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

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 comme étant une 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 d'acteurs définissent un *contexte d'échange* qui permet de déterminer exactement la signification des données reçues. Cela explique pourquoi plusieurs formalismes sont utilisés dans les systèmes d'échange existants pour capturer des connaissances dont les natures sont très diverses.

En outre, certains formalismes envisagés disposent d'une sémantique informelle c'est-à-dire qu'elle n'est pas traitable par machine. Cela contrarie 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 des inférences nécessaires pour prendre en compte le contexte d'échange. Une des approches dans ce sens est la construction d*'ontologies* [**Fen, 2001**]. En général, les ontologies sont développées pour fournir aux sources d'information une sémantique traitable par machine. Dans le contexte d'un système d'échange, une ontologie disposant d'un formalisme expressif permettra de modéliser de façon cohérente des informations avec les natures diverses en fonction du domaine d'un 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.

Un certain nombre d'études ont été réalisés 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 différents acteurs. Toutefois,

[1]Nouvelles Technologies de l'Information et de la Communication
[2]eXtended Markup Language for Electronic Data Interchange
[3]Electronic Business eXtended Markup Language
[4]Formal Language for Business Communication

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 modales de la *logique du premier ordre* pour formaliser la sémantique des langages utilisés dans le commerce. Afin de devenir une concurrente d'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 et informel pourrait être approprié pour démontrer la nécessité de la sémantique formelle et de la transparence sémantique.

EXAMPLE. (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 ENTREPRISE-B : **ProdRésAuPolluant** est un **Produit** résistant à la fois au **Bruit** et au **Feu**.
- Acteur ENTREPRISE-C : **ProdRésAuPolluant** est un **Produit** résistant au **Bruit**.
- Echange : l'acteur ENTREPRISE-B qui est une entreprise **EntrepriseEurop**, *vend* un produit PRODUIT-A de **ProdRésAuFeu** à l'acteur ENTREPRISE-C qui est une entreprise **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 ENTREPRISE-C car le concept **ProdRésAuFeu** peut être différemment défini dans son dictionaire. Nous avons les cas suivants :
- Si le dictionnaire d'ENTREPRISE-C est traditionnellement organisé, une intervention humaine éventuelle permet d'interpréter la définition du terme reçu et d'identifier un terme approprié dans son dictionnaire.
- Si le dictionnaire d'ENTREPRISE-C est formalisé, la recherche de la définition qui est la plus proche de la définition reçue, peut être effectuée par machine.
- En supposant que les dictionnaires soient formalisés, la règle suivante est ajoutée à la base de connaissances partagée :
  - S'il y a une entreprise **Européen** Y qui *vend* un **Produit** X à une entreprise **Américain** Z, alors le produit X doit être défini selon l'entreprise **Européen** *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 des entreprises européennes. Cela signifie une modification du dictionnaire de l'acteur ENTREPRISE-C 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 se compose des connaissances implicites qui ne sont pas traitables par machine. Cela est au prix de 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 ENTREPRISE-C, 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 une application de la règle. En effet,

- Par la définition 1. , ENTREPRISE-B étant **EntrepriseEurop** implique que ENTREPRISE-B est **Européen**.
- Par la définition 2. , ENTREPRISE-C étant **EntrepriseAmér** implique que ENTREPRISE-C est **Américain**.
- Par la définition 3. , PRODUIT-A est **Produit**.
- Par le cas de l'échange, ENTREPRISE-B *vend* PRODUIT-A à ENTREPRISE-C.

Alors, la règle est appliquée et donc, ENTREPRISE-B et ENTREPRISE-B 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 d'ENTREPRISE-C devrait être interprétée selon la nouvelle définition.

L'idée de l'utilisation de la logique du premier ordre résultant de l'approche de FLBC [**Kim, 1998**] 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, applelé *logique de description*, peut être utilisé pour formaliser cette sémantique. En revanche, le *mécanisme de contexte* [**ebX, 2001g**] venant de l'ebXML évoque 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. Dans ce contexte, en supposant que toutes les ontologies du système dérivent d'une ontologie de base, nous pouvons présenter deux instances du problème de transparence sémantique à étudier :

- Les ontologies dérivées utilisent des langages fondés sur la même sémantique formalisée par la Logiques de Description dont les expressivités peuvent être différentes.
- La première instance est étendue 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 l'ebXML, d'autre part d'établir un formalisme hybride avec la sémantique formelle

permettant 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 de transparence sémantique.

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 prototypes 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.
//

La deuxième partie présente les contributions au développements d'algorithmes pour les deux instances du problème de transparence sémantique.
//

La troisième partie propose les ontologies pour le commerce électronique.
//

[5]Building and Construction eXtended Markup Language

Première partie

Approche Logique de la Représentation de Connaissances Hybride 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$^+$, 2003**]. 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$^+$, 1988**].
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 [**HU, 1979**], à celui-ci.

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 [**Min, 1975**]. 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 [**Hay, 1979**], 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 [**Qui, 1968**]. 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.

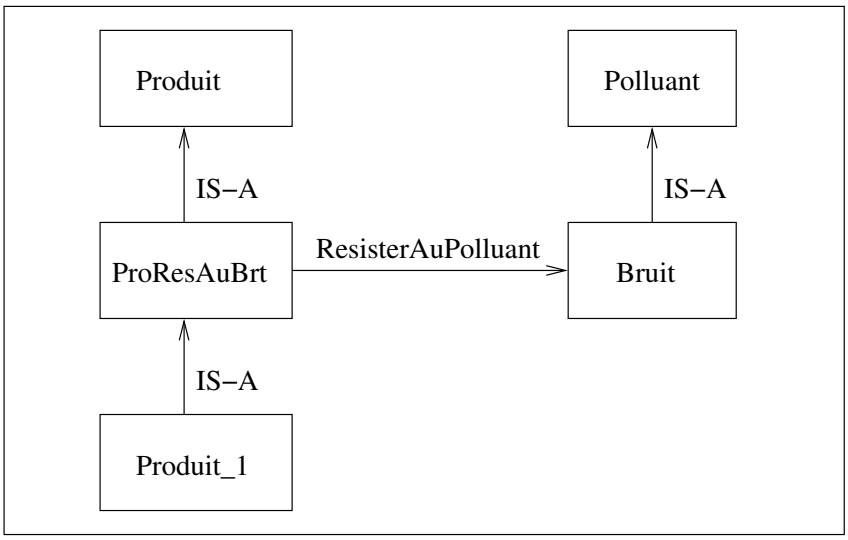

FIG. 1.3.1. Un réseau sémantique

Une des interprétations possibles de ce réseau est que Produit_1 est une instance du concept ProResAuBrt (ProduitRésisterAuBruit) qui est un sous-concept de Produit. Toute instance du concept ProResAuBrt 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 ProResAuBrt peut résister au feu, ou bien si les instances du concept ProResAuBrt ne résistent qu'au bruit. Par conséquent, différents systèmes de représentation des connaissances basés sur le même réseau sémantique pourraient avoir interprétation différente.

Brachman dans [**Bra, 1977, Bra, 1978**] 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 [**SS, 1991**] 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 ($\leq n\ r$, $\geq n\ r$) et négation primitive (ou complète).

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*.

1.3.2.1. *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

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

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	$\mathcal{EL}$	$\mathcal{FLE}$	$\mathcal{ALE}$	$\mathcal{ALC}$
$\top$	$\times$	$\times$	$\times$	$\times$
$\bot$			$\times$	$\times$
$\neg A$			$\times$	$\times$
$(C \sqcap D\ )$	$\times$	$\times$	$\times$	$\times$
$(C \sqcup D\ )$				$\times$
$\forall r.C$		$\times$	$\times$	$\times$
$\exists r.C$	$\times$	$\times$	$\times$	$\times$

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 :

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

$$\mathsf{ProResAuBrt} \equiv \mathsf{Produit} \sqcap \exists resisterAuPolluant.\mathsf{Bruit}$$

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

1.3.2.2. *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.

DEFINITION 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
ProResAuPoll :=	Produit $\sqcap \exists$*résisterAuPolluant*.Polluant
ProResAuBrt :=	Produit $\sqcap \exists$*résisterAuPolluant*.Bruit
ProResAuFeu :=	Produit $\sqcap \exists$*résisterAuPolluant*.Feu
ProPourBureau :=	ProResAuBrt $\sqcap$ ProResAuFeu

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 *dépilé* s'il n'existe pas de concept de définition qui contienne un nom défini. On peut obtenir le TBox dépilé d'un TBox acyclique en remplaçant systématiquement les noms définis par

leur concept de définition. Selon **[Neb, 1990b]**, la taille du TBox dépilé 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.

1.3.2.3. *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.

ProResAuPoll(FORMICA_1)	ProResAuBrt(FORMICA_1)
Feu(INF_1000)	*résisterAuPolluant*(FORMICA_1, INF_1000)

FIG. 1.3.5. 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 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 arssertions 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 ProPourBureau(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

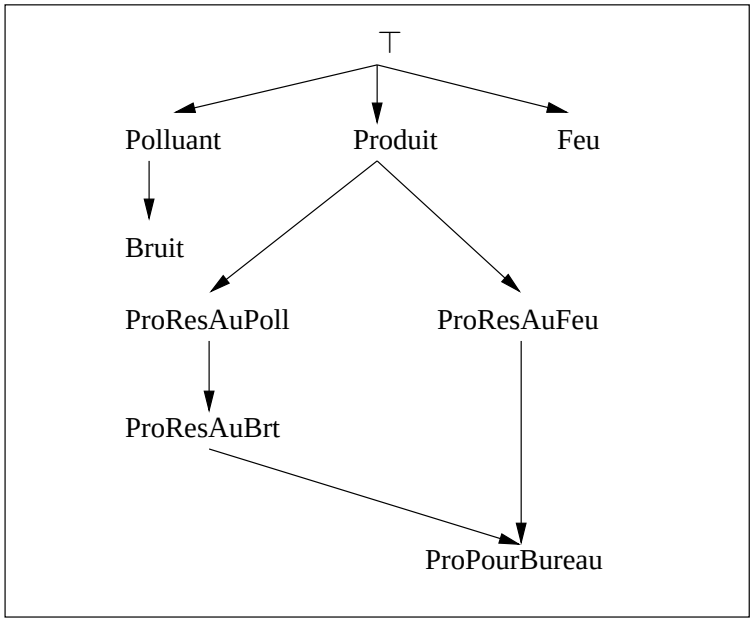

FIG. 1.3.6. Hiérarchie de subsomption

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

(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 [**SS, 1991**] 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$^+$, 1992**].

Par ailleurs, il y a une autre famille des inférences, appelée *inférence non-standard*, est également étudiée. Même si les inférences non-standards sont intéressées bien avant, les algorithmes corrects et complets pour ces inférences sont développés récemment [**BKM, 1999**]. 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*.

1.3.4.1. *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 montrée dans la Section 2.2.2 du Chapitre 2.

1.3.4.2. *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.

1.3.5. Applications (DAML+OIL, FaCT, RACER) (Annexes). DAML+OIL[1], les systèmes FaCT[2] et RACER[3]

[1]DARPA Agent Markup Language + Ontology Inference Layer
[2]Fast Classification of Terminologies
[3]Renamed Abox and Concept Expression Reasoner

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 [**Sow, 1984**]. 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 prédicate 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 [**BMT, 1999b**] 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 [**BMT, 1999b**] 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 restreint 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 :

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

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) ;

$\Gamma(\text{UNION } T_1,...,T_k \text{ END}) = \Gamma(T_1) \sqcup ... \sqcup \Gamma(T_k)$;

$\Gamma(\text{SET-OF } T) = \mathsf{SetType} \sqcap \forall \mathsf{member}.\Gamma(T)$;

$\Gamma(\text{RECORD } A_1 : T_1,...,A_k : T_k \text{ END}) = \mathsf{RecType} \sqcap \forall\Gamma(A_1).\Gamma(T_1) \sqcap (=1\ \Gamma(A_1)) \sqcap ... \sqcap \forall\Gamma(A_k).\Gamma(T_k) \sqcap (=1\ \Gamma(A_k))$.

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 [**CLN, 1999**].

1.5. Conclusion

Nous avons fait une partie de l'état de l'art de ce travail. Effectivement, les notions de base du domaine représentation de connaissances et le formalisme de la Logique de Description sont présentés. Cette logique qui est munie 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 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* [**Kim, 2000**]. Un des efforts afin de réunir les standards existants en introduisant un vocabulaire générique pour tout le domaine commercial est la proposition de la norme ebXML. Cependant, dans la nouvelle 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 et l'expressivité des langages de représentation. En effet, si le constructeur *fermeture transitive de rôle* récemment proposé dans [] 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 [**Mor, 1975**]. En outre, dans certains domaines d'application, il existe des connaissances qui sont inexprimables dans les Logiques de Description existantes, 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 ce travail. 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 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" **[Gru, 1993]**.

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 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 (DAML+OIL, OWL) étant employés pour concevoir 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.

Trois causes principales suivantes 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.

Ces conflits et donc, le problème d'interopérabilité sémantique peuvent être partiellement résolus si les sources d'informations des systèmes sont représentées comme des ontologies basées sur la Logique de Description (langage DAML+OIL). 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 correspondantes. Cela implique que le conflit de nom peut être réglé par une inférence de base implémentée dans les ontologies. De plus, le conflit d'unité peut se traduire

en conflit de confusion si les systèmes d'unité sont caractérisés par des informations contextuelles.

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

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

(2) Modifier les termes qui se ressemblent sémantiquement en utilisant l'inférence de subsomption.

(3) Vérifier la cohérence et la non-redondance de l'ontologie résultat.

La réalisation de ces étapes rencontre au moins les difficultés suivantes :

(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 existe des approches différentes à ces problèmes. D'abord, le langage utilisé pour décrire l'ontologie est représenté sous forme d'un méta-modèle. Ce dernier permet de représenter de façon uniforme les informations des applications basées sur les modèles. Cependant, ces approches ne prennent pas en compte tous les problèmes issus de la réalisation.

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 plus simple du problème de l'intégration des ontologies. Cette version, qui est connue comme problème de transparence sémantique dans le commerce électronique [**Kim, 2000**], est inspirée du modèle de représentation de données dans la norme ebXML
[**ebX, 2001a**]. Les détails de la norme ebXML seront discutés 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 [**ebX, 2001g**], 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* [**ebX, 2001h**], 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* [**ebX, 2001g**]. 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 la définition correspondant à ces valeurs contextuelles.

\\Référence à l'exemple d'introduction

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, l'un est plus expressif que l'autre, sur lesquels ces ontologies se basent sont différents. 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 certains cas, on a besoin de l'interprétation précise d'un concept reçu. Les définitions exactes peuvent être obtenues des applications de règles de contexte. En d'autres termes, si un concept défini dans l'ontologie récepteur qui a le même nom qu'un concept émetteur, alors la définition du concept dans l'ontologie récepteur devrait être modifiée. Cette modification est nécessaire, car une fois la transaction entre les deux acteurs est établie chaque référence à ce concept 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 [**Neb, 1990a**], et cette révision déclenche 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 est 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 qui sont 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 petite 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 $\Delta_{\mathcal{T}_2}$.

Example 2.1.1. (Exemple pour la première instance)

//

Pour formuler la deuxième instance du problème de transparence sémantique, nous avons besoin de préciser certaines conditions à satisfaire. Lorsque la transaction entre un émetteur et un récepteur est établie, les deux acteurs souhaitent partager la compréhension des connaissances. En d'autre termes, chaque référence à un nom de concept dans toute la transaction doit impliquer uniquement une définition. Par conséquent, 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 pour chaque référence à *ce concept* . 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 sous 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 la procédure de la 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.

Example 2.1.2. (Exemple pour la deuxième instance)

//

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* [**BKT, 2002a**], est exigée pour calculer la L_2-description de concept qui est "la plus proche" qu'une L_1-description de concept inexprimable dans le langage L_2. De plus, une autre inférence non-standard, appelée *réécriture* [**BKM, 2000**], 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 d'un ABox. 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 [**Gär, 1992**]. 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 [**AGM, 1985**] se traduisent convenablement en ceux pour la révision d'un TBox [**Neb, 1990a**].

Il y a encore une tâche à accomplir dans la deuxième instance. Il s'agit d'une inférence sur le formalisme hybride, qui permet de calculer la série de transformations 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 pour 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 [**ebX, 2001e**] est décrit en langage UMM[1] [**Cla, 2000**] 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$^+$, 2002**] 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 1.3.4.1 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. Une introduction d'un

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

travail traitant la combinaison des règles de Horn et une Logique de Description expressive, dit langage CARIN [**LR, 1998**], 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 1.3.4.2.

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$^+$, 2002**].

La sémantique des diagrammes de classe UML consiste en 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.

2.2.1.1. *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$^+$, 2002**].

Par ailleurs, le travail dans [**CDL, 2001**] 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 [**CDL, 2001**].

2.2.1.2. *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.

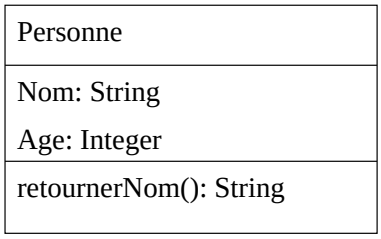

FIG. 2.2.2. Une classe UML

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 [**CDL, 2001**].

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.

2.2.1.3. *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

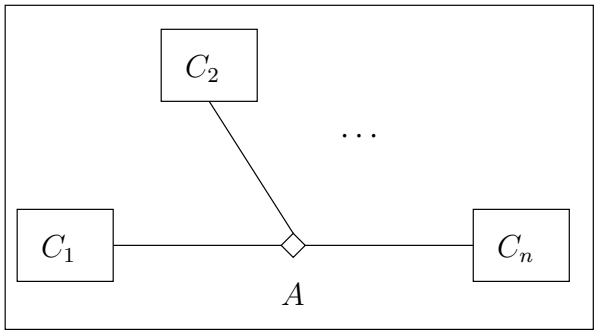

FIG. 2.2.4. Une association n-aire dans UML

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 :

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

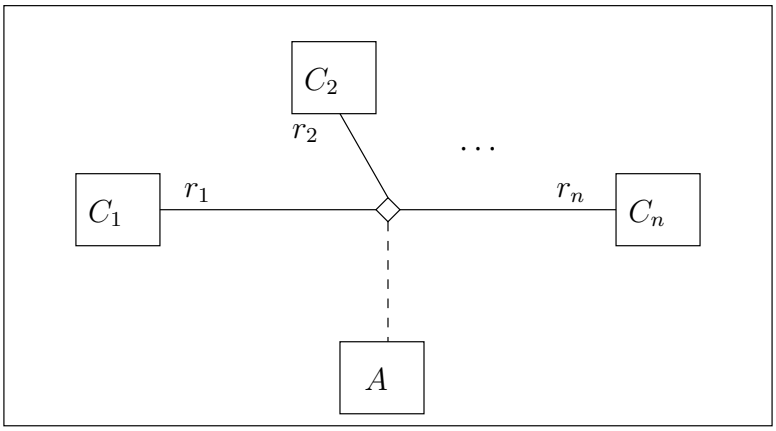

FIG. 2.2.5. Une association n-aire avec la classe associée dans UML

2.2.1.4. *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 :

$$C_1 \sqsubseteq C_2$$

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 :

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

2.2.1.5. *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 :

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

2.2.1.6. *Exemple.* Le TBox dans la Figure 1.3.4 peut se traduire en diagramme de classe UML dans la Figure 2.2.6.

