions bien fondées aux instances du problème, nous avons présenté l'organisation des ontologies du système selon la hiérarchie. La solution à la première instance du problème résulte d'une part des services d'inférence standard bien connus, *e.g* la subsomption entre deux descriptions de concept dans un langage de Logique de Description, d'autre part des services d'inférence non-standard : le Plus Spécifique

Subsumeur Commun dans le langage $\mathcal{ALE}$ et Approximation d'une description de concept dans un langage $\mathcal{ALC}$ par une description de concept dans le langage moins expressif $\mathcal{ALE}$. Nous avons développé un algorithme permettant d'obtenir le Plus Spécifique Subsumeur Commun dont la taille est polynômiale en fonction des tailles de deux $\mathcal{ALE}$-descriptions de concept d'entrée. Par conséquent, la taille de l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$ est également diminuée. Quant à la deuxième instance du problème, d'abord, nous avons étudié les opérations de révision de la base de connaissances. La révision d'une terminologie est indispensable lorsque l'on souhaite partager la compréhension des connaissances. Une projection du canevas AGM sur la révision de la terminologie nous permet de définir les opérations de révision de façon satisfaisante pour les terminologies utilisant le langage $\mathcal{FLE}$. Les opération de révision définies sont capables de modéliser les conséquents des règles de contexte dans ebXML. Le formalisme hybride résultant de la combinaison de la Logique de Description et les règles modélisées fournit un langage assez expressif capable de représenter la sémantique du modèle de données dans ebXML. En se basant sur les algorithmes proposés, nous avons montré que le calcul de ces opérations de révision est décidable. Ensuite, la procédure qui transforme la base de connaissances par l'application de règles de révision, est étudiée. Nous avons identifié une condition dans laquelle la terminaison de cette procédure est assurée. Nous avons également proposé une autre procédure qui permet de déterminer l'ordre d'application de règles selon lequel le nombre de règles appliquées est maximal. Cette procédure est justifiée par le fait que la différence entre les terminologies obtenues est "minimale".

La première remarque concerne la taille des descriptions de concepts obtenues du calcul du Plus Spécifique Subsumeur Commun (lcs) de n $\mathcal{ALE}$-descriptions de concept où $n > 2$. Effectivement, le travail dans [**BT02 (4)**] prouve que dans le cas où $n = 2$, la taille de lcs écrit dans la représentation ordinaire peut être exponentielle en fonction de n. Toutefois, le travail dans le Chapitre 3 de cette thèse montre que dans le cas où $n = 2$, la représentation compacte proposée pour les descriptions de concept permet à la fois de diminuer la taille des lcs et de préserver la complexité du problème de subsomption entre des descriptions de concept transformées en la représentation compacte. La question qui se pose est la suivante : existe-il dans le cas où $n > 2$, une (nouvelle) représentation compacte pour les $\mathcal{ALE}$-descriptions de concept qui possède la même propriété ? Notons que le travail dans [**BKM99 (5)**] a introduit un exemple selon lequel la taille du lcs des n $\mathcal{EL}$-descriptions de concept écrits dans la représentation ordinaire est exponentielle en fonction de n. La réponse positive à cette question impliquera que la taille de toutes les définitions de concept dans la base de connaissance augmente de façon polynômiale malgré les calculs de lcs et de l'approximation $\mathcal{ALC}$-$\mathcal{ALE}$. Cela apportera une contribution significative lorsque davantage d'outils permettent de libérer les concepteurs de traiter directement les descriptions de concepts dans la représentation ordinaire.

En ce qui concerne le problème de la révision de la base de connaissances basée sur la Logique de Description, nous avons défini et calculé les opérations de révision dans les langages $\mathcal{FLE}$. De plus, nous avons commencé à étudier les opérations de révision dans le langage $\mathcal{ELN}$. Des algorithmes pour le calcul des opérations de

révision feront l'objet de travaux ultérieurs. Par ailleurs, selon un de nos travaux préliminaires, l'opération SEMI-OUBLIER dans des langages autorisant le constructeur de disjonction sera unique s'il existe. Cependant, il n'est pas évident de définir la notion de concept simple et de proposer une caractérisation de la subsomption structurelle dans e tels langages. Ainsi, le problème de la décidabilité des opérations de révision dans un tel langage mérite d'être étudié.