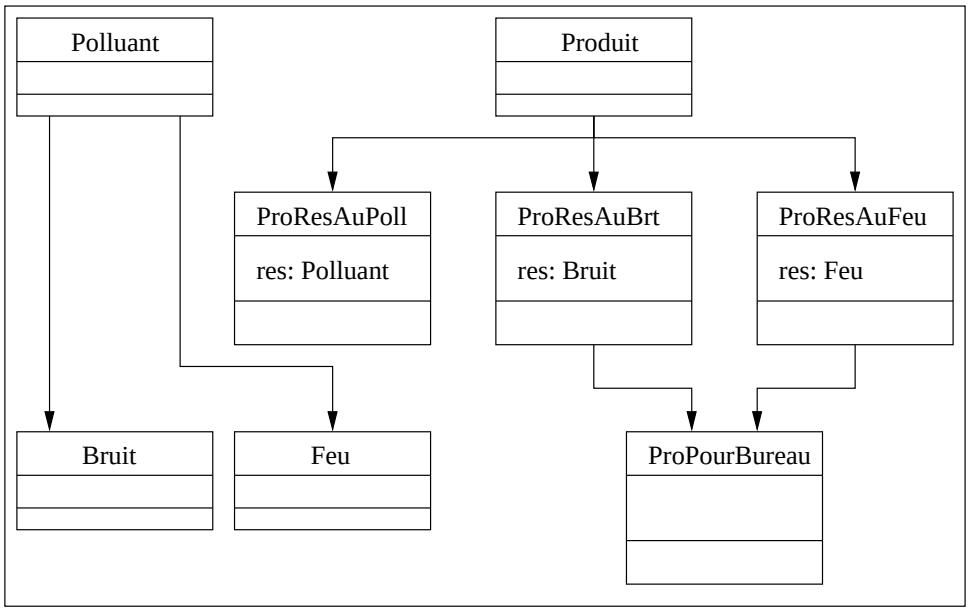

FIG. 2.2.6. Un diagramme de classe UML

Le diagramme de classe dans la Figure 2.2.6 peut être formalisé en langage $\mathcal{DLR}$ comme suit :

Bruit	$\sqsubseteq$	Polluant
Feu	$\sqsubseteq$	Polluant
ProResAuPoll	$\sqsubseteq$	Produit $\sqcap \forall[1](\textit{résisterAuPolluant} \Rightarrow$(2 :Polluant))
ProResAuPoll	$\sqsubseteq$	$\geq 1[1]\textit{résisterAuPolluant}$
ProResAuBrt	$\sqsubseteq$	Produit $\sqcap \forall[1](\textit{résisterAuPolluant} \Rightarrow$(2 :Bruit))
ProResAuBrt	$\sqsubseteq$	$\geq 1[1]\textit{résisterAuPolluant}$
ProResAuFeu	$\sqsubseteq$	Produit $\sqcap \forall[1](\textit{résisterAuPolluant} \Rightarrow$(2 :Feu))
ProResAuFeu	$\sqsubseteq$	$\geq 1[1]\textit{résisterAuPolluant}$
ProPourBureau	:=	ProResAuBrt $\sqcap$ ProResAuFeu

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

ProPourHabitation := ProPourBureau $\sqcap$ ProResAuPoll

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

2.2.2. Inférences non-standards . Les inférences s'appuyant sur les algorithmes de décision suscitent beaucoup d'intérets 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 [**Wil, 1986**]. 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 petit 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*).

2.2.2.1. *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.

[3]Fast Classification of Terminologies

[4]Renamed Abox and Concept Expression Reasoner

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 subsumeur commun le plus petit de $\mathcal{ALE}$-descriptions de concept, est décrite.

DEFINITION 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 avec la forme où

- 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 $\mathcal{G}$;
- 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.

EXAMPLE 2.2.2. //

La Figure ? est l'arbre de description du concept C dans l'Exemple 2.2.2. 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.

DEFINITION 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.

DEFINITION 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 :

THEOREM 2.2.5. **[BKM, 1999]** *Soient* C, D *des $\mathcal{ALE}$-descriptions de concept. Alors* C $\sqsubseteq$ D *si et seulement s'il existe un homomorphisme de $\mathcal{G}_D$ dans $\mathcal{G}_C$.*

D'après **[DHL$^+$, 1992]**, 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.

EXAMPLE 2.2.6. //

2.2.2.2. *Subsumeur commun le plus petit (lcs).*

DEFINITION 2.2.7. 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 petit 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 petite 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 subsumeur commun le plus petit 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 subsumeur commun le plus petit 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, $lcs(C_1,...,C_k) \equiv C'$.

REMARK 2.2.8. (Propriétés de lcs)
A partir de la Définition 2.2.7, 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.

DEFINITION 2.2.9. 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 petit 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.

THEOREM 2.2.10. **[BKM, 1999]** *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 subsumeur commun le plus petit 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 2.2.11. **[BKM, 1999]** *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.*

2.2.2.3. *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.* (2002) 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.

DEFINITION 2.2.12. 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'.

Notons que si L_2=$\mathcal{ALE}$ et L_2=$\mathcal{ALC}$, alors $approx(C)$ existe uniquement. En effet, supposons que D' soit une $\mathcal{ALE}$-description de concept qui vérifie la Définition 2.2.12. Puisque

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

alors, par définition, on obtient : $approx(C) \sqcap D' \equiv approx(C) \equiv D'$.

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.

DEFINITION 2.2.13. 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) Alors, la *forme normale* de C est définie comme suit :
$C = C_1 \sqcup ... \sqcup C_n$
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.

THEOREM 2.2.14. **[BKT, 2002a]** *Soit D une $\mathcal{ALC}$-description de concept dans la forme normale spécifiée par la Définition 2.2.13, 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.

ALGORITHM 2.2.15. **[BKT, 2002a]**
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 2.2.15 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 ?

2.2.2.4. *Exemple.* Supposons que dans une ontologie basée sur le langage $\mathcal{ALC}$, le concept ProResAuPoll soit redéfini en langage pré-DAML-OIL comme suit :

<EQUALC>
<CONCEPT>
<PRIMITIVE NAME="PORTE_RESISTANT_POLLUANTS"/>
</CONCEPT>
<CONCEPT>
<AND>
<PRIMITIVE NAME="PORTE"/>
<SOME>
<PRIMROLE NAME="resisterPollution"/> <TOP/>
</SOME>
<OR>
<ALL>
<PRIMROLE NAME="resisterPollution"/> <PRIMITIVE NAME="FEU"/>
</ALL>
<ALL>
<PRIMROLE NAME="resisterPollution"/> <PRIMITIVE NAME="BRUIT"/>
</ALL>
</OR>
</AND>
</CONCEPT>
</EQUALC>

D'après l'algorithme 2.2.15, cette description de concept peut être approximée par une description de concept dans le langage $\mathcal{ALE}$ comme suit :

<AND>
<SOME>
<PRIMROLE NAME="resisterPollution"/> <PRIMITIVE NAME="POLLUANTS"/>
</SOME>
<ALL>
<PRIMROLE NAME="resisterPollution"/> <PRIMITIVE NAME="POLLUANTS"/>
</ALL>
</AND>

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 constitué. 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 pas être exprimé par l'autre [**Bor, 1994**], 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$^+$, 1998b**]. Le langage CARIN [**LR, 1998**], 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 1.3.4.2) 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 quelques 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 (voir **[LR, 1998]**)
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 (voir **[LR, 1998]**).
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 (voir **[LR, 1998]**).

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 $\beta \cup \Delta_{\mathcal{T}}$ **[LR, 1998]**.

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 **[LR, 1998]**) :

(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 :

THEOREM 2.2.16. **[LR, 1998]** *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.

THEOREM 2.2.17. **[LR, 1998]** *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 [**LR, 1998, LR, 1996**] 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.

Afin d'illustrer l'idée principale de l'algorithme décrit, nous considérons l'exemple suivant.

EXAMPLE 2.2.18.

$\mathcal{T}_1$:= {
European $\sqcap$ American $\sqsubseteq \bot$
European-associate := $\exists$associate.European
American-associate := $\exists$associate.American
No-fellow-company := $\forall$associate.$\neg$American
International-company := European-associate $\sqcup$ American-associate
}
$\mathcal{A}_1$:= { Made-by(a,b), Monopoly(b,a,usa), American-associate(b) }
$\mathcal{R}_1$:= {
r_1 : Made-by(X,Y) $\wedge$ No-fellow-company(Y) $\Rightarrow$ Price(X, usa, high)
r_2 : Made-by(X,Y) $\wedge$ Associate(Y,Z) $\wedge$ American(Z) $\wedge$ Monopoly(Y, X, usa) $\Rightarrow$ Price(X, usa, high)
}
Avec la B.C. Δ_1=($\mathcal{T}_1$, $\mathcal{A}_1$, $\mathcal{R}_1$), l'affirmation suivante est vérifiée :
$\Delta_1 = \models$ *Price(a, usa, high)*
En effet, les faits *American-associate(b)* et *American-associate* := $\exists$*associate.American* dans $\mathcal{T}_1$ permettent de déduire *American(b)*. Dans ce cas-là l'antécédent de la règle r_2 est vérifié. Par conséquent, *Price(a, usa, high)* est vérifié. Notons qu'aucune règle de $\mathcal{R}_1$ ne peut être complètement instanciée à partir de $\mathcal{A}_1$.

Si l'on suppose avoir un autre ensemble des faits :
$\mathcal{A}_2$:= { Made-by(a,b), Monopoly(b,a,usa), International(b) }, l'affirmation suivante est vérifiée :
$\Delta_2 = \models$ *Price(a, usa, high)* où Δ_2=($\mathcal{T}_1$, $\mathcal{A}_2$, $\mathcal{R}_1$)
En effet, on doit recourir au raisonnement d'analyse de cas : i) si b a au moins un associé américain alors l'antécédent de la règle r_2 est vérifié. Donc, on a *Price(a, usa, high)*. ii) si b n'a aucun associé américain alors *No-fellow-company(b)* sera vérifié. Donc, *Price(a, usa, high)* est vérifié par la règle r_1.

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 [**BCD, 2003**] 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 : 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'augementer 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

Représentation compacte des descriptions de concept et inférences non-standard

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 *2.2.15* vient des deux sources suivantes. Premièrement, la normalisation d'une $\mathcal{ALC}$-description de concept selon la Définition 2.2.13 peut générer une description de concept dont la taille est exponentielle en fonction de la taille de la description de concept d'entrée [**?**]. 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 [**BKM, 1999**]. 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 Petit Subsumeur Commun (*lcs*) (Définition 2.2.7) dans la Logique de Description $\mathcal{ALE}$. Comme montré dans [**BT, 2002**], 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 les *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 [**?**], 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 de vide arcs, 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. 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.

DEFINITION 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) $\{w_1, ..., w_m\} \in N^k$ si $u_1 \forall r w_1, ..., u_m \forall r w_m$

   (b) $\{w_1, ..., w_m, w_{m+1}\} \in N^k$ si $u_1 \forall r w_1, ..., u_m \forall r w_m$ et $u_i r w_{m+1}$, $u_i \in n^{k-1}$

REMARK 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 $n^k_i \neq n^k_j$.

DEFINITION 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 :

DEFINITION 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 $v r v''$ et $v \forall r v'$, 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 $u_i \forall r w_i$, $u_j \forall r w_j$ 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 $u_i r w_i$, $u_j \forall r w_j$. 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 $u_1 \forall r w_1$, $u_2 \forall r w_2$ ou $u_1 r w_1$, $u_2 \forall r w_2$. 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 $u_1 \forall r w_1$, $u_2 \forall r w_2$ ou $u_1 r w_1$, $u_2 \forall r w_2$ (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. □

REMARK 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.

DEFINITION 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 $\mathcal{G}^{\epsilon}_C$ 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, cette 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 des nœuds de l'arbre résultat.

ALGORITHM 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 $\mathcal{G}$.

(1) *Au niveau, un voisinage unique 0-voisinage $\{v_0\}$, qui est la racine de $\mathcal{G}$, 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 $\mathcal{G}$ 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 $n^{k-1} \forall r n^k$.*
   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 $\mathcal{G}$ obtenu dans l'étape 2. :*

   (a) *$l(v) \to l(v) \setminus \{\top\}$ où $v \in V_G$ (règle 4)*
   (b) *$v' r v \to v$ si $l(v) = \{\bot\}$, v est une feuille et $v' r v \in E_G$ (règle 6)*
   (c) *$\mathcal{G}_C(v) \to \bot$ où $\bot \in l(v)$ et $\mathcal{G}^\epsilon(v)$ est un sous-arbre de $\mathcal{G}$ (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.

EXAMPLE 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.

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

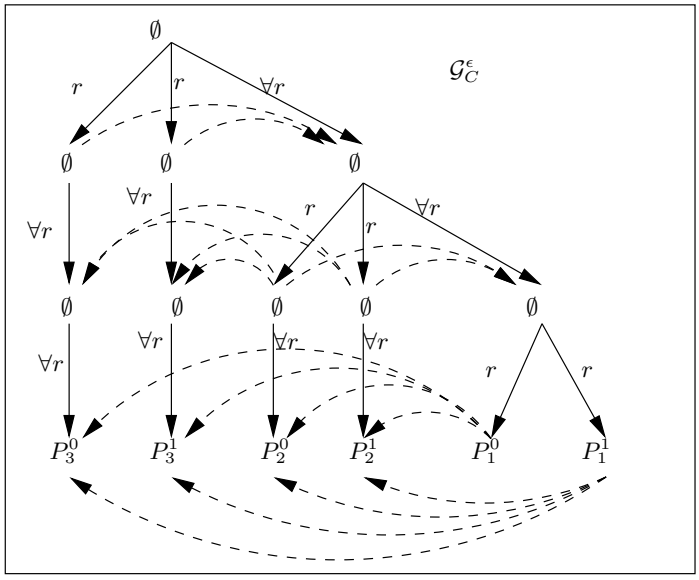

FIG. 3.2.1. ϵ-arbre de description

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 $\mathbf{B}$ 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 $\{v_1, ..., v_m\} = \bot$. 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 $k \leq |\mathcal{G}|$. □

REMARK 3.2.12. (complexité et subsomption)

(1) L'Algorithme 3.2.8 prend un temps exponentiel et une 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.

(2) Quant à la subsomption, la vérification de l'existence d'un homomorphisme entre deux arbre de descriptions peut être effectuée en temps polynômial en fonction de la taille de l'arbre d'après [**BKM, 1999**]. De plus, cette vérification nécessite également une espace polynômiale en fonction de la profondeur de l'arbre de description selon [**BKM, 1999**] et [**SS, 1991**].
D'autre part, dans l'Algorithme 3.2.8, chaque $(k-1)$-voisinage a au plus un nombre polynômial de k-voisinages en fonction de la taille de la description de concept d'entrée. Ainsi, l'algorithme effectue au plus un calcul d'un nombre exponentiel des chemins de l'arbre en fonction de la taille de la description de concept d'entrée. Par conséquent, grâce à la correspondance entre l'arbre de description et le ϵ-arbre de description la subsomption établie par la Proposition 3.2.11, nous pouvons conclure que la subsomption dans le langage $\mathcal{ALE}$ basée sur la représentation du ϵ-arbre de description est exponentielle en temps et polynômial en espace.

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).

Formellement, à l'exception de la présence de ϵ-arcs et de drapeaux, la définition du produit de deux ϵ-arbres de description n'est pas différente de la définition du produit de deux $\mathcal{ALE}$-arbres de description décrite dans [**BKM, 1999**].

DEFINITION 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 : $l(\{v_1, v_2\}) = eval(\{\{P\} \cap \{Q\}\}, \{\neg P\}) = \{\neg P, Q\}$.

EXAMPLE 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 $A, B, C \in N_C$, $r \in N_R$

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,

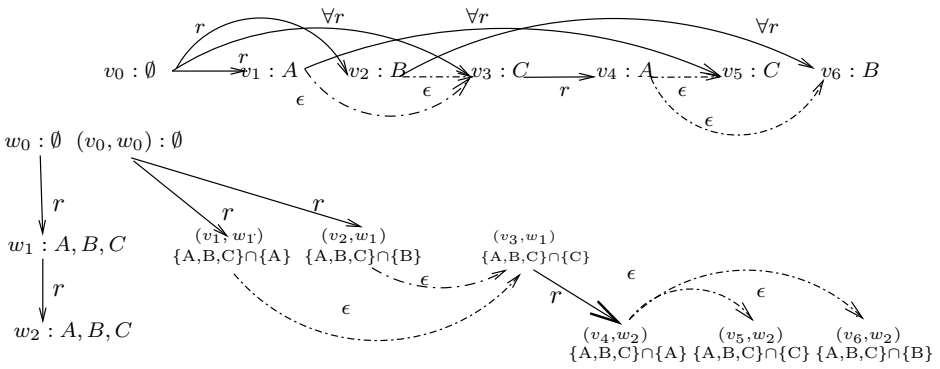

FIG. 3.3.1. Product

lorsque l'on calcule le produit de deux ϵ-arbres de description simple, par exemple selon la Définition 2.2.9, le ϵ-arbre de description obtenu peut contenir un ϵ-arc tel qu'il n'existe pas d'arc normal dont une extrémité est une destination de ce 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 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 $(v'ev)$, $e \neq \epsilon$.

DEFINITION 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.

DEFINITION 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 est généré par la Définition 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.*

ALGORITHM 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 $\mathcal{G}$.

(1) *Au niveau 0, un 0-voisinage $\{v_0, v_{0,1}, ..., v_{0,m}\}$, qui est la racine de $\mathcal{G}$, 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* $\mathcal{G}^\epsilon_C$ *correspondant au* $(k-1)$*-nœud de* $\mathcal{G}$ *où* $\bot \notin \bigcup^m_{v_i \in n^{k-1}} \{l(v_i)\}$*, les k-nœuds et les arcs de* $\mathcal{G}$ *sont créés d'un k-voisinage du* $\mathcal{G}^\epsilon_C$ *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* $m^k_j = n^k \cup \{v'_j\} \cup V^\epsilon$*,* n^k *est déterminé dans l'étape (a) , et les r-arcs correspondants :* $n^{k-1} r m^k_j$,
$l(m^k_j) = eval_{v' \in m^k_j}\{l(v')\}$.

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

(a) $l(v) \to l(v) \setminus \{\top\}$ *où* $v \in V_G$ *(règle 4)*

(b) $v'rv \to v$ *si* $l(v) = \{\bot\}$*,* v *est une feuille et* $v'rv \in E_G$ *(règle 6)*

(c) $\mathcal{G}_C(v) \to \bot$ *où* $\bot \in l(v)$ *et* $\mathcal{G}^\epsilon(v)$ *est un sous-arbre de* $\mathcal{G}$ *(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 :* $\{l(v_1), ..., l(v_m)\} := \bigcup^m_i l(v_i) =$
$(Q_{1,1}, Q_{2,1}, ..., Q_{m,1}) \cap ... \cap (Q_{1,m_1}, Q_{2,m_2}, ..., Q_{m,m_m})$

(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 Petit 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 petit 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 **B** du côté gauche est appliquée à un ϵ-arbre de description étendu alors que l'opérateurs **B** du côté droit sont appliqués aux ϵ-arbres de description simples.

THEOREM 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* $\varphi'(m_i^{k+1}, n_j^{k+1}) = q_j^{k+1}$.

□

EXAMPLE 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 conjunction de disjunctions de noms de concept.

Dans l'exemple suivant qui est extrait de **[BT, 2002]**, 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.

EXAMPLE 3.3.9. (extrait du **[BT, 2002]**)
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 3.3.2 illustre la représentation graphique de l'Exemple 3.3.9.

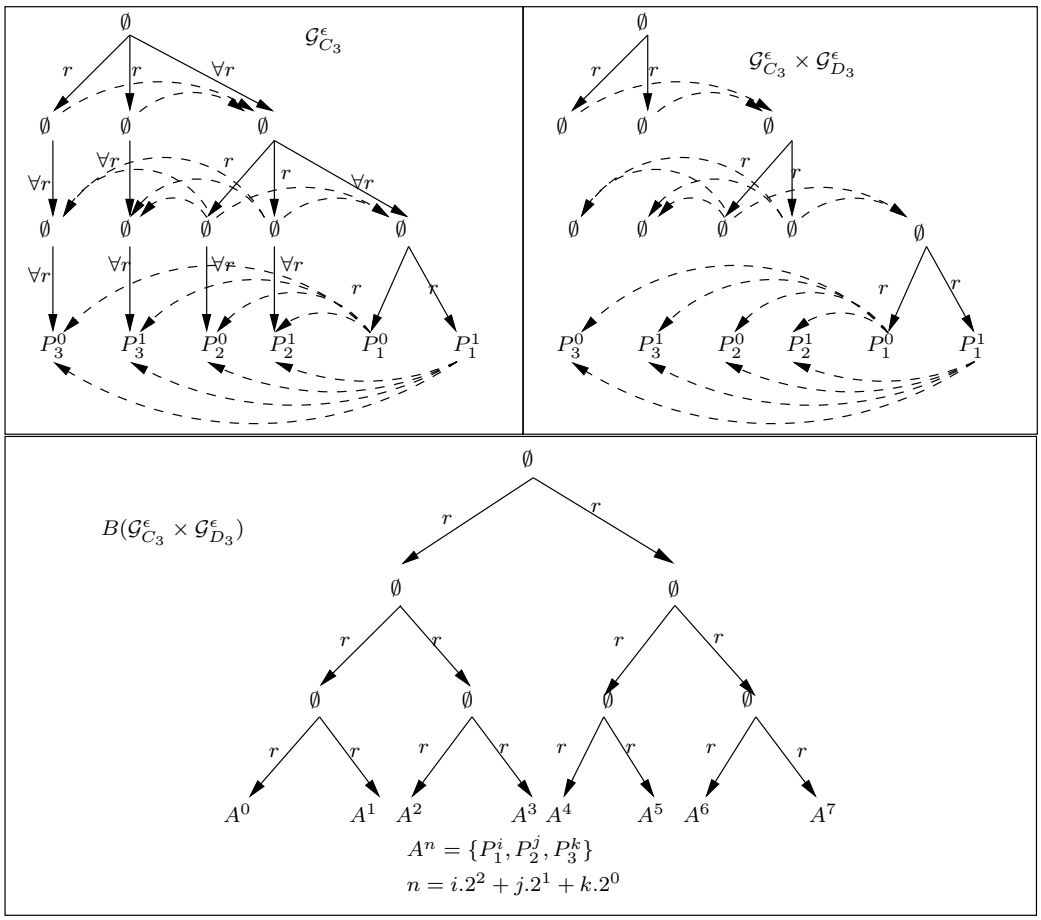

FIG. 3.3.2. Produit $\mathcal{G}^{\epsilon}_{C_3} \times \mathcal{G}^{\epsilon}_{D_3}$ et sa représentation ordinaire obtenue par l'Algorithme **??**

REMARK 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.

La Proposition *3.2.11* et le Théorème 3.3.7 nous permettent de représenter le Plus Petit 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 a besoin de l'application de l'opérateur **B** au ϵ-arbre de description étendu.

Ce résultat et la nouvelle représentation des $\mathcal{ALE}$-descriptions de concept par ϵ-arbre de description nous permettent d'une part de calculer la subsomption en espace polynômiale, d'autre part de calculer les Plus Petits Subsumeurs Communs de deux $\mathcal{ALE}$-descriptions de concept en temps polynômial. Cela implique que l'on peut calculer le Plus Petit Subsumeur Commun de deux $\mathcal{ALE}$-descriptions de concept sans normalisation par la règle 2. dans la Définition 2.2.4. Comme indiqué dans [**BKM, 1999**] et [**BT, 2002**], l'application de cette règle accroît de façon exponentielle la taille du Plus Petit 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 Petit 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 Petit 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 2.2.9, 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 Petit 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 **??**, puisque A et B sont les $\mathcal{ALE}$-descriptions de concept normalisées, l'arbre de description $\mathcal{G}_A$ est une concaténation exacte des deux arbres de description $\mathcal{G}_{A_1}$ et $\mathcal{G}_C$. De même, l'arbre de description $\mathcal{G}_A$ 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 **??**, 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 puisse être écrite comme suit : $C = C_1 \sqcup C_2$ où $\bot \sqsubset C_1,\ C_2$. 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 $approx_{\mathcal{ALE}}(C_1) \sqcup approx_{\mathcal{ALE}}(C_2) \sqsubseteq lcs(approx_{\mathcal{ALE}}(C_1),\ approx_{\mathcal{ALE}}(C_2))$.
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 $d^{\mathcal{I}} \notin D^{\mathcal{I}}$.
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 est une contradiction 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 est une contradiction 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 *2.2.15* est qu'il peut fournir une implémentation plus simple. Pour des raisons de simplicité, on écrit $approx(C)$ à la place de $approx_{\mathcal{ALE}}(C)$.

ALGORITHM 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 *2.2.15* est dans le pire des cas doublement exponentiel en fonction de la taille d'entrée. Une question laissée ouverte par les auteurs du travail **[BKT, 2002a]** 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 *2.2.15,* 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}.

3.4.2.1. *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 :

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

Selon l'Algorithme *2.2.15*, 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 : $E(B_2, B_1, B_1, B_1) = EX_{(I)}(B_2, B_1, B_1, B_1) \cup EX_{(II)}(B_2, B_1, B_1, B_1)$ $\cup\ EX_{(III)}(B_2, B_1, B_1, B_1)$.

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 la 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 $f \not\sqsubseteq e$.

3.4.2.2. *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 *2.2.15*, $approx_{ALE}(C)$ est calculée comme suit :
$approx_{ALE}(C) \equiv$

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

$\forall r.\text{lcs}\{val(C_i) \mid 1 \leq i \leq m\}$
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 :

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

On désigne par $EX_{(III)}(B'_1, ..., B'_m)$ l'ensemble

$EX_{(III)}(B'_1, ..., B'_m) := \{Q_{i_k} \sqcap P^{\bar{i}_1}_1 \sqcap P^{\bar{i}_2}_2 \sqcap ... \sqcap P^{\bar{i}_n}_n \mid B'_k = \exists r.(Q_{i_k} \sqcap \bigsqcap_{i=1..n}(P^1_i \sqcap P^2_i)), \bar{I}(k) = (\bar{i}_1, ..., \bar{i}_n), 1 \leq k \leq m\}$.

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^i_v,$A^j_v \in \{A'_1,...,A'_p\}$, $1 \leq v \leq n$, $i,j \in \{1,2\}$. Cela implique que $e \in E_1(B'_1,...,B'_m)$.

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

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

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

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

$$\forall r.\Big(\bigsqcap_{(j_1,...,j_n)\in(\{1,2\}\times...\times\{1,2\})} \exists r.(P^{j_1}_1 \sqcap ... \sqcap P^{j_n}_n \sqcap Q_1 \sqcap Q_2) \sqcap \bigsqcap_{k=1}^{n} \exists r.(Q_1 \sqcap Q_2 \sqcap \bigsqcap_{l=1,l\neq k}^{n}(P^1_l \sqcap P^2_l)) \Big)$$

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

- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P^{\bar{i}_1}_1 \sqcap ... \sqcap P^{\bar{i}_n}_n)$ pour tout $e' \in E_1(B'_1,...,B'_m)$ car $\{Q_2, P^{\bar{i}_1}_1, ..., P^{\bar{i}_n}_n\} \not\subseteq \{Q_1,Q_2\} \cup \bigcup_{l=1,l\neq h}^{n}\{P^1_l, P^2_l\}$ pour $h \in \{1..n\}$.
- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P^{\bar{i}_1}_1 \sqcap ... \sqcap P^{\bar{i}_n}_n)$ pour tout $e' \in E_2(B'_1,...,B'_m)$ car $\{Q_2, P^{\bar{i}_1}_1, ..., P^{\bar{i}_n}_n\} \not\subseteq \bigcup_{l=1}^{n}\{P^1_l, P^2_l\}$.
- $e' \not\sqsubseteq \exists r.(Q_2 \sqcap P^{\bar{i}_1}_1 \sqcap ... \sqcap P^{\bar{i}_n}_n)$ 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 $B'_k = \exists r.(Q_1 \sqcap \bigsqcap_{v=1}^{n}(P^1_v \sqcap P^2_v))$ 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 $approx_{ALE}(C)$.

REMARK 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 2.2.13. En effet, dans le langage $\mathcal{EL}$, la taille exponentielle d'un lcs peut résulter de l'applications de ce lcs à des n $\mathcal{EL}$-description de concept [**BKM, 1999**].

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 Petit 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 de 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 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 Inférence avec 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 et des rôles 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 c'est-à-dire que l'expansion de la terminologie ne peut pas remplacer la révision. La raison est que le mécanisme de la *référence par nom* est utilisé dans la terminologie. 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
[**AGM, 1985**] 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 cohérente. Par contre, un ajout d'un concept avec la définition à une terminologie ne cause jamais d'incohérence 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 ces approches sémantiques ne peuvent pas être directement appliquées à la révision de la terminologie, d'autre part l'approche syntaxique proposée dans [**Neb, 1990a**] ne respecte pas toujours les principes établis. Cela nous amène à construire une autre méthode, appelée *approche structurelle*, qui est présentée dans la troisième section du chapitre. L'approche structurelle permet à la fois de capturer le mécanisme de la référence par nom et de définir les opérations de révision qui satisfont les principes cités. Les opérations de révision issues de l'approche structurelle sont capables de traiter un langage DL 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 Terminologies

Cette section commence par un exemple. Supposons qu'une BC 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 BC. Toutefois, notre BC devient incohérent. Si nous voulons maintenir la cohérence de la BC, 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 BC 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 BC ?

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 :

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

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 :
$K + P := Cn(K \cup \{P\})$ 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 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 les deux identités suivantes comme hypothèses :
Identité de Levi : $K * P = (K - \sim P) + P$ et,
Identité de Harper : $K - P = (K * \sim P) \cap P$.
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 [**Ant, 2000**].

(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)
(K7) $(K - P) \cap (K - Q) \subseteq K - (P \wedge Q)$
(K8) Si $P \notin K - (P \wedge Q)$, $A - (P \wedge Q) \subseteq K - P$

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 **[Ant, 2000]**. 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 les 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 BC un ensemble de toute base de connaissances possible, L un langage pour exprimer les requêtes (langage de requête) :

$$\begin{aligned} \text{DIRE} &: BC \times L \rightarrow BC \quad : \text{DIRE}(C, \mathit{Exp}) \\ \text{OUBLIER} &: BC \times L \rightarrow BC \quad : \text{OUBLIER}(C, \mathit{Exp}) \end{aligned}$$

A partir du canevas AGM, les principes suivants sont identifiés pour que les opérations de révision les satisfassent **[Neb, 1990a]** :
(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 BCT (Base de Connaissances Terminologiques).
(P2) **Indépendance de syntaxe**. Si les expressions de requête de révision sont sémantiquement équivalentes, les opérations respectives causent les mêmes effets sur la BCT.
(P3) **Fermeture**. Ce critère exige que n'importe quelle opération de révision doive amener la BCT à un état unique interprétable par le formalisme utilisé.
(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 (Exp) doit être dérivable de la BCT et après OUBLIER la connaissance enlevée (Exp) doit ne plus être dérivable de la BCT.
(P5) **Modification minimale** (selon une mesure de minimalité). Cela nous entraîne à définir la distance entre deux états de la BCT. Des considérations pragmatiques sont exigées pour fixer le sens de la notion "*modification minimale*" dans les cas réels.
A priori, ces principes ne s'adaptent pas directement à la révision d'une base de connaissances terminologique. D'une part, la négation et la disjonction au niveau propositionnel ne correspondent à rien dans les terminologies (P3). D'autre part, si l'on considère l'introduction d'un nouveau terme comme une expression de requête de

révision, il n'est pas évident que comment exprimer les opérations DIRE et OUBLIER pour procéder cette expression (P1).
Maintenant nous faisons un bref état de l'art sur les solutions existantes aux problèmes de la révision de la terminologie.

4.2.1. Éditeur du réseaux (système BACK). Cette approche propose de manipuler directement le réseau de la base de connaissances terminologiques. Par exemple, une modification d'une restriction universelle est effectuée sur les liens entre deux concepts. La suppression d'un concept du réseau, autorisée par cette approche, peut générer quelques problèmes importants : références nulles, l'ambiguïté qui apparaîtra après une suppression d'un lien hérité si la distinction entre les liens littéraux et dérivés n'est pas maintenue. Il est évident que ces problèmes ne peuvent qu'être corrigés au niveau conceptuel.

4.2.2. Éditeur de la base de connaissances (système KREME). Un tel système permet d'ajouter/supprimer de nouveaux concepts et rôles, de modifier des définitions de concepts existants. Cette modification peut changer des relations sémantiques du système. Pour l'opération de suppression, cette approche suggère de supprimer l'introduction d'un terme, et soit de supprimer toute occurrence de ce terme dans les autres termes, soit de remplacer une occurrence de ce terme dans les autres termes par son côté droit. Il semble qu'aucun de ces choix ne soit convainquant. Le premier amène à une terminologie réduite à la fois sans terme supprimé et sans signification originale de ce terme. Le second empêche d'utiliser ce terme mais préserve sa "signification" dans la terminologie. Cela montre que ni intuition ni sens formel peuvent justifier cette approche.

4.2.3. Ajout ou suppression de définitions (système LOOM). Cette approche essaye de réduire et de limiter l'ensemble d'opérations décrit dans l'approche précédente. C'est-à-dire :

- Ajouter une introduction de terme à une terminologie.
- Enlever une introduction de terme d'une terminologie et la remplacer par un concept primitif.

Ces opérations manipulent une terminologie dans laquelle la base de connaissances est considérée comme une collection des introductions de termes symboliques plutôt que comme un *bloc de connaissances*. C'est pourquoi nous pouvons proposer sans difficulté une description formelle pour les opérations de révision. Notons que l'opération OUBLIER dans cette approche peut être définie via DIRE.
Si nous évaluons cette approche en utilisant les principes (P1)-(P5), la majorité de ces principes sont satisfaits. En effet, (P1) est satisfait car l'expression de requête de révision porte un sens : introduire un terme. De même, la base de connaissances obtenue après les exécutions des opérations est bien définie. Donc, (P3) est satisfait. De plus, toute expression DIRE est réussie dans la mesure où les relations de subsomption exprimées par l'introduction de terme ajoutée à la base de connaissances sont dérivables. En outre, si deux introductions de terme sont sémantiquement équivalentes, alors les effets sémantiques, *i.e* relations de subsomption, sont les mêmes. Donc, (P2)

et (P4) sont satisfaits. Il nous reste le principe le plus problématique : (P5). Toutefois, ce principe peut être satisfait si nous adaptons les définitions des opérations de façon raisonnable. Cette solution sera présentée dans l'approche suivante.
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. 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. C'est pourquoi une hypothèse implicite de la conception d'une terminologie est qu'un nom de concept devrait être utilisé pour se référer simplement à une structure donnée.

4.2.4. Approche sémantique (système KRYPTON). 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. La première conséquence de tels systèmes 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. 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 suite :

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

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

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

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.

4.2.5. 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.

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 :

$$\text{DIRE}(C, \mathit{Exp}) = C \sqcap \mathit{Exp}.$$

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

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

pour toute interprétation $\mathcal{I}$. Donc, $C \equiv \text{DIRE}(C, \mathit{Exp})$ car

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

pour toute interprétation $\mathcal{I}$. Cela contredit le principe (P4, *succès*).
Afin d'éviter cette incompatibilité, un ajout de l'*opération épistémique* est envisagé [**DLN$^+$, 1998**]. Nous désignons par W_1 l'ensemble des interprétations de la BC avant la révision. Pour respecter le principe (P4), la relation de subsomption $C \sqsubseteq \text{DIRE}(C, \mathit{Exp})$ 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 **ProRésAuFeu** peut être défini comme suit :

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

A partir de cette définition du concept, on peut instancier de **ProRésAuFeu** 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(**ProRésAuFeu**, "résister à la température du 900∘C au 1000∘C"),

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

4.2.6. Approche syntaxique. Nous avons examiné plusieurs approches du problème de révision d'une BCT : l'approche au niveau de symbole offre des opérations de révision sur la BCT au niveau syntaxique, et l'approche sémantique considère une BCT comme l'ensemble des relations de subsomption possibles et modifiables. Dans l'approche sémantique, 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.
Par contre, l'approche syntaxique peut causer des modifications inutiles de la BC. En outre, il est difficile d'avoir des idées sur la signification des opérations de révision car elles manipulent syntaxiquement des définitions de termes. Si nous admettons 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"* [**Neb, 1990a**]

,nous aurons une issue à ce dilemme. En effet, 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 [**Neb, 1990a**], 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 du concept.
- OUBLIER. Cette opération est définie comme l'enlèvement d'un *concept simple* de l'ensemble des concepts simples déterminant la définition du concept.

Plus formellement,

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

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. Cette hypothèse nous permet de remettre le problème de révision dans le cadre de la logique du premier ordre.