Le formalisme hybride combinant la Logique de Description avec les règles de révision qui est introduit dans cette thèse, évoque une autre combinaison. Il s'agit d'intégrer un sous-langage de CARIN dans ce formalisme. Cette intégration non seulement posera des problèmes à résoudre, par exemple l'interaction entre les règles de révision et les règles de Horn ainsi que l'impact des règles de Horn sur les inférences établies, mais aussi apportera de nouvelles applications grâce à la richesse d'expression du nouveau formalisme.

Sur le plan de la perspective applicative, le système ONDIL attend une implémentation capable de gérer les ABox. Un tel module permet de mettre en place les règles de révision et l'inférence sur le formalisme hybride défini dans le système. D'autre part, le problème de stockage des bases de connaissances comportant les TBox et les ABox n'est pas été traité dans le cadre de cette thèse. L'utilisation d'un SGBD permettant de stocker les bases de connaissances sous la forme de schémas XML est envisageable.

Bibliographie

[AGM85 (1)] Alchourrón, C. E, Gärdenfors, P. et Makinson, D. (1985). On the logic of theory change : Partial meet functions for contraction and revision. *Journal of symbolic Logic*, 50.

[Ant00 (2)] Anthony, J.R. (2000). Belief Revision-An overview. *Departement of Computer Science, Keele University, Stanffordshire.*

[Baa96 (3)] Baader, F. (1999). Logic-based Knowledge Representation. *Artificial Intelligence Today, Recent Trends and Developments*, *Lecture Notes in Computer Science, Springer Verlag*, pp.13-41.

[BT02 (4)] Baader, F. et Turhan, A.Y. (2002). On the problem of Computing Small Representations of Least Common Subsumers. *Proceeding of KI2002.* pp. 99-113.

[BKM99 (5)] Baader, F., Küsters, R. et Molitor, R. (1999). Computing least common subsumer in Description Logics with existential restrictions, *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence* (*IJCAI-99*), pp. 96-101.

[BKM00 (6)] Baader, F., Küsters, R. et Molitor, R. (2000). Rewriting Concepts Using Terminologies. *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning* (KR2000). pp. 297-308.

[BMT99b (7)] Baader, F., Molitor, R. et Tobies, S. (1999). On the relation between Description Logics and conceptual graphs. *William Tepfenhart and Walling Cyre editors*, *Pro. of the 7th Int. Con. on Conceptual Structures* (JCCS'99).

[BCM$^+$03 (8)] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (2003). The Description Logic Handbook-Theory, Implementation and Applications, *Cambridge University Press.*

[BCD03 (9)] Berardi, D., Calvanese, D. et De Giacomo, G. (2003). Reasoning on UML Class Diagrams is EXPTIME-hard. *Proc. of Int. Workshop on Description Logics (DL 2003).*

[Ber66 (10)] Berger, R. (1966). The undecidability of the dominoe problem. *Mem. Amer. Math. Soc.*, *66,* pp. 1-72.

[BHL01 (11)] Berners-Lee, T., Hendler, J. et Lassila, O. (2001). The semantic Web. *Scientific American.*

[Bor94 (12)] Borgida, A. (1996). On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence,* 82, pp. 353-367.

[Bra77 (13)] Brachman, R. J. (1977). What's a concept : Structural foundations for semantic networks. *International Journal of Man-Machine Studies.*

[Bra78 (14)] Brachman, R.J. (1978). Structured inheritance networks. *Research in Natural Language Understanding.*

[BKT02a (15)] Brandt, S., Küsters, R. et Turhan, A. Y. (2002). Approximation and Difference in Description Logics. *Proceeding of KR2002,* pp. 203-214.

[BKT02b (16)] Brandt, S., Küsters, R. et Turhan, A.Y. (2002). Approximation ALCN-Description Logics. *Proceeding of DL'02.*

[CCD+02 (17)] Calì , A., Calvanese, D., De Giacomo, G. et Lenzerini, M. (2002). A Formal Framework for Reasoning on UML Class Diagrams. *Proc. of the 13th Int. Sym. on Methodologies for Intelligent Systems (ISMIS 2002).*

[CLN99 (18)] Calvanese, D., Lenzerini, M. et Nardi, D. (1999). Unifying class-based representation formalismes. *Journal of Artificial Intelligence Research*, 11, 199-240.

[CDL01 (19)] Calvanese, D., De Giacomo, G. et Lenzerini, M. (2001). Identification constraints and functional dependencies in description logics. *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001),* pp.155-160.