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 (P4, *succès*) 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 (P5, *modification minimale*). Quant à l'opération OUBLIER, elle ne respecte pas le principe (P4, *succès*). 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 existe encore pour les opérations DIRE et OUBLIER par rapport aux principes (P4, *succès*) et (P5, *modification minimale*). 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 [**Neb, 1990a**] :

EXAMPLE 4.3.1. (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**

Selon la définition de concepts simple, on a

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

Il est évident que

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

ne satisfait pas le principe (P4, *succès*) 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

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

ne satisfait pas le principe (P5, *modification minimale*) car le concept simple "au moins 2 *membres*" est inutilement ajouté à S(**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 [**Neb, 1990a**]. 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 :

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

alors la révision

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

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

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

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 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 tenue 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 reste encore là 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 :

DEFINITION 4.3.2. Une $\mathcal{FLE}$-description de concept est appelée *simple* si elle est normalisée selon les règles de normalisation dans la section 2.2.4 et 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.
Nous désignons par $\mathcal{S}_C$ l'ensemble des concepts simples d'un concept C dans une terminologie.

La proposition suivante assure que l'ensemble des concepts simples est uniquement déterminé par la description de concept qui se compose de ces concepts simples.

PROPOSITION 4.3.3. *Soient C, D $\mathcal{FLE}$-descriptions de concept équivalentes dans toute terminologie. On a : $\mathcal{S}_C$=$\mathcal{S}_D$ i.e pour chaque $s \in \mathcal{S}_C$, il existe $s' \in \mathcal{S}_D$ tel que $s \equiv s'$ et pour chaque $s' \in \mathcal{S}_D$, il existe $s \in \mathcal{S}_C$ tel que $s' \equiv s$.*

DÉMONSTRATION. Puisque C et D sont équivalentes dans toute terminologie, ils doivent être construits des mêmes ensembles des noms de concepts et de rôles. Puisque $C \sqsubseteq D$, pour chaque $s' \in \mathcal{S}_D$, il existe $s \in \mathcal{S}_C$ tel que $s \sqsubseteq s'$. De même, puisque $D \sqsubseteq C$, pour chaque $s \in \mathcal{S}_C$, il existe $s'' \in \mathcal{S}_D$ tel que $s'' \sqsubseteq s$. Par conséquent, si $s' \neq s''$alors $s' \sqsubseteq s''$. Cela contredit la définition de $\mathcal{S}_D$. Cela implique que $s' = s$. De la même manière, on peut démontrer que pour chaque $s \in \mathcal{S}_C$, il existe $s' \in \mathcal{S}_D$ tel que $s = s'$. □

NOTE 4.3.4. 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, si la référence par sens est acceptée dans le système de représentation *i.e* la description de concept du côté droit de chaque définition dans un TBox impose la condition *nécessaire et suffisante* au sens du concept défini du côté gauche, alors

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.

Notons que, en général, les opérations de révision et d'expansion (l'opération d'expansion fait seulement l'ajout au TBox) d'un TBox sont équivalentes dans le système de représentation où la référence par sens est utilisée. Pourtant, le dépliage limité de TBox permettant de préserver la plupart de dépendances entre les concepts fait la différence entre les deux opérations.

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 $Exp \notin \mathcal{S}_C$. 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 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 (P4, *succès*) 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

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

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

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

Cela signifie que le principe (P5, *modification minimale*) 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 qui sont construites dans cette section prennent en compte les trois aspects suivants :

(1) Le mécanisme de la référence par nom affaibli est appliqué. C'est-à-dire que chaque introduction d'un concept défini dans le TBox est considérée comme une définition *i.e* le côté droit d'une introduction de concept impose la condition nécessaire et suffisante au sens du concept défini. D'autre part, les opérations de révision modifient le sens d'un concept en manipulant les descriptions de concept en ajoutant ou en enlevant un concept simple, qui est défini par la Définition 4.3.2.

(2) Les opérations de révision devraient assurer la perturbation minimale du TBox suivant une révision d'un concept (P5, *modification minimale*).

(3) Le succès des opérations de révision devraient être assuré (P4, *succès*).

Plus précisément,

(1) Pour l'opération OUBLIER, le principe (P4, *succès*) exige que OUBLIER(C, *Exp*) $\not\sqsubseteq$ *Exp*. Par ailleurs, le principe (P5, *modification minimale*) peut être interprété de façon suivante : OUBLIER(C, *Exp*) doit être une $\mathcal{FLE}$-description de concept la plus petite par rapport à la relation de subsomption telle qu'elle subsume C. De plus, les postulats (K2, inclusion), (K3, vacuité) et (K6, récupération) concernent le principe (P5). Cependant, (K2) et (K3) ne s'applique pas à la base de connaissances terminologique. Le postulat (K6) donne une borne supérieure à l'opération OUBLIER. Ce postulat sera considéré avec l'opération DIRE.

(2) Pour l'opération DIRE, le principe (P4, *succès*) exige que DIRE(C, *Exp*) $\sqsubseteq C$. De plus, l'opération DIRE avec l'opération OUBLIER doit satisfaire (K6, récupération).

(3) Quant au principe de la référence par nom, les opérations de révision ne manipulent plus simplement 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 DL permettant de définir la notion de concept simple. Cependant, cette dernière devient moins importante lorsque le mécanisme de la référence par nom avec le sens fort ne domine plus la conception des systèmes de représentation. Nous avons besoin de la définition de l'opération SEMI-OUBLIER suivante pour la raison technique.

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.

Definition 4.3.5. (Opération 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 $\mathcal{FLE}$-description de concept telle que $C \sqsubseteq D$ et $D \not\sqsubseteq Exp$, alors $D \not\sqsubseteq$ SEMI-OUBLIER(C, Exp).

Nous introduisons également une définition plus forte pour l'opération OUBLIER. La différence unique entre les deux définitions est que la condition 3. est renforcée dans la définition OUBLIER.

Definition 4.3.6. (Opération 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 $\mathcal{EL}$-description de concept telle que $C \sqsubseteq D$ et $D \not\sqsubseteq Exp$, alors OUBLIER(C, Exp) $\sqsubseteq D$.

Remark 4.3.7. 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 OUBLIER(C, Exp) défini ne peut pas exister, car $\bot \sqsubseteq D$ pour tout D.

L'exemple suivant illustre les définitions ci-dessus.

Example 4.3.8.

(1) Soient
$C := \forall r.(A \sqcap B \sqcap C)$, $Exp := \forall r.(B \sqcap C)$
On a : $C \sqsubseteq Exp$, et donc, SEMI-OUBLIER(C, Exp) $= \{\forall r.(A \sqcap B)\}$.
Effectivement, on obtient : $C \sqsubseteq \forall r.(A \sqcap B)$ et $\forall r.(A \sqcap B) \not\sqsubseteq \forall r.(B \sqcap C)$.
De plus, si $C \sqsubseteq C'$ et $C' \not\sqsubseteq Exp$, alors
$C' \in \{\top, \forall r.(A \sqcap B), \forall r.(A \sqcap C), \forall r.A, \forall r.B, \forall r.C\}$ et
$\top, \forall r.(A \sqcap C), \forall r.A, \forall r.B, \forall r.C \not\sqsubseteq \forall r.(A \sqcap B)$

(2) Soient
$C := \forall r.(A \sqcap B) \sqcap \exists r.(A \sqcap B)$, $Exp := \exists r.A$
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, si $C \sqsubseteq C'$ et $C' \not\sqsubseteq Exp$, alors $\forall r.B \sqcap \exists r.B \sqsubseteq C'$.

Remark 4.3.9. Nous montrons que les opérations décrites ci-dessus sont bien définies pour certains langages et considérons quelques cas particuliers.

(1) Il est évident que pour un langage DL $\mathcal{L}$, si 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 $D \not\sqsupset$ OUBLIER(C, *Exp*).

(2) Soient C, *Exp* des $\mathcal{L}$-descriptions de concept normalisées, *Exp* est une description de concept simple. 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, donc $C_1 \sqcap C_2 \not\sqsubseteq$ *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 :
OUBLIER(C_1, *Exp*) $\sqsubseteq$ OUBLIER(C_2, *Exp*)
En effet, puisque $C_1 \sqsubseteq C_2 \sqsubseteq$ OUBLIER(C_2, *Exp*) et OUBLIER(C_2, *Exp*) $\not\sqsubseteq$ *Exp*, alors par définition de l'opération OUBLIER, on a :
OUBLIER(C_1, *Exp*) $\sqsubseteq$ OUBLIER(C_2, *Exp*).

(4) Pour le langage $\mathcal{FLE}$, SEMI-OUBLIER(C, *Exp*) peut ne pas être unique. Donc, OUBLIER(C, *Exp*) peut ne pas exister. Par exemple, soient
$C := \forall r.(A_1 \sqcap A_2 \sqcap A_3)$,
$Exp := \forall r.(A_2 \sqcap A_3)$
Si SEMI-OUBLIER(*C,Exp*) $\in \{ \forall r.(A_1 \sqcap A_2), \forall r.(A_1 \sqcap A_3)\}$, alors $C \sqsubseteq$ SEMI-OUBLIER(*C,Exp*), SEMI-OUBLIER(*C,Exp*) $\not\sqsubseteq$ *Exp* et si une description de concept E telle que $C \sqsubseteq E$, $E \not\sqsubseteq$ *Exp*, alors $E \not\sqsupset$ SEMI-OUBLIER(*C,Exp*). 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 les SEMI-OUBLIER du concept C, alors $C \sqsubseteq C_1$, $C \sqsubseteq C_2$, $C_1 \not\sqsubseteq$ *Exp*, $C_2 \not\sqsubseteq$ *Exp*, et donc $C \sqsubseteq C_1 \sqcap C_2$. Puisque C est une restriction existentielle, alors C_1, C_2 sont également les restrictions existentielles (sous la forme irréductible) et donc $C_1 \sqcap C_2 \not\sqsubseteq$ *Exp*. Par définition de l'opération OUBLIER, on obtient : $C_1 \sqcap C_2 \not\sqsubseteq C_1$. 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*).

Nous nous rendons compte que les définitions de l'opération OUBLIER ne nous permet pas de calculer une description de concept OUBLIER. Dans ce cas, l'approche pour l'algorithme de construction montre à nouveau son importance. Par la suite, nous montrons que OUBLIER existe toujours pour le langage $\mathcal{EL}$ et présentons un

algorithme qui calcule la description de concept OUBLIER pour le langage $\mathcal{EL}$.

Sans perte de la généralité, supposons que l'ensemble de rôles N_R des langages L.D $\mathcal{EL}$ et $\mathcal{FLE}$ 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.

ALGORITHM 4.3.10. *(MINUS pour $\mathcal{EL}$) Soient* C, D *des $\mathcal{EL}$-descriptions de concept normalisées,* D *est une description de concept simple, irréductible et* $C \sqsubseteq D$. *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 : $(\mathcal{G} - \mathcal{H}) = \mathcal{G}_{MINUS(C,D)}$

(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'abre de* $(\mathcal{G} - \mathcal{H})$. *Sinon,*

(2) *Si* w_0 *n'a pas de successeur,* $(\mathcal{G} - \mathcal{H})$ *est une copie de* $\mathcal{G}$ *mais 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* $l_D(w) \nsubseteq l_C(v)$, *on obtient un* r*-successeur* (v, w) *de* (v_0, w_0), $l(v, w) = l_C(v)$ *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 l'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), $l(v, w) = l_C(v) \setminus \{P\}$ *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), $l(v, w) = l_C(v)$ *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 1. de l'algorithme et on obtient les* r*-successeurs* (v, w) *correspondant de* (v_0, w_0) *dans* $(\mathcal{G} - \mathcal{H})$.

EXAMPLE. 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 4.3.10, 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 4.3.10, 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*) = $\exists r.B \sqcap \exists r.C$.

REMARK 4.3.11. Selon l'Algorithme 4.3.10, la taille de MINUS(*C,D*) est un produit des tailles de C et de D.

Maintenant, nous formulons et montrons la proposition suivante qui établit l'équivalence entre l'opération OUBLIER définie dans la Définition 4.3.6 et la description de concept calculée par l'Algorithme 4.3.10.

PROPOSITION 4.3.12. *Soient* C, D des $\mathcal{EL}$*-descriptions de concept normalisées et* D *est une description de concept simple et irréductible. De plus, supposons que* $C \sqsubseteq D$. *L'opération OUBLIER peut être calculée comme suit :*

$$OUBLIER(C,\ D) \equiv MINUS(C,\ D)$$

DÉMONSTRATION. Nous démontrerons 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 $(\mathcal{G} - \mathcal{H}) = G_{MINUS(C,D)} = (N_{C-D},\ E_{C-D},\ u_0,\ l_{C-D})$.

- $|C| = 0$. 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 : $|D| = |C| = 0$ 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 4.3.10, on a : $l_{D-C}(u_0) \subseteq l_C(v_0)$.
  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 4.3.10, alors $l_{D-C}(v, w) = l_C(v)$ 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 4.3.10, alors $l_{D-C}(v, w) \subseteq l_C(v)$ 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 4.3.10, 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 4.3.10, 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 4.3.10, alors $l_{C-D}(v, w) = l_D(v) \not\sqsubseteq l_C(w)$ 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 4.3.10, alors $l_{C-D}(v, w) \not\sqsubseteq l_D(w)$ 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 4.3.10, 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 4.3.10, 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) $\not\sqsubseteq D$.

(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 4.3.10, 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 4.3.10, 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) $\sqsubseteq A$.

□

La Remarque 4.3.9 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 nous sommes capables de déterminer l'ensemble de *toutes* les descriptions de concept SEMI-OUBLIER(*C, Exp*), alors OUBLIER(*C, Exp*) peut être approximée en quelque sorte par cet ensemble. Pour cet objectif, nous commençons par proposer une procédure qui permet de calculer l'ensemble {SEMI-OUBLIER(*C, Exp*)} où C est une $\mathcal{FLE}$-description de concept simple (*Exp* reste toujours une $\mathcal{FLE}$-description de concept simple).

ALGORITHM 4.3.13. *(MINUS pour les concepts simples) Soient* C, D *des $\mathcal{FLE}$-descriptions de concept normalisées et C est une description de concept simple i.e $C = \alpha.C'$ 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, irréductible et* $C \sqsubseteq D$. *La description de concept MINUS(C,D) est calculée 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) et $\mathcal{H}$ =(N_D, E_D, w_0, l_D). L'ensemble des description de concepts MINUS(C, D) est calculé par récurrence sur la profondeur de $\mathcal{G}$ comme suit :*

Entrée : $\mathcal{G}$, $\mathcal{H}$
Sortie : Ensemble des arbres $T = \{\mathcal{G} - \mathcal{H}\} = \{\mathcal{G}_{C'} | \ C' \in MINUS(C, D)\}$

(1) *$T := \emptyset$. Soient v, w les r-successeurs respectivement de v_0 et de w_0.*

(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 $|l_C(v_0)| = |l_D(w_0)| = 1$. Sinon,*

(3) *Si w_0 n'a pas de successeur, $(\mathcal{G} - \mathcal{H})$ comporte un seul arbre qui est une copie de $\mathcal{G}$ mais le nœud (v_0, w_0) étiqueté par $l_C(v_0) \setminus l_D(w_0)$.*

(4) *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, $l_t(t_0) = l_C(v_0) \setminus l_D(w_0)$. De plus, (v, w) qui est un α-successeur de t_0, $l_t(v, w) = l_C(v) \setminus \{P\}$, est la racine d'une copie du sous-arbre $\mathcal{G}(v)$ de $\mathcal{G}$. On a : $T := T \cup \{t\}$.*

(5) *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) qui est un α-successeur de t_0, $l_t(v, w) = l_C(v)$, est la racine de l'arbre t', et les copies des sous-arbres $r.\mathcal{G}(v'')$ où v'' est un r-successeur de v.*
On a : $T := T \cup \{t\}$.

(6) *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, un arbre t est créé où t_0 est la racine de t tel que (v, w) qui est un α-successeur de t_0, $l_{C-D}(v, w) = l_C(v)$, est la racine des sous-arbres :

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

   (b) *$r.F^{i_1}_{v^1}, ..., r.F^{i_m}_{v^m}$ où $(i_1, ..., i_m) \in \{1..k_1\} \times ... \times \{1..k_m\}$ et $(r.F^{i_1}_{v^1}, ..., r.F^{i_m}_{v^m}) \in (T_{v^1} \times ... \times T_{v^m})$*

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

On a $T := T \cup \{t\}$. Notons que pour chaque r-successeur w'' de w, chaque ensemble V et chaque $(i_1, ..., i_m) \in \{1..k_1\} \times ... \times \{1..k_m\}$, 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.

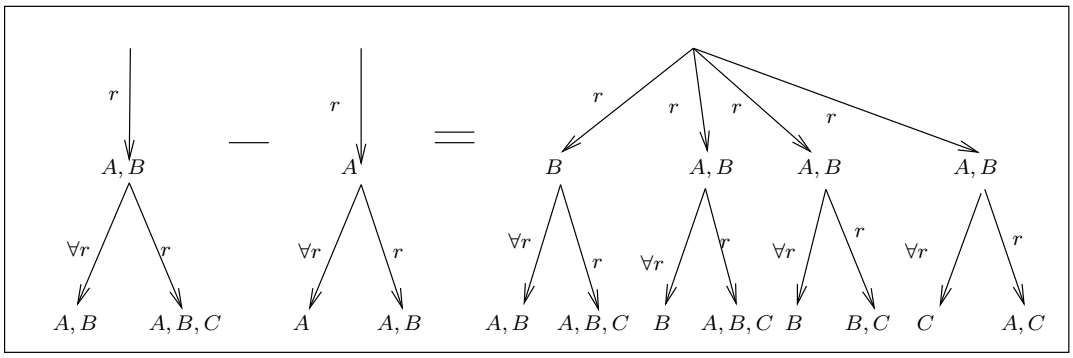

FIG. 4.3.1. MINUS pour les Concepts Simples

EXAMPLE 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 4. 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 5. de l'algorithme, on obtient :
    $M_2 = \exists r.(A \sqcap B \sqcap \forall r.B \sqcap \exists r.(A \sqcap B \sqcap C))$
  – Par l'étape 6. de l'algorithme, on obtient :
    $M_3 = \exists r.(A \sqcap B \sqcap \forall r.B \sqcap \exists r.(B \sqcap C))$
  – Par l'étape 6. de l'algorithme, on obtient :
    $M_4 = \exists r.(A \sqcap B \sqcap \forall r.C \sqcap \exists r.(A \sqcap C))$
  De plus, on a également OUBLIER(C,Exp) $= M_1 \sqcap M_2 \sqcap M_3 \sqcap M_4$
– Soient
$C := \forall r.(A \sqcap B \sqcap \forall r.A \sqcap \exists r.(A \sqcap B \sqcap C))$,
$Exp := \forall r.(A \sqcap \forall r.A \sqcap \exists r.(A \sqcap B))$.
Par l'algorithme avec 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, $C \equiv M'_1 \sqcap M'_2 \sqcap M'_3 \sqcap M'_4 \sqsubseteq Exp$.

REMARK 4.3.15. Selon l'Algorithme 4.3.13, le nombre des éléments de l'ensemble MINUS(C,D) est borné par une fonction exponentielle en tailles de C et de D. // ? 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 et irréductible. De plus, supposons que* C $\sqsubseteq$ D. *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* $t \in \{\mathcal{G}_{C'} \mid C' \in \{MINUS(C, D)\}\}$, $C_t \sqsubseteq A$.

En particulier,

(1) *si* $\alpha = r$ *i.e* $C = r.C'$ *où* C' *est une* $\mathcal{FLE}$*-description de concept, l'opérateur 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. 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 $t \in \{\mathcal{G}_{C'} | C' \in \text{MINUS}(C, D)\}$, $t = (N_{C-D}, E_{C-D}, t_0, l_{C-D})$.

– $|C| = 0$. 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) $C \sqsubseteq C_t$, $\{\mathcal{G}_{C'} | C' \in \text{MINUS}(C, D)\} = \{t\}$. 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 $A \sqsubseteq C_t$, $t \in \{\mathcal{G}_{C'} | C' \in \text{MINUS}(C, D)\}$. 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 $t \in \{\mathcal{G}_{C'} | C' \in \text{MINUS}(C, D)\}$. 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 $t \in \{\mathcal{G}_{C'} | C' \in \text{MINUS}(C, D)\}$ est créé par l'étape 4. de l'Algorithme 4.3.13, alors il est évident que $C \sqsubseteq C_t$.

(b) Si $t \in \{\mathcal{G}_{C'} | C' \in \text{MINUS}(C, D)\}$ est créé par l'étape 5. 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 6. 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.(b), 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 4. 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 5. 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 6. 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 $\mathcal{F} = (N_A,\ E_A,\ x_0,\ l_A)$, (v, w) le successeur de t_0 où v, w sont les successeurs de v_0 et de w_0. Puisque $C \sqsubseteq A$, il existe uniquement un α-successeur x de x_0, $l_A(x_0) \subseteq l_C(v_0) = l(t_0)$.

(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 4. 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 5. 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 6. 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 $l_D(w'') \not\subseteq l_A(v'')$, *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 4.3.5, cela implique que $C_t \equiv S$.

□

A partir de l'algorithme pour calculer l'opérateur MINUS d'une $\mathcal{FLE}$-description de concept simples, nous pouvons construire un algorithme qui permet de calculer MINUS d'une $\mathcal{FLE}$-description de concept arbitraire.

ALGORITHM 4.3.17. *(MINUS) Soient* C, D *des $\mathcal{FLE}$-descriptions de concept normalisées et* D *est une description de concept simple, irréductible et* C $\sqsubseteq$ D. *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) et $\mathcal{H}$ =(N_D, E_D,*

w_0, l_D). L'ensemble des description de concepts MINUS(C, D) est calculé par récurrence sur la profondeur de $\mathcal{G}$ comme suit.

Entrée : $\mathcal{G}$, $\mathcal{H}$
Sortie : Ensemble des arbres $T = \{\mathcal{G} - \mathcal{H}\} = \{\mathcal{G}_{C'} | \ C' \in MINUS(C, D)\}$

(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 $|l_C(v_0)| = |l_D(w_0)| = 1$. Sinon,*

(3) *Si w_0 n'a pas de successeur, $(\mathcal{G} - \mathcal{H})$ est une copie de $\mathcal{G}$ mais 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 $\mathcal{G}$.*

   (a) *Au premier niveau de l'arbre,*
   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 dont la racine t_0 est la racine de t', $l_t(t_0) = l_C(v_0)$ et des copies des sous-arbres $r.\mathcal{G}(v)$ pour tout r-successeur v de v_0. On a : $T := T \cup \{t\}$.

   (b) *A partir du second niveau de l'arbre, cette étape est remplacée par l'Algorithme 4.3.13.*

(5) *Si le successeur w de w_0 est un r-successeur dans $\mathcal{H}$, alors on crée un ensemble vide des arbres T'.*

   (a) *Si v est le successeur unique de v_0, on suppose que $l_D(w) \subseteq l_C(v)$. 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'', $l_{t'}(t'_0) = l_C(v_0)$. On a : $T' := T' \cup \{t'\}$.*

   (b) *S'il existe plusieurs successeurs v de v_0 (cette étape est exigée au premier niveau de l'arbre),*
   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'', $l_{t'}(t'_0) = l_C(v_0)$. De plus, pour chaque r-successeur v de v_0 dans $\mathcal{G}$ où $l_D(w) \not\subseteq l_C(v)$, 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) *Pour chaque sous-ensemble des arbres $V = \{t^1, ..., t^m\}$ de T', on obtient un arbre t dont la racine t_0, $l_t(t_0) = l_{t^1}(t^1_0) \cup ... \cup l_{t^m}(t^m_0)$ 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 : $T' := T' \cup \{t\}$.

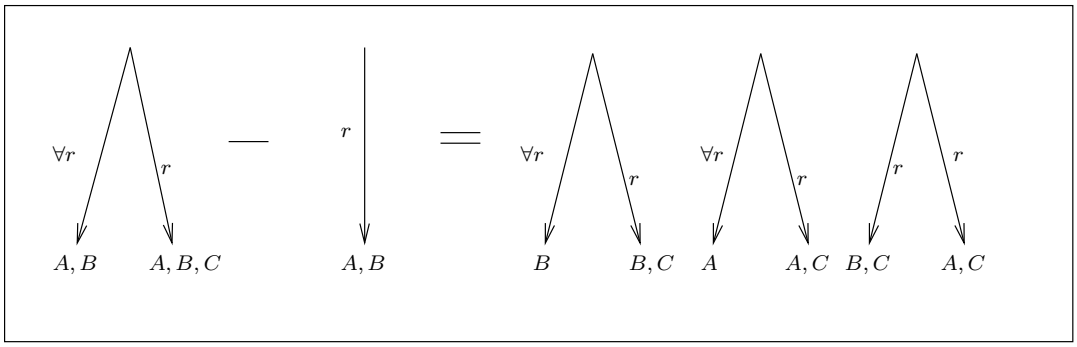

FIG. 4.3.2. MINUS où Exp est une restriction existentielle

La figure 4.3.2 illustre le calcul de l'Exemple 4.3.18 par l'Algorithme 4.3.17.

EXAMPLE 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 correction de l'Algorithme 4.3.17.

THEOREM 4.3.19. *Soient* C, D des $\mathcal{FLE}$*-descriptions de concept normalisées et* D *est une description de concept simple et irréductible. De plus, supposons que* C $\sqsubseteq$ D. *Alors, l'ensemble SEMI-OUBLIER(C,D) peut être calculé par l'algorithme MINUS :*

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

En particulier, si C ne contient pas de restriction universelle au top-level, l'opération OUBLIER peut être calculée comme suit :

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

DÉMONSTRATION. 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 $t = (N_t, E_t, u_0, l_t)$ 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) $C \sqsubseteq \text{MINUS}(C, D)$. 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) $\text{MINUS}(C, D) \not\sqsubseteq D$. 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 $A \sqsubseteq \text{MINUS}(C, D)$. 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 $t \in \{\mathcal{G}_{C'} | \ C' \in \text{MINUS}(C, D)\}$. 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), $\forall r.t(v, w) \in \{\mathcal{G}_{C'} | \ C' \in \text{MINUS}(C_{\forall r.\mathcal{G}(v)}, C_{\forall r.\mathcal{H}(w)})\}$ 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 $t \in \{\mathcal{G}_{C'} | \ C' \in \text{MINUS}(C_{r.\mathcal{G}(v)}, C_{r.\mathcal{H}(w)})\}$ 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*,
$C_{\mathcal{G}(v')} \sqsubseteq lcs\{\{C_{t(v,w)} \mid \text{pour tout } r\text{-successeur } (v, w) \text{ de } t_0\} \cup \{C_{\mathcal{G}(v')} \mid v' : \forall r\text{-successeur de } v_0\}\}$. Par conséquent, $C \sqsubseteq \text{MINUS}(C,D)$.

(2) $C_t \not\sqsubseteq D$ où $t \in \{\mathcal{G}_{C'} | \ C' \in \text{MINUS}(C, D)\}$

(a) Si le successeur w de w_0 est un $\forall r$-successeur, t a un $\forall r$-successeur (v, w) où $\forall r.t(v, w) \in \{\mathcal{G}_{C'} | \ C' \in \text{MINUS}(C_{\forall r.\mathcal{G}(v)}, C_{\forall r.\mathcal{H}(w)})\}$, 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 que 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 au restriction universelle. Par conséquent, on obtient : MINUS(C,D) $\sqsubseteq A$.

Maintenant, on démontre que si $S \in$\{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 4.3.5, cela implique que $C_t \equiv S$.

En particulier, puisque C se compose des restrictions existentielles au top-level, alors par l'Algorithme 4.3.17, 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 a : $C \sqsubseteq \bigsqcap_{t \in \{\mathcal{G}_{C'} | C' \in MINUS(C,D)\}} \{C_t\}$ et $\bigsqcap_{t \in \{\mathcal{G}_{C'} | C' \in MINUS(C,D)\}} \{C_t\} \not\sqsubseteq D$. Par conséquent, on obtient *cqfd.*

□

Maintenant, nous essayons de définir l'opérateur 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, soient les $\mathcal{EL}$-descriptions de concept :

$$C := \exists r.(A \sqcap B) \sqcap \exists r.(A \sqcap C)$$

$$Exp := \exists r.A$$

Selon l'Algorithme 4.3.10, 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 grande 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

$$\text{DIRE}(\text{OUBLIER}(C,\ Exp),Exp) = \exists r.B \sqcap \exists r.C \sqcap \exists r.A,$$

et donc

$$C \sqsubset \text{DIRE}(\text{OUBLIER}(C,\ Exp),Exp),$$

qui ne satisfait pas le postulat de récupération (K6).
Pour faciliter la lecture, nous commençons par un algorithme, appelé PLUS pour le langage $\mathcal{EL}$, qui permet de calculer une combinaison entre une $\mathcal{EL}$-description de concept et un $\mathcal{EL}$-concept simple. L'opération DIRE sera directement défini via l'opérateur PLUS.

ALGORITHM 4.3.20. *(Opérateur* PLUS *pour $\mathcal{EL}$) Soient* C, D *$\mathcal{EL}$-descriptions de concept normalisées* et D *est une description de concept simple, irréductible et* C $\not\sqsubseteq$ D. *L'opération PLUS(C, D) est défini comme l'arbre de description obtenu d'une fusion des deux arbres $\mathcal{G}_\mathcal{C}$ et $\mathcal{G}_\mathcal{D}$. Plus formellement, soit $\mathcal{G}$ =(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 : PLUS(C, D)

- *Si $|C| = 0$ ou $|D| = 0$ alors PLUS(C, D) := $C \sqcap D$.*
- *Sinon, $(\mathcal{G} + \mathcal{H})$ est défini par récurrence sur la profondeur des arbres. Le nœud u_0 étiqueté par $l_G(v_0) \cup l_H(w_0)$ est la racine de $(\mathcal{G} + \mathcal{H})$.*
  - *Pour chaque $r-$successeur v de v_0 dans $\mathcal{G}$ et chaque successeur w de w_0 dans $\mathcal{H}$, on obtient un $r-$successeur u de u_0 dans $(\mathcal{G} + \mathcal{H})$ où $l_{C+D}(u) := l_C(v) \cup l_D(w)$. Le nœud u est la racine du sous-arbre $\mathcal{G}(v) + \mathcal{H}(w)$. S'il n'existe pas de $r-$successeur v de v_0 dans $\mathcal{G}$, alors on ajoute à v_0 des copies des sous-arbres $r.\mathcal{H}(w)$ pour tout r-successeur w de w_0.*

A partir de l'algorithme de l'opérateur PLUS ci-dessus, nous pouvons proposer un algorithme étendu pour le langage $\mathcal{FLE}$ en prenant en compte les restrictions universelles.

ALGORITHM 4.3.21. *(Opérateur* PLUS*) Soient* C, D $\mathcal{FLE}$*-descriptions de concept normalisées et* D *est une description de concept simple, irréductible et* C $\not\sqsubseteq$ D. *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, soit* $\mathcal{G}$ *=(*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 : $(\mathcal{G}+\mathcal{H}) = \mathcal{G}_{PLUS(C,D)}$

(1) *Si* $|C| = 0$ *ou* $|D| = 0$ *, alors PLUS(C,D) :=* $C \sqcap D$.

(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) *Pour chaque* $r-$*successeur* v *de* v_0 *dans* $\mathcal{G}$ *et chaque successeur* w *de* w_0 *dans* $\mathcal{H}$*, on obtient un* $r-$*successeur* (v, w) *de* u_0 *dans* $(\mathcal{G}+\mathcal{H})$ *où* $l_{C+D}(v, w) := l_C(v) \cup l_D(w)$*. Le nœud* (v, w) *est la racine des sous-arbres* $\mathcal{G}(v) + \alpha.\mathcal{H}(w')$ *où* w' *est un* $\alpha-$*successeur de* w*,* $\alpha \in \{r, \forall r\}$*. S'il n'existe pas de* $r-$*successeur* v *de* v_0 *dans* $\mathcal{G}$*, alors on ajoute à* (v_0, w_0) *des copies des sous-arbres* $r.\mathcal{H}(w)$ *pour tout* r*-successeur* w *de* w_0*. S'il n'existe pas de* $r-$*successeur* w *de* w_0 *dans* $\mathcal{H}$*, alors on ajoute à* (v_0, w_0) *des copies des sous-arbres correspondant aux descriptions de concept* $\exists r.(C_{\mathcal{G}(v)} \sqcap C')$ *pour tout* r*-successeur* v *de* v_0 *où* C' *est obtenue de l'étape (b).ii.*

   (b) *Le* $\forall r$*-successeur* u' *de* u_0 *dans* $(\mathcal{G}+\mathcal{H})$ *est la racine du sous-arbre de la description de concept :*

      (i) $lcs\{C_{(\mathcal{G}+\mathcal{H})(u)} \mid$ pout tout r*-successeur* u *de* $u_0\}$ *s'il n'existe pas de* $\forall r$*-successeur de* w_0*. Sinon,*

      (ii) *le* $\forall r$*-successeur* u' *de* u_0 *dans* $(\mathcal{G}+\mathcal{H})$ *est la racine du sous-arbre de la description de concept :* $C' = C_{(\mathcal{G})(v')} \sqcap C_{(\mathcal{H})(w')}$ *où* v', w' *sont les* $\forall r$*-successeurs de* v_0 *et de* w_0.

REMARK 4.3.22. Le calcul effectué par l'Algorithme 4.3.21 nous permet d'obtenir une description de concept qui n'est pas la "meilleure" *i.e* PLUS(C,D) n'est pas la description de concept la plus grande telle qu'elle est subsumée par C et par D.

DEFINITION 4.3.23. (Opération DIRE) Soient C, D $\mathcal{FLE}$-descriptions de concept normalisées, D une description de concept simple, irréductible.

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

(2) Si $C \sqsubseteq$ *Exp*, DIRE(C, *Exp*) := C,

(3) Si $C \not\sqsubseteq$ *Exp*, alors DIRE(C, *Exp*) := PLUS(C,*Exp*)

La proposition suivante affirme que les opérations de révision définies répondent aux propriétés attendues.

PROPOSITION 4.3.24. *Soient* C, D *des* $\mathcal{FLE}$*-descriptions de concept normalisées,* D *une description de concept simple, irréductible. Les opérations de révision DIRE et OUBLIER ont les propriétés suivantes :*

(1) *$D \sqsubseteq$ OUBLIER(C, D) et OUBLIER(C, D) $\not\sqsubseteq D$ (succès pour OUBLIER)*

(2) *DIRE(C, D) $\sqsubseteq C$ et DIRE(C, D) $\sqsubseteq D$ (succès pour DIRE)*

(3) *DIRE(OUBLIER(C, D), D) $\sqsubseteq C$ (récupération)*

DÉMONSTRATION. La preuve de 1. est directe

(1) Par la définition de l'opération OUBLIER.

(2) 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) $= C \sqsubseteq D$. Supposons que $C \not\sqsubseteq D$. S'il n'existe pas de $\forall r$-successeur de v_0 et de w_0, par hypothèse de récurrence, on a : $l_{G+H}(v, w) = l_G(v) \cup l_H(w)$, PLUS($C_{\mathcal{G}(v)}$, $C_{\alpha.\mathcal{H}(w')}$) $\sqsubseteq C_{\mathcal{G}(v)}$ et PLUS($C_{\mathcal{G}(v)}$, $C_{\alpha.\mathcal{H}(w')}$) $\sqsubseteq C_{\alpha.\mathcal{H}(w')}$ pour tout α-successeur w' de w. Donc, $C_{(\mathcal{G}+\mathcal{H})(v,w)} = \bigsqcap_{w'}$ PLUS($C_{\mathcal{G}(v)}$, $C_{\alpha.\mathcal{H}(w')}$) $\sqsubseteq C_{\mathcal{G}(v)}$ et $C_{(\mathcal{G}+\mathcal{H})(v,w)}$ $= \bigsqcap_{w'}$ PLUS($C_{\mathcal{G}(v)}$, $C_{\alpha.\mathcal{H}(w')}$) $\sqsubseteq C_{\mathcal{H}(w)}$ pour tout r-successeur v de v_0 et w de w_0. De plus, on obtient également un $\forall r$-successeur (v', w') de (v_0, w_0) qui diminue encore PLUS(C, D) par rapport à C et à D.
S'il existe de $\forall r$-successeurs w et de w_0, d'après l'étape 2.b de l'Algorithme 4.3.21, il obtient un $\forall r$-successeur (v', w') de (v_0, w_0) tel que $C_{(\mathcal{G}+\mathcal{H})(v',w')} = C_{\mathcal{G}(v')} \sqcap C_{\mathcal{H}(w')} \sqsubseteq C_{\mathcal{G}(v')}$ et $C_{(\mathcal{G}+\mathcal{H})(v',w')} = C_{\mathcal{G}(v')} \sqcap C_{\mathcal{H}(w')} \sqsubseteq C_{\mathcal{H}(w')}$.
S'il n'existe pas de $\forall r$-successeurs w de w_0 et il existe le $\forall r$-successeurs v de v_0, sans perte de la généralité on suppose que $C = \exists r.C_1 \sqcap \exists r.C_2 \sqcap \forall r.C_3 \sqcap P$, $D = \exists r.D'$ et DIRE(C,D) $= \exists r.E_1 \sqcap \exists r.E_2 \sqcap \forall r.E_3 \sqcap P$ où $C_1 \sqsubseteq C_3$, $C_2 \sqsubseteq C_3$. D'après l'étape 2.b de l'Algorithme 4.3.21, on a : pour tout $i \in \{1, 2\}$ $E_i \sqsubseteq C_j$, $j \in \{1, 2\}$ et $D' \sqsubseteq E_k$ $k \in \{1, 2\}$. Donc, $E_3 = lcs(E_1$, $E_2) \sqsubseteq C_3$. Par conséquent, on a : PLUS(C, D) $\sqsubseteq C$ et PLUS(C, D) $\sqsubseteq D$.