[CHI01 (20)] Chiaramonti, C. (2001). Echange de données informatisé. *Technique de l'Ingénieur , Traité Informatique.*

[Cla00 (21)] Clark, C.J. (2000). The UN/CEFACT Unified Modeling Methodology - An Overview. *UNCEFACT/TMWG/N097.*

[DHL+92 (22)] Donini, F.M., Hollunder, B., Lenzerini, M., Spaccamela, A.M., Nardi, D. et Nutt, W. (1992). The complexity of existential quantification in concept languages. *Artificial Intelligence*, 2, pp. 309-327.

[DLN+96 (23)] Donini, F.M., Lenzerini, M., Nardi, D. et Schaerf, A. (1996). Reasoning in Description Logics. *Gerhard Brewka, editor, Principles of Knowledge Representation, Studies in Logics, Language and Information. CSLI Publications*, pp. 193-238.

[DLN+98 (24)] Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W. et Schaerf, A. (1998). An Epistemic Operator for Description Logics, *Artificial Intelligence,* 100, pp. 225-274.

[DLN+98b (25)] Donini, F.M., Lenzerini, M., Nardi, D. et Schaerf, A. (1998) AL-log : Integrating Datalog and Description Logics. *Journal of Intelligence Information Systems,* 10, 227-252.

[ebX01a (26)] ebXML (2001). ebXML Core Components, Document Assembly and Context Rules. Voir *http ://www.ebxml.org.*

[ebX01b (27)] ebXML (2001). ebXML Business Process Analysis Worksheets & Guideline. Voir *http ://www.ebxml.org.*

[ebX01c (28)] ebXML (2001). ebXML Business Process and Business Information Analysis. Voir *http ://www.ebxml.org.*

[ebX01d (29)] ebXML (2001). ebXML Business Process Specification Schema. Voir *http ://www.ebxml.org.*

[ebX01e (30)] ebXML (2001). ebXML Core Component Discovery and Analysis. Voir *http ://www.ebxml.org.*

[ebX01f (31)] ebXML (2001). ebXML Core Component Overview. Voir *http ://www.ebxml.org.*

[ebX01g (32)] ebXML (2001). ebXML Core Components, Context and Reutisability of Core Components. Voir *http ://www.ebxml.org.*

[ebX01h (33)] ebXML (2001). ebXML Technical Architecture Specification. Voir *http ://www.ebxml.org.*

[Fen01 (34)] Fensel, D. (2001). Ontologies : A Silver Bullet for Knowledge Management and Electronic Commerce. *Springer.*

[Goh97 (35)] Goh, C.H. (1997). Representing and Reasonning about Semantic Conflicts in Heterogeneous Information Sources. *PhD. Thesis, MIT.*

[Gär92 (36)] Gärdenfors, P. (1992). Belief Revision. *Cambridge University Press.*

[GOR97 (37)] Grädel, E., Otto, M. et Rosen, E. (1997). Two-variable logique with counting is decidable. *Proceedings of the Twelfth Annual IEEE Symposium on Logic in Computer Science (LICS-97),* pp. 306-317.

[Gru93 (38)] Grubert, T. (1993). A Translation approach to portable ontologies. *Knowledge Acquisition, 5.*

[Gru95 (39)] Grubert, T.(1995). Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum. Comput. Stud. 43.*

[HM02 (40)] Haarslev, V., Möller, R. (2002). RACER User's Guide and Reference Manual. *Concordia University, Computer Science Departement.*

[Hay79 (41)] Hayes, P.J. (1979). The logic of frames. *Frame Conception and Text Understanding. Walter de Gruyter and Co..*

[HU79 (42)] Hopcroft, J.E., Ullman, J.D. (1979). Introduction to Automata Theory, Languages and Computation. *Addison Wesley.*

[Hor97 (43)] Horrocks, I. (1997). Optimising Tableaux Decision Procedures for Description Logics. *PhD thesis, University of Manchester.*

[KR99 (44)] Kalakota, R., Robinson, M. (1999). E-business : roadmap for success. *Addison Wesley.*

[Kim98 (45)] Kimbrough, S.O. (1998). Formal Language for Business Communication : Sketch of basic theory. *International Journal of Electronic Commerce.*

[Kim00 (46)] Kimbrough, S.O. (2000). EDI, XML, and the Transparency Problem in Electronic Commerce. *INFORMS, San Antonio.*

[KM97 (47)] Kimbrough, S.O. et Moore, S.O. (1997). On Automated Message Processing in Electronic Commerce and Work Support Systems : Speech Act Theory and Expressive Felicity . *Transaction on Information Systems.*

[Küs01 (48)] Küsters, R. (2001). Non-Standard Inferences in Description Logics. *PhD. Thesis, Springer.*

[KM01 (49)] Küsters, R. et Molitor, R. (2001). Computing least common subsumer for $\mathcal{ALEN}$. *Proceeding of IJCAI01, Morgan Kaufman,* pp. 219-224.

[LL03 (50)] Le Duc, C., Le Thanh, N. (2003). On the problems of representing Least Common Subsumer and Computing Approximation in DLs. *Prod. of Int. Workshop in Description Logics. DL03.*

[LL02 (51)] Le Duc, C., Le Thanh, N. (2002). Problématique de l'ebXML et logiques de description. *Rapport de recherche à l'I3S.*

[LB85 (52)] Levesque, H. J. et Brachman, R. J. (1985). A Fundamental Tradeoff in Knowledge Representation and Reasoning. *Readings in Knowledge Representation. Morgan Kaufmann.*

[LR98 (53)] Levy, A.Y. et Rousset, M-C. (1998). Combining Horn Rules and Description Logics in CARIN, *Artificial Intelligence,* 104, pp. 165-209.

[LR96 (54)] Levy, A.Y. et Rousset, M-C. (1996). The Limits on Combining Recursive Horn Rules with Description Logics. *Proceedings of American Conference on Artificial Intelligence, AAAI 96, Portland.*

[Min75 (55)] Minsky, M. (1975). A framework for representing knowledge. The Psychology of Computer Vision. *McGraw-Hill.*

[Moo97 (56)] Moore, S. O. (1997). A foundation for flexible automated electronic comunication. *Information System Research.*

[Mor75 (57)] Mortimer, M. (1975). On languages with two variables. *Zeitschr. f. math. Logik und Grundlagen d. Math., 21.*

[Neb90a (58)] Nebel, B. (1995). Reasoning and Revision in Hybrid Representation Systems. *Lecture Notes in Computer Science, Springer-Verlag.*

[Neb90b (59)] Nebel, B. (1990). Terminological reasoning is inherently intractable. *Artificial Intelligence*, 43, pp. 235-249.

[Ome02 (60)] Omelayenko, B. (2002). Integrating Vocabularies : Discovering and Representing Vocabulary Maps. *Proceeding of the First International Semantic Web Conference.*

[PA91 (61)] Pollock, J. L. et Anthony, S.G. (1991). Belief Revision and Epistemology. *University of Arizona, Tucson.*

[Qui68 (62)] Quillian, M. (1968). Semantic memory. Semantic Information Processing. *MIT Press, Cambrigde.*

[SS91 (63)] Schmidt-Schauß, M. et Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, *48,* pp. 1-26.

[Sow84 (64)] Sowa, J. F. (1984). Conceptural structures : Information Processing in Mind and Machine. *Addison Wesley.*

[SHG+01 (65)] Stevens, R., Horrocks, I., Goble, C. et Bechhofer, S. (2001). Building a Reason-able Bioinformatics Ontology Using OIL. *Dans Proceedings of the IJCAI-01 Workshop on Ontologies and Information Sharing.*

[TGL+88 (66)] Thayse, A., Gribomont, P., Louis, G., Snyers, D. et Wodon, P., Gochet, P., Grégoire, E., Sanchez, E., Delsarte, P. (1988). Approche Logique de l'Intelligence Artifitielle, *DUNOD Informatique.*

[UC01 (67)] UN/CEFACT (2001). Introduction, Business Modeling Business Requirements and Design of UMM. *UNCEFACT/TMWG/N097.*

[VTL+01 (68)] Van Rees, R., Tolman, F., Lima, C., Fies, B., Fleuren, J., Zarli, A. (2001). eBuisiness in Building and Construction : the eConstruct project. *eBusiness and eWork Conference.*

[Wac99 (69)] Wache, H. (1999). Towards Rule-Based Context Transformation in Mediators. *EFIS 1999*, pp. 107-122.

[WS01 (70)] Wache, H. et Stuckenschmidt, H. (2001). Practical Context Transformation for Information System Interoperability. *CONTEXT 2001*, pp. 367-380.

[WV02 (71)] Wache, H. et Vögele, T. (2002). Ontology-Based Integration of Information A Survey of Existing Approches. *Proceeding of the First International Semantic Web Conference.*

[Wil86 (72)] Wilf, H.S. (1986). Algorithmes et Complexité. *Prentice Hall.*