(3) On se rend compte que, par l'Algorithme 4.3.17, si $(v, w) \in N_{C-D}$, $v \in N_C$ et $l_{C-D}(v, w) \neq l_C(v)$, alors $l_{C-D}(v, w) = l_C(v) \setminus \{P\}$, $P \in l_C(v) \cap l_D(w)$. Donc, si $l_{C-D}(v, w) \neq \emptyset$, alors $l_{C-D}(v, w)$ reprend la valeur $l_C(v)$ après DIRE car s'il existe un chemin c de v_0 à v dans $\mathcal{G}$, alors il existe également le même chemin de w_0 à w dans $\mathcal{H}$. Si $l_{C-D}(v, w) = \emptyset$ et, peut-être, (v, w) est éliminé de l'arbre $(\mathcal{G} - \mathcal{H})$, l'opération DIRE permet de le récupérer par des copies des sous-arbres de $\mathcal{H}$.
Plus formellement, si $|D| = 0$, la proposition est évidente. Sinon, supposons que $\exists r.C'$ est une restriction existentielle de C et $D = \exists r.D'$, $C' \sqsubseteq D'$. Alors, selon l'Algorithme 4.3.17, le $\mathcal{G}_{OUBLIER(C,D)}$ comporte i) le sous-arbre $r.\mathcal{G}_{C'}$ où $C_{\mathcal{G}_{C'}} \sqcap P \equiv C'$, le primitif P appartient à $l(w)$ et w est le r-successeur de w_o, ii) les sous-arbres $r.\mathcal{G}_{C''}$ où $C'' =$ OUBLIER(C', $C_{\alpha.\mathcal{H}(w')}$), w' est un successeur de w, $\alpha \in \{r, \forall r\}$. Par l'étape 2.a de l'Algorithme 4.3.21, le $\mathcal{G}_{DIRE(OUBLIER(C,D),D)}$ comporte les sous-arbres $r.\mathcal{G}'$ où $l(u'_0) = l(v) \cup l(w)$, v est un r-successeur de v_0 , u'_0 est la racine du sous-arbre $\mathcal{G}'$ et $\mathcal{G}'$ se compose des sous-arbres $\mathcal{G}_{C'}(v)+\alpha.\mathcal{H}(w')$ et $\mathcal{G}_{C''}(v)+\alpha.\mathcal{H}(w')$ pour tout successeur w' de w. Par 1. de cette proposition et l'hypothèse de récurrence $C_{\mathcal{G}_{C''}+\alpha.\mathcal{H}(w')}$ $=$ DIRE(C'', $C_{\alpha.\mathcal{H}(w')}$) $=$ DIRE(OUBLIER(C', $C_{\alpha.\mathcal{H}(w')}$), $C_{\alpha.\mathcal{H}(w')}$) $\sqsubseteq C'$, on a : $C_{\mathcal{G}_{C'}(v)+\alpha.\mathcal{H}(w')} \sqsubseteq C_{\mathcal{G}_{C'}(v)}$ et $C_{\mathcal{G}_{C''}(v)+\alpha.\mathcal{H}(w')} \sqsubseteq C_{\mathcal{G}_{C'}(v)}$. Alors, on obtient : $C_{\mathcal{G}_{C'}(v)+\mathcal{H}(w)} \sqsubseteq C_{\mathcal{G}_{C'}(v)}$. Le même raisonnement permet d'obtenir : $C_{\mathcal{G}_{C'}(v)+\mathcal{H}(w)} \sqsubseteq C_{\mathcal{G}_{C'}(v)}$ pour tout r-successeur v de v_0.

D'autre part, supposons que $\forall r.C'$ est la restriction universelle de C. Selon l'étape 2.b de l'Algorithme 4.3.21, $C'' \sqsubseteq C'$ où $\forall r.C''$ est la restriction universelle retournée par l'Algorithme 4.3.21. Notons que à l'étape 2.b de l'Algorithme 4.3.21, C'' est calculé comme un $lcs\{X_i\}$ où chaque X_i est subsumé par une expression sous la restriction existentielle au top-level de C , comme démontré ci-dessus . Puisque C est normalisé, alors $lcs\{X_i\}$ est subsumé par l'expression sous la restriction universelle C' au top-level de C. Finalement, on obtient *cqfd.*

□

Le corollaire suivant peut être directement déduit des propositions.

COROLLARY 4.3.25. *Soient C une $\mathcal{FLE}$-description, C_2, C_3 des $\mathcal{FLE}$-descriptions de concept simple, qui ne contiennent pas de restriction universelle au top-level. De plus, supposons que $C_2 \not\sqsubseteq C_3$, $C_3 \not\sqsubseteq C_2$. On a :*

(1) *OUBLIER(DIRE(C, C_2), C_3) $\sqsubseteq C_2$*

(2) *OUBLIER(DIRE(OUBLIER(C, C_3), C_2), C_3) $\sqsubseteq C_2$*

DÉMONSTRATION. La preuve est directe.

(1) D'après la Proposition 4.3.24, on a : DIRE$(C, C_2) \sqsubseteq C_2$. Puisque C, C_2 ne contiennent pas de restriction universelle au top-level, alors DIRE(C, C_2) ne contient pas non plus de restriction universelle au top-level. Donc, par le Théorème 4.3.19, OUBLIER(DIRE(C, C_2), C_3) existe et puisque $C_2 \not\sqsubseteq C_3$, alors par la propriété de OUBLIER, OUBLIER(DIRE(C, C_2), C_3) $\sqsubseteq C_2$.

(2) Effectivement, puisque C, C_2, C_3 ne contiennent pas de restriction universelle au top-level, alors OUBLIER(C, C_2) et *OUBLIER(DIRE(OUBLIER(C, C_3), C_2), C_3)* existent. D'autre part, d'après la Proposition 4.3.24, on a : DIRE(OUBLIER(C, C_3), C_2) $\sqsubseteq C_2$. Puisque $C_2 \not\sqsubseteq C_3$, on obtient : OUBLIER(DIRE(OUBLIER(C, C_3), C_2), C_3) $\sqsubseteq C_2$ selon le Théorème 4.3.19 et la propriété de OUBLIER.

□

La condition qui exige l'absence de la restriction universelle au top-level assure l'existence de la description de concept OUBLIER. C'est-à-dire que nous ne considérons que les $\mathcal{FLE}$-descriptions de concept qui ne contiennent pas de restriction universelle au top-level.

REMARK 4.3.26. Selon la Remarque 4.3.22, l'opération PLUS définie dans cette section n'est pas optimale. Cependant, cette définition de l'opération PLUS avec l'opération OUBLIER définie de façon optimale dans la Définition 4.3.6 permet de satisfaire le postulat de récupération (K6). Effectivement, une définition optimale pour l'opération PLUS est impossible dans ce contexte. L'exemple suivant montre cette impossibilité :

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

Si DIRE(C, D) est définie comme $C \sqcap D$, alors on obtient :

DIRE(OUBLIER(C,D)) $= \exists r.B \sqcap \exists r.A$, et donc $C \sqsubset$ DIRE(OUBLIER(C,D))

qui ne satisfait pas le postulat de récupération (K6).

Cependant, la définition optimale l'opération OUBLIER comme dans Définition 4.3.6 joue un rôle très important dans le développement de l'inférence sur le formalisme hybride (voir les démonstrations des Propositions 4.5.4 et 4.5.11). Cela justifie notre choix de la définition de l'opération DIRE.

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 ce 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 individuel vérifie la définition du concept C , alors cet individuel 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*), doivent 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 individuel 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 le TBox et $\Delta_{\mathcal{A}}$ est l'ABox. Pour chaque assertion $C^I(d^I)$ où $d^I \in \mathcal{O}$, nous pouvons utiliser l'inférence de *vérification d'instance* [**?**] pour savoir si l'assertion $C^I(d^I)$ est vérifié *i.e* $\Delta \models (DIRE(C, Exp))^I(d^I)$.

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 un opérateur 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 : $\Delta = (\Delta_{\mathcal{T}}, \Delta_{\mathcal{A}}, \Delta_{\mathcal{R}})$.

EXAMPLE 4.4.1. (Formalisation de l'exemple d'introduction)
Soit $\Delta_{\mathcal{T}}$:
$\Delta_{\mathcal{T}} := \{$
Européen $\sqcap$ *Américain* $\sqsubseteq \bot$;
EntrepriseEurop := $\exists$*associer.Européen ;*
EntrepriseAmér := $\exists$*associer.Américain ;*
ProdRésAuPolluant := *Produit* $\sqcap$ $\exists$*résister.Polluant*
$\}$
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, *Americain*). Le concept *ProdRésAuPolluant* est défini comme un *Produit* capable de *résister* au *Polluant.*
Soient $\Delta_{\mathcal{A}}$ des faits :
$\Delta_{\mathcal{A}}$:=$\{$*vendu-par*(a,b), *acheté-par*(a,c), *résisterAuPolluant(a), EntrepriseEurop*(b), *EntrepriseAmér*(c), *Polluant*(*Feu*), *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 *:*
$\Delta_{\mathcal{R}} := \{$ r :
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*$(Z_2) \wedge$
résister$(X, V_1) \wedge$ *Polluant*$(V_1) \Rightarrow$ DIRE(*ProdRésPolluant*, ≥ 2 *résister*) $\}$
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.Européan* impliquent *associer*(b, b') et *Européen*(b') .
Les faits *EntrepriseAmér*(c) et *EntrepriseAmér* := $\exists$*associer.Américain* impliquent *associer*(c, c') et *Américain*(c').
Les faits *ProdRésAuPolluant*(a) et *ProdRésAuPolluant* := *Produit* $\sqcap$ $\exists$*résister.Polluant* impliquent *résister*(a, d') et *Polluant*(d'). 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. 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 $a^I \in \mathcal{O}$. Une interprétation 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 I est un modèle d'une assertion $C(d)$ dans $\Delta_{\mathcal{A}}$ si $d^I \in C^I$.
Une interprétation I est un modèle d'une règle $r : p \Rightarrow$ DIRE(C, Exp) si, lorsque la fonction $\cdot^{\mathcal{I}}$ transforme les individus $d^I \in C_i^I$ dans le domaine $\mathcal{O}$ pour chaque concept C_i de l'antécédent de la règle r, alors
– Si $a^I \in C^I$ alors $a^I \in (\text{DIRE}(C, Exp))^I$
Une interprétation I est un modèle de $r : p \Rightarrow$ OUBLIER(C, Exp) si, lorsque la fonction $\cdot^{\mathcal{I}}$ transforme les individus $d^I \in C_i^I$ dans le domaine $\mathcal{O}$ pour chaque concept C_i de l'antécédent de la règle r, alors
– Si $a^I \in C^I$ alors $a^I \in (\text{OUBLIER}(C, Exp))^I$

REMARK 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. L'extension procédurale de la base de connaissances Δ par l'application des règles du composant $\Delta_{\mathcal{R}}$ sera étudiée dans la Section 4.5.

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 un service d'inférence sur le formalisme hybride qui est défini dans la section précédente. Ce service nous permet d'exploiter les connaissances formalisées dans le composant $\Delta_{\mathcal{R}}$ pour obtenir l'harmonie entre les acteurs dans un contexte d'échanges de données. // (rôle et modification de $\Delta_{\mathcal{A}}$)

DEFINITION 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 $Cons(r)$.

REMARK 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 $\Delta_{\mathcal{R}}$.

DEFINITION 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 $Cons(r_{i_l})$.

REMARK. 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 sont causés par des applications de règles. Maintenant, nous formulons et montrons la condition de terminaison de l'inférence sur le formalisme hybride. La démonstration de la proposition suivante est subordonnée à l'existence des propriétés des opération de révision.

PROPOSITION 4.5.4. *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 :* $Ex(r_i) \not\sqsubseteq Ex(r_j)$ *et* $Ex(r_j) \not\sqsubseteq Ex(r_i)$*. Alors, pour toute suite de transformations de* Δ *effectuée par les règles* $r_1, ..., r_i, ... \in \Delta_{\mathcal{R}}$ *:*

$$\Delta_1, ..., \Delta_i \ ...$$

, 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. En effet,

(1) Si $r^i, r^j \in S$ telles que $r^i = r^j$, $j > i$ et $REV(r^i) = REV(r^j) =$ DIRE, 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 $i < k < j$, $E_k \sqsubseteq Ex(r^i)$ si $E_{k-1} \sqsubseteq Ex(r^i)$ et $REV(r^k)$ = DIRE. D'autre part, puisque $Ex(r^i) \not\sqsubseteq Ex(r^k)$, alors par le Théorème 4.3.19 et la Définition 4.3.6, $E_k \sqsubseteq Ex(r^i)$ 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 4.3.23, 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, par la Définition 4.3.6, $E_q \sqsubseteq ... \sqsubseteq E_k$ 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.* □

La proposition 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 est terminée après un nombre fini d'application des règles selon un ordre d'application de règles. Cette restriction, qui s'applique aux expressions de requête de révision (Exp) pour le même concept à réviser, semble raisonnable pour les deux règles de révision DIRE et OUBLIER. 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 BC 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 BC obtenue peut dépendre de l'ordre d'application de règles. Pour que la BC 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 $\Delta_\mathcal{A}$. Cette restriction doit s'appuyer sur le nombre des règles appliquées ou sur la différence du composant $\Delta_\mathcal{T}$ entre 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.

EXAMPLE 4.5.5. $\Delta_\mathcal{T}$:={
Homme ⊓ **Femme** ⊑ ⊥ ;
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**(france),
ActivitéPhysique(danse), **ActivitéChorégraphique**(danse) };
$\Delta_{\mathcal{R}}$:={
r1 : **Epreuve**(danse) ⇒ DIRE(**Equipe**, ∀membre. **Femme**),
r2 : **EquipeDeFoot**(france) ⇒ OUBLIER(**PetiteEquipe**, ≤ 5 membre),
r3 : **EquipeDeFoot**(france) ⇒ 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**, ∀membre. **Femme**),
DIRE(**Epreuve**, **ActivitéIntellectuelle**),
OUBLIER(**Epreuve**, **ActivitéChorégraphique**) et
les 4 concepts suivants sont modifiés : **Epreuve**, **Équipe**, **PetiteEquipe**, **GrandeEquipe**. En effet, après l'application de la règle r1, l'assertion **EquipeDeFoot**(france) 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 considérerons deux politiques opposées par le nombre des règles appliquées.

DEFINITION 4.5.6. 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 une *complétion de règles* ou une *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 applicable (activée) 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 BC selon laquelle la BC est modifiée au minimum après l'application des règles.
D'abord nous remarquons qu'il existe deux ensembles minimaux des règles qui génèrent deux BC différentes après les applications des règles des deux ensembles. On considère l'exemple suivant :

EXAMPLE. $\Delta_{\mathcal{A}} := \{D_1(d),\, D_2(d),\, C_1(d)\}$;
$\Delta_{\mathcal{R}} := \{$
$r_1 : C_1(d) \Rightarrow \text{DIRE}(D_1,\, exp1)$,
$r_2 : D_1(d) \Rightarrow \text{DIRE}(D_2,\, exp2)$,
$r_3 : D_2(d) \Rightarrow \text{DIRE}(X,\, exp3)$ $\}$
Il est évident que $\Delta_{\mathcal{R}}$ est non-récursif. Sans perte de la 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 BC 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.

DEFINITION 4.5.7. 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 $Ant(r'')$ est défini *directement* ou *indirectement* via $Cons(r')$ *i.e* $Ant(r'')$ dépend de $Cons(r')$.

Les nœuds sans arc d'entrée sont appelés *nœuds initiaux*. Les noeuds sans arc de sortie sont appelés *nœuds terminaux*.

La Figure 4.5.1 représente le graphe de dépendance des règles dans l'Exemple 4.5.5.

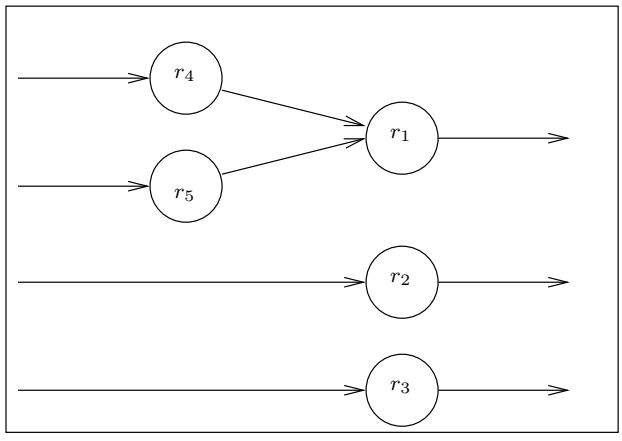

FIG. 4.5.1. Graphe de Dépendance

REMARK 4.5.8. 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.

ALGORITHM 4.5.9. *(Ordre maximal)*
Entrée : Graphe de dépendance $\mathcal{G}_R$
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.*

   (c) *Pour chaque deux nœuds DIRE ou deux nœuds OUBLIER* r_1, r_2 *correspondant aux arcs* (r_1, r) *et* (r_2, r)*, si* $Cons(r_1)$ *dépend de* $Cons(r_2)$*, alors* r_2 *est plus prioritaire 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'où* $\Delta_{\mathcal{T}}$ *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.*

EXAMPLE 4.5.10. 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 4.5.9 à ce graphe donne l'ordre maximal comme suit : (r_3, r_2, r_1).

La proposition suivante assure la correction de l'Algorithme 4.5.9 et l'existence unique de l'ensemble maximal des règles.

PROPOSITION 4.5.11. *L'ensemble des règles O retourné par l'Algorithme 4.5.9 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 les lemmes suivants.

LEMMA 4.5.12. *Soient O l'ensemble des règles retourné par l'Algorithme 4.5.9 et O' une complétion avec l'ordre. Soit* $r \in O'$ *et REV(r) = OUBLIER, qui est activée par l'application d'un ensemble de règles OUBLIER* $R' \subseteq O'$.

Si $R' \subseteq O$ *et l'ordre d'application des règles* $r' \in R'$ *est préservé dans O par rapport à l'ordre dans* O', *alors r est activée par l'application des règles* R' *dans O.*

DÉMONSTRATION. Puisque r est activé par l'application des règles nécessaires $r' \in R'$, alors les règles r' sont les prédécesseurs de la règle r dan le graphe de dépendance. Nous montrons les affirmations suivantes :

– Il n'existe pas une nouvelle règle DIRE $r_d \notin R'$ qui est un prédécesseur de r et est choisie avant une règle $r' \in R'$ dans l'ordre O. Nous le montrons par l'absurde.

  En effet, puisque l'ordre des applications des règles r' dans l'ordre O est préservé, alors les règles r' sont toujours les prédécesseurs de r dans l'ordre O jusqu'où la règle DIRE r_d est choisie à une itération de l'Algorithme 4.5.9. On a : 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 4.5.9, 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' n'est jamais choisie, ce qui contredit $r' \in R' \subseteq O$, 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 4.5.9, la règle OUBLIER r_x sera choisie au lieu de la règle DIRE r_d.

  On a $r_x \notin R'$ car si $r_x \in R'$, 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 4.5.9. C'est une contradiction.
Donc, tout chemin de la règle OUBLIER r_x vers la règle $r' \in R'$ doit passer par une règle DIRE (activée). Cela ne permet pas à la règle r_x d'activer la règle $r' \in R'$ dans l'ordre O. Cela signifie que la règle $r' \in R'$ 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.

– S'il existe une nouvelle règle OUBLIER $r_o \notin R'$ qui est un prédécesseur de r et est choisie avant une règle $r' \in R'$ dans l'ordre O, alors par le critère de priorité (c) de l'étape 5. de l'Algorithme 4.5.9, $Cont(r')$ dépend de $Cont(r_o)$ pour toute règle $r' \in R'$ qui n'est pas encore choisie. Donc, par la Remarque 4.5.8, l'application de la règle OUBLIER r_o ne casse aucun arc entre r' et r pour toute règle $r' \in R'$ 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 $\{r'|r' \in R'\}$ active la règle r dans l'ordre O', alors l'application de toutes les règles $\{r'|r' \in R'\} \cup \{r_0\}$ active également la règle r dans l'ordre O.

Dans tous les cas, nous avons démontré que si l'application de toutes les règles $\{r'|r' \in R'\}$ active la règle r dans l'ordre O', alors l'application de cet ensemble dans le même ordre dans O active également la règle r .

□

LEMMA 4.5.13. *Soient O l'ensemble des règles retourné par l'Algorithme 4.5.9 et O' une complétion avec l'ordre. Soit r ∈ O' et REV(r) = OUBLIER, qui est activée par l'application d'un ensemble de règles OUBLIER R' ⊆ O'. Si R' ⊆ O et l'ordre d'application des règles r' ∈ R' est modifié dans O par rapport à l'ordre dans O', alors r est activée par l'application des règles R' dans O.*

DÉMONSTRATION. Notons que s'il existe un arc entre une règle $r' \in R'$ et la règle r au moment où une règle $r'_1 \in R'$ est choisie dans l'ordre O', alors par la Remarque 4.5.8 et par le critère de priorité (c) de l'étape 5. de l'Algorithme 4.5.9, il existe aussi cet arc au moment où la règle r'_1 est choisie dans l'ordre O. Donc, pour chaque sous-ensemble $R'' \subseteq R'$, si toutes les règles $r'' \in R''$ sont choisies par les deux ordres O' et O, alors tous les prédécesseurs r' de r qui existent suivant l'application de toutes les règles $r'' \in R''$ dans l'ordre O' restent encore les prédécesseurs r' de r suivant l'application de toutes les règles $r'' \in R''$ dans l'ordre O. Ainsi, on peut utiliser l'argument dans la démonstration du Lemme 4.5.12 pour déduire qu'il n'existe pas d'une règle DIRE qui est choisie avant une règle $r' \in R'$ dans l'ordre O. De plus, on peut déduire également que s'il existe une règle OUBLIER $r_o \notin R'$ qui est un prédécesseur de r et est choisie avant une règle $r' \in R'$ dans l'ordre O, alors cela ne change pas le fait que la règle r soit activée. Tout cela assure que l'application d'une règle $r'' \notin R'$ dans l'ordre O ne change pas l'activation de la règle r.
Maintenant, supposons qu'il existe deux règles $r'_1, r'_2 \in R'$ sont choisies et appliquées dans cet ordre dans O' mais, par contre, ces deux règles sont choisies et appliquées dans l'ordre inverse dans O. Puisque r'_1, r'_2 sont les prédécesseurs de r, alors le graphe $\mathcal{G}_R$ a deux arcs (r'_1, r) et (r'_2, r). Si $Cons(r'_1)$ et $Cons(r'_2)$ ne sont pas dépendantes,

alors les concepts $Ant(r)$ obtenus par les deux ordres d'application : $[r'_1, r'_2]$ dans O' et $[r'_2, r'_1]$ dans O, sont équivalents. Si $Cons(r'_1)$ et $Cons(r'_2)$ sont dépendantes, selon le critère de priorité 5.c de l'Algorithme 4.5.9, on a : $Cons(r'_1)$ dépend de $Cons(r'_2)$. Donc, $Cons(r'_2)$ obtenu des deux ordres d'application : $[r'_1, r'_2]$ dans O' et $[r'_2, r'_1]$ dans O, sont équivalents.
Montrons que $Cons_{[r'_1,r'_2]}(r'_1) \sqsubseteq Cons_{[r'_2,r'_1]}(r'_1)$ où $Cons_{[r'_1,r'_2]}(r'_1)$ est le concept $Cons(r'_1)$ obtenu de l'ordre d'application $[r'_1, r'_2]$ dans O' et $Cons_{[r'_2,r'_1]}(r'_1)$ est le concept $Cons(r'_1)$ obtenu de l'ordre d'application $[r'_2, r'_1]$ dans O.

En effet, puisque $Cons(r'_1) \sqsubseteq Cons_{[r'_2]}(r'_1)$ et $Cons(r'_1)$ dépend de $Cons(r'_2)$, alors, par la Remarque 4.3.9, on a : $Cons_{[r'_1]}(r'_1) \sqsubseteq Cons_{[r'_2,r'_1]}(r'_1)$. De plus, si $Cons_{[r'_1]}(r'_1)$ dépend de $Cons(r'_2)$, alors $Cons_{[r'_2,r'_1]}(r'_1)$ dépend également de $Cons(r'_2)$ (la relation de subsomption est préservée de la substitution de $Cons(r'_2)$ par $Cons_{[r'_2]}(r'_2)$). Donc, on a : $Cons_{[r'_1,r'_2]}(r'_1) \sqsubseteq Cons_{[r'_2,r'_1,r'_2]}(r'_1)$.

D'autre part, puisque $Cons_{[r'_2]}(r'_2) \not\sqsubseteq Ex(r'_2)$, $REV(r'_1) = REV(r'_2) =$ OUBLIER et donc, $Cons_{[r'_2]}(r'_2) = Cons_{[r'_2,r'_2]}(r'_2)$, alors $Cons_{[r'_2,r'_1,r'_2]}(r'_1) = Cons_{[r'_2,r'_1]}(r'_1)$. On obtient : $Cons_{[r'_1,r'_2]}(r'_1) \sqsubseteq Cons_{[r'_2,r'_1]}(r'_1)$.
Par ailleurs, tout ordre des règles $[r'_{i_1}, ..., r'_{i_l}]$ peut être transformé en l'ordre $[r'_1, ..., r'_l]$ par plusieurs échanges $[r'_{i_u}, r'_{i_v}]$. Cela implique qu'un concept $Ant_1(r)$ obtenu de l'ordre d'application des règles dans O' est subsumé par le concept $Ant_2(r)$ obtenu de l'ordre d'application des règles dans O *i.e* si la règle r est activée par l'ordre d'application des règles $r' \in R'$ dans O', alors la règle r est également activée par l'ordre d'application des règles $r' \in R'$ dans O. □

A partir des Lemmes 4.5.12 et 4.5.13, nous établissons la démonstration de la Proposition 4.5.11.

DÉMONSTRATION. (la proposition 4.5.11) D'abord on montre que l'Algorithme 4.5.9 se termine. Effectivement, 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 4.5.7 (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 4.5.9 pour la saturation de O se termine. Et cela est également assuré par la Proposition 4.5.4. On obtient que O est une complétion.
S'il n'y a pas de confusion, on peut référer une règle au lieu du nœud correspondant dans un graphe de dépendance.
Maximalité : Soit *O'* un ordre d'application de règles tel que *O'* est une complétion. On doit démontrer que $O' \subseteq O$. Soit $r \in O'$.

- Si r est activée initialement, on a $r \in O$. Effectivement, supposons que $r \notin O$. Alors à une itération de l'Algorithme 4.5.9, 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 n'a jamais lieu selon la construction de l'étape 4. de l'Algorithme 4.5.9.
- Si r est activée par l'ensemble des règles OUBLIER nécessaires R' dans l'ordre O' où les $r' \in R'$ sont activées (initialement ou non) et choisies avant r, alors

les $r' \in R'$ sont des prédécesseurs de r dans $\mathcal{G}_R$ dans l'ordre O'. On montre que si $R' \subseteq O$, alors $r \in O$. On considère les cas suivants :

(1) L'ordre des applications des règles $r' \in R'$ est préservé dans O par rapport à l'ordre dans O'. Par le Lemme 4.5.12, on obtient que r est activée .

(2) L'ordre des applications des règles $r' \in R'$ dans O est modifié par rapport à l'ordre dans O'. Par le Lemme 4.5.13, on obtient également que $r \in O$.

Ces affirmations nous permettent de conclure que la règle r devient activée dès que les règles OUBLIER $r' \in R'$ sont toutes choisies et appliquées dans l'ordre O. Dans ce cas, aux itérations suivantes les choix des $r' \in R'$ dans l'ordre O, il est impossible qu'un prédécesseur DIRE de r est choisie si r n'est pas choisie auparavant selon la construction de l'étape 4. de l'Algorithme 4.5.9. Cela veut dire que la règle r ne sera pas désactivée par une autre règle dans O. Puisque la règle r est toujours activée et O est une complétion, la règle r doit être choisie à une itération suivante dans l'ordre O. Donc, $r \in O$.

On a démontré que chaque règle r qui est activée et appliquée dans l'ordre O', est également activée et appliquée dans l'ordre O. En conséquence, O est la complétion maximale.

Unicité : 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. □

EXAMPLE 4.5.14. (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**,

Européen $\sqcap$ **Américain** $\sqsubseteq \bot$,

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** $\sqcap \exists$*contrôller*.**Toxicité** $\sqcap \exists$*résister*.**Chaleur**

$\}$

et
$\Delta_{\mathcal{T}_2}$ (construction) := {
ProdRésAuPolluant := **Produit** ⊓ ∃*résister*.**Feu** ⊓ ∃*résister*.**Bruit** ,
ProdDImport := **ProdRésAuPolluant** ⊓∃*contrôller*.**Toxicité**
}
Le rôle *contrôller* 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) ∧ *acheté-par*(X, Z_1) ∧
associer(Y_1, Y_2) ∧ *associer*(Z_1, Z_2) ∧
Européen(Y_2) ∧ **Américain**(Z_2)
⇒ DIRE(**ProdRésPolluant**, ∃ *résister*.**Feu**) ;
r_2 : **EntrepriseDeConst**(Y_1) ∧ *vendu-par*(X, Y_1) ∧
ProdRésPolluant(X)
⇒ OUBLIER(**ProdDImport**, ∃ *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** := ∃*associer*.**Européen** impliquent *associer*(b, b') et **Européen**(b') .
Les faits **EntrepriseAmér**(c) et **EntrepriseAmér** := ∃*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 proriété 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** $\sqcap\exists$*contrôller.***Toxicité** $\sqcap$ $\exists$*résister.***Chaleur**, $\exists$ *résister.***Chaleur**)
= **ProdRésAuPolluant** $\sqcap\exists$*contrôller.***Toxicité**

Selon l'Algorithme 4.3.21, on obtient :

DIRE (**Produit** $\sqcap$ $\exists$*résister.***Bruit** , $\exists$ *résister.***Feu**)
= **Produit** $\sqcap$ $\exists$*résister.***Bruit** $\sqcap\exists$ *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 4.3.21, on obtient :

DIRE (**Produit** $\sqcap$ $\exists$*résister.***Bruit** ,$\exists$ *résister.***Feu**)
= **Produit** $\sqcap$ $\exists$*résister.***Bruit** $\sqcap\exists$ *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).

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 nombre 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

Commerce électronique et langage ebXML : état de l'art

==========

Dans la section suivante, nous indiquerons comment concevoir une ontologie pour Core Component de l'ebXML. Nous présenterons également comment concevoir et implémenter sur cette ontologie les services d'inférences qui permettent d'intégrer les vocabulaires locaux.

3. Intégration des vocabulaires dérivés dans l'ebXML

Le mélange du langage naturel et de formulaires d'informations HTML sur Internet actuel est une barrière majeure pour l'automatisation du commerce électronique car la sémantique des informations est compréhensible seulement pour l'homme. L'ebXML s'appuyant sur XML est l'un des issus de ce problème.

3.1 Définition du problème

La représentation sémantique de l'ebXML et les parties portant la sémantique sont discutées dans [2].

Puisque l'utilisation de CCs est essentielle pour l'analyse et la conception des documents commerciaux ebXML, ce dernier propose une méthode de les découvrir à partir des processus commerciaux qui évoquent l'échange de ces documents. Cette méthode est décrite dans [2]. De plus, la réutilisation CCs peut se traduire en la recherche des CCs sur Core Library. Cette recherche accepte comme une entrée une description d'un CC en langage de représentation de l'ontologie. Dans plusieurs cas où il n'existerait pas d'exact appariement, de certains CCs seraient réutilisés. S'il n'existe pas d'appariement, de nouveaux CCs sont ajoutés au Core Library comme des composants spécifiques de domaine. Les informations contextuelles dans lesquelles ces CCs sont utilisés y sont enregistrées. Pour éviter de maintenir l'ontologie Core Library énorme nous proposons une technique comme suivante :

* Une ontologie de base

* Les ontologies locales qui dérivent de l'ontologie de base

* Développements des services nécessaires permettant de transformer les termes d'une ontologie dérivée vers l'autre

Nous choisissons la logique de description (DL) pour concevoir les ontologies. La justification de notre choix est présentée dans [2].

3.2 Représentation of CC en DL

+Composant de base (Core Component)

Un composant de base, appelé Core Component Type, est un bloc commun et est utilisé à travers des industries, donc il est libre de contexte. Les composants de domaine, appelés Basic Information Entity et Aggregate Basic Information Entity, sont spécifiques pour un contexte commercial. Ces derniers portent une sémantique.

Par exemple, ń quantity ż n'a aucune de sens, mais ń quantity shipped ż porte une signification.

Pour chaque composant de base, les informations suivantes doivent être définies : nom, description de la nature et du sens, identifiant unique, composants réutilisés, type de données, remarques (exemples, références), Core Component Type et finalement, convention de nomination.

+ Représentation

La logique DL est présentée dans [1] et [2] comme un outil permettant de construire une ontologie. L'initiative de cette démarche est langage OIL1 facilitant la conception des ontologies basées sur DL. Il s'agit d'un langage qui s'appuie sur XML au niveau de syntaxe et sur DL au niveau de sémantique. Toutes les informations sur OIL se situent sur [4]. Le dictionnaire CC, défini dans [5], peut être transformé vers une ontologie OIL sur laquelle les services d'inférences DL sont implantés. Ces services facilitent sans doute la génération et l'interprétation de documents commerciaux.

==========

1. Introduction

La prolifération de standards et d'initiatives différents (RosettaNet, BizTalk, etc.) qui ont pour objectif de faciliter l'échange de documents commerciaux fait apparaître plusieurs systèmes de classification de produits et de services. Cette diversité montre que le marché B2B n'a pas atteint un consensus sur les systèmes encodants, sur le niveau de détail de leurs descriptions. Dans ce contexte, il coexiste deux axes de recherche. Il s'agit pour le premier axe de réunir tous les standards et initiatives existants pour construire un standard unique dans lequel un vocabulaire de base est défini. L'ebXML résultant de cet axe est proposé par le monde industriel. Pour le deuxième axe il s'agit de présenter un mécanisme permettant de trouver les transformations adaptées entre des vocabulaires indépendants quel que soit la provenance.

Cependant, la première approche exige une acceptation large des entreprises sur le standard unifié. De plus, un vocabulaire unique pour tout le commerce peut être énorme et difficile à maintenir. Cette approche demande également les transformations nécessaires de standards (vocabulaires) existants vers le nouveau standard unifié. Ces conditions ne sont pas faciles à atteindre, à moins que le nouveau standard puisse montrer nettement ses avantages. En ce moment l'ebXML est considéré comme le plus prometteur pour les raisons suivantes :

* UN/CEFACT et OASIS qui créent l'ebXML sont des organismes internationaux importants, non-profits. UN/CEFACT s'occupe du développement technique et politique concernant la facilitation du commerce électronique dans le monde entier. Cette entreprise a développé avec succès le standard EDIFACT. Quant à OASIS, elle est l'un des quatre seules organismes dans le monde capable de créer les standards de jure (légaux)

* Les entreprises haute technologie comme IBM, Sun Microsystem, Oracle... dont les ingénieurs participent à la construction du standard.

Quant à la deuxième approche, la recherche des transformations sans perte de sémantique nous met toujours au défi. En effet, à notre connaissance il n'existe pas de technique qui permette de traduire sans perte d'information un terme défini par un vocabulaire vers un autre. Et pourtant, les techniques proposées récemment fournissent les transformations approximatives et permettent de mesurer la différence

entre la version originale et celle de transformée. Dans cette direction, une technique typique proposées dans [6] utilise un classificateur Naive-Bayes. Cette technique s'appuie sur la probabilité conditionnelle et la technique d'apprentissage automatique.

L'idée principale de cette technique peut être décrite comme suit : supposons qu'il faut transformer les termes entre deux vocabulaires utilisés dans deux ensembles de documents différents. Nous pouvons entraîner le classificateur Naive-Bayes sur chaque document. Cela permet au classificateur d'obtenir la probabilité d'occurrence d'un certain mot participant à une classe. (chaque classe a une description). A partir de ces probabilités (connaissances acquises), une description d'un terme est classifiée dans chaque vocabulaire avec des probabilités, respectivement P1 et P2. Pour découvrir une transformation entre la classe source et la classe destinataire il faut alors maximiser la probabilité d'occurrence d'une description dans les deux classes en même temps. Il est évident que cette méthode a besoin des interventions humaines pour vérifier les transformations obtenues. Des informations plus complètes sur cette méthode se trouvent dans [6].

Dans ce contexte, notre recherche de l'intégration des vocabulaires s'oriente vers les axes suivants :

* Si un standard est largement accepté, il aura besoin d'un vocabulaire de base. Nous essayons de proposer une technique permettant de dériver les vocabulaires locaux de ce dernier. Cette technique doit fournir des mécaniques transformant sans ambiguïté un terme du vocabulaire dérivé vers un autre. Par conséquent, la maintenance d'un vocabulaire unique et énorme pourrait être évitée. Notre technique proposée sera développée sur l'ebXML.

* Dans le cas où aucun standard ne serait nettement favori, la construction d'un méta-vocabulaire à partir des vocabulaires existants (standards) est une solution. Ces vocabulaires peuvent être considérés comme les dérivés de ce méta-vocabulaire. Dans ce cas là, nous pouvons profiter de la technique obtenue de l'axe de recherche précédent.

La section suivante est consacrée à la présentation de l'ontologie. Cette technique contribue à résoudre les problèmes posés par un système d'échanges de données de façon transparente sémantique en général et par le système ebXML en particulier.

5.1. Introduction

Les échanges d'informations sous forme de document tiennent une place importante dans les flux d'informations entre les acteurs. Ces informations qui sont des données administratives, techniques, commerciales (processus commerciaux, données des produits, des fournisseurs, etc.), etc. sont explicitement formulées sur papier pour des raisons de vérification de la sécurité, de législation et de complexité. Aujourd'hui, l'apparition des NTIC1 permet de remplacer peu à peu la communication sur ń papier ż par une manière électronique d'échanger les informations en conservant la même fiabilité, le même statut juridique mais avec plus de rapidité et de précision. La nouvelle manière d'échanger a besoin non seulement de réorganiser la procédure et le contenu d'échange mais aussi de les formaliser. Ainsi, la composition et l'interprétation de documents échangés exigent un mécanisme permettant de capturer et représenter la sémantique à partir du monde réel. Ce mécanisme nécessite des modèles d'échange formels ou informels sur lesquels il se fonde. Par conséquent, la construction d'un

système d'échange transparent sémantique pour les utilisateurs avec un tel modèle nous met toujours au défi. Particulièrement, comment représenter et structurer des documents échangés permettant à des acteurs d'effectuer des échanges sans accord préalable, c'est-à-dire, par exemple, qu'un nouvel acteur est capable de participer au système d'échanges sans difficulté.

En effet, pour l'échange de données techniques dans le secteur de la construction, par exemple, où les données sont relativement bien définies, nous nous confrontons déjà à des difficultés. Une analyse du CNRS a indiqué que la production, la diffusion et la circulation de ces documents sont des sources majeures de dysfonctionnement, de mauvaise qualité et de surcoût dans l'exécution de l'ouvrage de construction. L'auteur de l'étude a souligné la nécessité des solutions basées sur une meilleure communication entre les acteurs, mais aussi sur un échange d'information à fort contenu sémantique, adapté au rôle et à l'objectif de chaque acteur. Il existe des travaux qui ont proposé des solutions différentes s'appuyant sur les standards existants comme STEP, CORBA, etc. Cependant, ces solutions répondent partiellement au problème de l'interopérabilité aux deux niveaux : opérationnelle et sémantique. L'absence d'un modèle formel a limité la capacité d'inférence, l'expression sémantique, et la généralité de ces standards.

D'autre part, pour l'échange de données commerciales où les statuts d'acteurs sont très variés, la majorité des données sont sous une forme implicite et ambiguë, les difficultés se multiplient. Heureusement, la brillante perspective de l'eCommerce pour la réduction considérable du coût d'échange encourage de grandes entreprises à franchir ces difficultés. Les modèles d'échanges basés sur des standards résultent de ces efforts. Premièrement, c'est la recherche d'un standard, qui utilise un format propriétaire, permettant à tous les acteurs d'effectuer des échanges selon des accords préalables sur des protocoles d'échanges. Il s'agit du modèle EDI2, typiquement EDIFACT3, qui est un exemple réussi dans cette direction. Ce modèle-là, néanmoins, est fortement critiqué par la plupart des entreprises (PME) à cause de son coût et de l'absence de l'interopérabilité sémantique (voir la section 2.1). De plus, aujourd'hui, l'accessibilité, la performance des réseaux (Internet), et l'arrivée du langage de balise XML répondent mieux aux exigences de ces industries. C'est pourquoi l'EDI n'est plus le choix unique. L'événement le plus important dans ce contexte est la naissance de l'ebXML. C'est un standard universel (ou presque) dans le monde commercial grâce à son influence sur tous les secteurs industriels. Ce modèle-là définit des spécifications détaillées sur les modèles de données (processus commerciaux, structuration de documents commerciaux, etc.), l'ensemble de composants de bases indépendants des secteurs industriels et libres de contexte, un registre des informations d'acteurs, etc. Finalement, tout document échangé est représenté par XML (voir la section 2.2).

Une autre approche basée sur un langage formel capable de capturer une sémantique très proche de celle du langage naturel est en cours de développement. L'ambition de cette approche est de trouver une représentation suffisamment puissante résultant de la logique modale sur laquelle est fondé le langage. Ce langage-là donnera les moyens de composer les documents supportant l'échange transparent sémantique pour les utilisateurs. Alors tous les acteurs parlent la même langue avec la même base de connaissances pour décrire le monde, donc ils se comprennent. Par

conséquent, l'échange de données aura lieu sans accord préalable ! L'exemple le plus typique dans cette direction est le langage FLBC4 (voir la section 2.3).

Un document échangé peut être considéré comme une hiérarchie de concepts dont la racine est le concept document. Il est très fréquent qu'un concept soit différemment interprété selon les connaissances sur le domaine, la géographie et la culture des acteurs. Ainsi, il peut exister un même concept qui porte des noms différents et qui s'exprime par plusieurs descriptions selon le domaine. L'échange de données transparent sémantique entre deux acteurs sous la forme de document implique donc un appariement des deux bases de connaissances. Cet appariement devrait être capable de déterminer une identité, une similarité, ou une différence totale entre les descriptions des concepts étant contenus dans le document. Les deux dernières entraîneraient la mise à jour des bases de connaissances. Ces concepts seraient représentés dans les bases de connaissances par un modèle formel ayant une grande capacité expressive et des services d'inférence efficaces. Ce modèle-là permettrait de détecter un concept avec des descriptions différentes, d'engendrer un nouveau concept à partir des concepts similaires, etc.

La question qui nous intéresse est la suivante : trouver un tel modèle formel qui, à la fois, soit approprié à la représentation sémantique permettant la transparence d'échange de documents et facilite le développement d'algorithmes d'inférence, de construction de connaissances spécifiques de façon dynamique. Particulièrement, ce modèle-là devrait permettre de faciliter la capture, la représentation sémantique commerciale, de composer dynamiquement des documents commerciaux avec de nouvelles connaissances, de traduire sans encombre des tels documents avec une base de connaissances propre.

En partant de la représentation des bases de connaissances en XML et de l'intégration de ces bases, nous allons examiner également la possibilité de gestion de bases de connaissances de façon décentralisée.

Le reste de ce rapport sera organisé en section. La deuxième constitue l'étude bibliographique dans laquelle nous présenterons deux catégories de modèles d'échange. Nous indiquerons également les avantages et les inconvénients pour chaque modèle. Une synthèse sur ces modèles achèvera la section. La troisième section se concentrera sur la logique descriptive (description logic) dans l'espoir de trouver un modèle formel capable de s'appliquer au plus de domaine possible. Dans cette section, nous essayerons d'esquisser la structure et le fonctionnement du modèle. Nous indiquerons également le travail restant à faire pour compléter la base théorique. La quatrième section sera consacrée à l'étude d'une application de ce modèle-là à l'eCommerce.

2 Modèles d'échange de données

2.1 Modèle EDI

2.1.1 Définition

[Claude CHIARAMONTI, Echange de données informatisé]

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.

Il faut savoir que l'objectif de l'EDI est de fournir à des entreprises une plateforme pour échanger automatiquement des données administratives et commerciales

entre les ordinateurs. De plus en plus il devenait un outil de communication standardisé. En effet, dans le monde EDI, il existe deux normes les plus importantes : ANSI X12 pour l'Amérique du Nord et EDIFACT pour le reste. L'organisation OASIS et ONU ont réuni tous les groupes professionnels qui avaient 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. Les mêmes groupes se sont 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. Le plus important parmi eux est le problème de transparence sémantique.

2.1.2 EDIFACT et ses répertoires

Le standard EDIFACT comporte deux blocs principaux. Le premier est un vocabulaire étant contenu 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 (expliquer sa signification conventionnelle et aider à définir le contenu de l'information fournie) ; iv) une spécification du mode de représentation de l'information (l'espace disponible ...). Une partie importante de la sémantique EDIFACT s'y inclut. Ce vocabulaire est établi par les spécialistes et recouvre de différentes activités : transactions commerciales, opérations logistiques, formalités administratives.

Le deuxième bloc correspond aux définitions syntaxiques comparables à des règles grammaires. Il permet de structurer les éléments afin de pouvoir construire des segments types, et 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 les éléments : AAA est ń prix net ż ; SRP est ń prix proposé ż

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, la fin du segment. Les segments sont identifiés par un code alphabétique à trois caractères. Il y a deux catégories de segments : i) les segments de données qui sont composés uniquement de données : les noms, les adresses, des 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 servent à l'échange interactif : les types de règles de syntaxe, l'émetteur du message, etc.

2.1.3 Accord d'échange : sous-ensemble du standard

Les messages EDIFACT (messages types) sont des unités finales avec une signification entière de la communication. Ils peuvent remplacer les documents en papier dans la méthode traditionnelle. Mais la structuration des messages est très variée en

fonction des domaines d'activité. Elle est l'un des objectifs de la négociation entre deux acteurs avant d'arriver au premier échange. On dit également l'accord d'échange.

L'accord d'échange 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 précise les méthodes à utiliser pour la transmission physique de données dans le cadre de l'application considérée, définit les méthodes de codage communes qui devraient être utilisées par les acteurs, les problèmes juridiques et 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 que lieu entre les membres dans le même groupe. La négociation est indispensable pour sélectionner les structures des messages appropriées à son domaine, pour détailler les standards logiciels utilisés (ex : traducteurs), etc. Une quantité importante de connaissances ne peut pas être découverte à partir des messages échangés mais elle est décrite dans les manuels d'utilisateurs.

Voici un message ń Order ż en EDIFACT :

UNH+1+ORDERS :D :96A :UN' // indiquer que le document est celui de l'EDIFACT et identifier le type de

//document : ORDERS

BGM+220+AGL153+9+AB' // ń begin of message ż

DTM+137 :20000310 :102' // indiquer ń Date or Time of the Order ż

DTM+61 :20000410 :102' // indiquer ń Date or Time is the last day for deliveryż

NAD+BY+++PLAYFIELD BOOKS+34 FOUNTAIN SQUARE PLAZA+CINCINNATI+OH+4520

// indiquer ń name and address of buyer ż

NAD+SE+++QUE+201 WEST 103RD STREET+INDIANAPOLIS+IN+46290+US'

// indiquer ń name and address of seller ż

LIN+1' // la première ligne de l'ordre

PIA+5+0789722429 :IB'QTY+21 :5' // indiquer ń product indentifier ż et ń quantity ż

PRI+AAA :24.99 : :SRP' // indiquer ń price ż

LIN+2' //

PIA+5+0789724308 :IB'

QTY+21 :10'

PRI+AAA :42.50 : :SRP'

UNS+S' // indiquer que les messages suivants sont le résumé du message

CNT+3 :2' // indiquer ń count = number of lines ż

UNT+17+1' // ń number of segments and of messages (repetition) ż

N.B : la structure d'un message varie selon le secteur industriel. Dans quelques 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.

2.1.4 Avantages

* Automatisation des traitements et échange entre applications hétérogènes. L'EDI n'est qu'une forme de l'échange d'informations utilisant les réseaux. Il peut être défini comme l'échange entre applications hétérogènes, avec automatisation des traitements.

Ainsi, il s'agit d'échanges non seulement de machine à machine mais de système d'information à système d'information, le tout sans qu'il y ait à modifier les applications amenées à fournir ou recevoir des informations.

* Echange 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.

2.1.5 Limites

* Les documents ń fichiers plats ż et illisibles pour humain. La complexité de ses messages est causée par le format très compressé à cause des contraintes de réseaux.

* Les coûts élevés. Ils sont le coût des développements de traducteurs, coût de l'intégration des traducteurs au système, le coût de l'installation des réseaux de communications, le modifications pour de nouveaux acteurs et coût de négociation pour parvenir à des agréments d'échange.

* Manque d'interopérabilité sémantique (problème transparent). Il est très difficile d'établir des 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 des pays, des langues, des 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 supportées. Le monde EDI-EDIFACT est resté isolé.

2.2 EbXML (electronic business XML)

2.2.1 Présentation XML

Le XML est un langage de balises capable 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 ń mise en page ż) 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 : le format des documents pour l'échange inter-applications, le publication sur le WEB (création de pages dynamiques, transmission de données et mises à jour, présentation sur de multiples supports) et pour la gestion de données. L'ebXML est une initiative profitant des capacités du XML pour développer une plate-forme eCommerce.

2.2.2 Initiative ebXML

Figure 1 : Interaction entre deux entreprises ebXML - une vue de haut niveau

L'ebXML propose un ensemble des spécifications permettant aux entreprises, 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 1 est une illustration de ń ebXML Technical Architecture Specification ż. L'entreprise A examine le contenu du Référentiel ebXML (Repository) , notamment Bibliothèque de base, pour déterminer si l'ebXML est approprié à son business ou quels sont les exigences d'implémentation de l'ebXML. En se basant sur les informations disponibles sur le Référentiel ebXML, A peut construire ou acheter l'implémentation ebXML adaptée à ses transactions ebXML. Dans l'étape suivante, A crée et enregistre un CPP dans le Référentiel. A peut contribuer à de nouveaux processus commerciaux. CPP contient les informations nécessaires pour que un partenaire potentiel puisse déterminer le rôle commercial de A et le type de protocole pour ce rôle. Après l'enregistrement de A, B peut rechercher et examiner le CPP de A pour savoir si celui-ci est compatible avec le sien. Si c'est le cas, une négociation sera automatiquement effectuée pour arriver au CPA. Finalement, les deux entreprises configurent leurs système en se basant sur le CPA obtenu. Alors, la première transaction peut commencer !

Dans les sous-sections suivantes, nous ne nous concentrerons que sur l'analyse des composants ebXML concernant la représentation sémantique commerciale car notre objectif est d'étudier comment l'ebXML résout le problème de l'interopérabilité sémantique et cela jusqu'à quel point.

2.2.3 Capture et représentation de la sémantique commerciale

Dans cette sous-section nous examinons comment l'ebXML capture et représente la sémantique commerciale. Nous expliquons également ce processus sous la forme des étapes via un exemple. 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 capture de ces informations. Cette analyse pourrait ouvrir une direction permettant de formaliser le mécanisme de capture et de représentation sémantique commerciale.

2.2.3.1 Processus commerciaux : capture de la sémantique

L'ebXML utilise le mécanisme, qui s'appelle UMM (UN/CEFACT Modeling Methodology), 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 chorégraphique de transactions commerciales. Chaque transaction s'exprime par un échange de documents commerciaux. Les documents commerciaux se composent des objets d'informations commerciales (Business Information Objects). Les objets d'informations commerciales peuvent se composer des composants de base (Core Components) (les concepts documents commerciaux, objets d'informations commerciales, composants de bases seront précisés dans 2.2.3.2). 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 permettant une collaboration commerciale avec un autre partenaire (chaque schéma qui est décrit par deux versions UML et XML permet de configurer le système ebXML run-time). 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 standards patterns. Les spécification de processus commerciaux servent à former les CPP (Collaboration Protocol Profiles) et les CPA (Collaboration Protocol Agreements).

Un processus d'analyse pour capturer la sémantique d'un processus commercial spécifique s'effectue par une équipe de spécialistes. Cette équipe-là 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 étant inclus dans les transactions. La sortie des activités de l'analyse est les spécifications du processus commercial et les définitions de documents commerciaux. Ces spécifications sont enregistrées dans une bibliothèque commerciale (Business Library). Une bibliothèque commerciale est un référentiel ebXML (repository) des spécifications des processus commerciaux et des objets d'informations commerciales. Donc, une bibliothèque commerciale publique supporte fortement l'interopérabilité commerciale.

Pour préciser le processus de capture sémantique commerciale, l'ebXML propose les fiches descriptives (worksheets) conformément à l'UMM. Cette méthode comporte plusieurs étapes comme suit (voir l'exemple dans l'annexe) :

i) Identification et découverte de processus commercial (Business Process Identification and Discovery)

Cette étape sert à faire un dénombrement (inventory) des processus commerciaux et à identifier l'existence des processus et les intéressés (stakeholder) sans décrire des détails. Concrètement, on essaie de comprendre la structure et la dynamique du secteur commercial, d'unifier la reconnaissance des utilisateurs, les fournisseurs logiciels, etc. sur le domaine commercial . D'abord, un cadre de référence pour les processus commerciaux est défini (fiche 1). Et puis, on regroupe les processus commerciaux selon leur fonction commerciale (fiche 2) et définit un ensemble de types de processus (fiche 3). Chaque type de processus est une série de processus formant un secteur commercial. Finalement, les processus sont identifiés (fiche 4).

ii) Elaboration de processus commercial.

Les fiches descriptives de cette étape identifient les acteurs et les pre-, post-conditions des processus commerciaux. Dans cette étape nous commençons à entrer dans l'analyse de conception. L'entrée et la sortie du processus doivent être spécifiées alternativement dans la pré-condition et la post-condition (fiche 5)

iii) Collaboration commerciale et éléments économiques.

Dans ces fiches descriptives, nous définissons des événements économiques qui ont lieu pour accomplir le processus commercial. Elles permettent de définir des frontières du système et le protocole gouvernant le flux d'information. D'abord, l'entrée et la sortie du déclenchement de la collaboration sont spécifiées, ensuite, le rôle et la contrainte de la collaboration (fiche6). Le protocole de la collaboration est également décrit (fiche 7)

iv) Transaction commerciale et rôle autorisé.

Dans cette étape, on définit les activités actuelles and parties autorisées dans l'organisation qui initie ces transactions. L'objectif de l'étape est d'identifier les transactions qui implantent la collaboration. Chaque transaction possède plusieurs activités et chacune dispose d'un rôle autorisé (fiche 8). Le fiche 9 est conditionnel

v) Description d'informations commerciales.

Ces fiches descriptives définissent la longueur des champs de l'information, types de données, descriptions, traçabilité de requis et, le contexte pour construire le document à partir de Composants de Base (Core Components, sous-section 2.2.3.2). L'objectif de cette étape est d'identifier les informations requises pour les documents commerciaux spécifiés dans les transactions commerciales (fiche 10,11).

2.2.3.2 Composant de base : représentation d'une partie de la sémantique

Le processus commercial détermine les caractéristiques du charge final (payload) du document commercial. Parmi ces caractéristiques, il y en a qui varient dramatiquement à travers des industries alors que les autres ne varient pas. Si l'on considère un document comme un ensemble de composants où chacun est un ń bloc ż contenant des informations commerciales, alors les composants de base sont des composants capable d'apparaître dans plusieurs circonstances et dans des domaines commerciaux différents. Un composant de base, s'appelle 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, s'appelle Basic Information Entity et Aggregate Basic Information Entity dans le dictionnaire de composants de base, sont spécifiques pour un contexte commercial. Ces derniers portent une sémantique. Par exemple, ń quantity ż n'a aucune de sens commercial (fiche de composant CCT, section 5.2), mais ń quantity shipped ż porte une signification (fiche de composant BIE, section 5.2). 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 (des mots ou phrases ont le même sens que le nom du composant de base. Ils servent à capturer des noms communs, commerciaux du composant de base), les composant réutilisés, type de données, remarques (exemples, références), Core Component Type 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, les documents commerciaux sont toujours construits 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. Ils peuvent être extraits de la bibliothèque commerciale ou bibliothèque de domaine (composant de domaine) ou bibliothèque de base (composant de base).

2.2.3.3 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 la pré-définition d'une telle bibliothèque n'est pas adaptée 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 essayons de répondre aux questions : comment ebXML définit le contexte et comment il applique le contexte sur des 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 chaussure, les valeurs contextuelles sont les suivantes :

Catégorie contextuelle
Valeur
Process
Procurement (Approvisionnement)
Product Classification
Glue (colle)
Region d'acheteur
France
Region de vendeur
USA

Avec la définition ci-dessus, le contexte fournit les informations pour éviter des conflits de données, exprimer les relations commerciales entre des données, découvrir des composants de base, etc.

Pour répondre à la deuxième question, l'ebXML définit un modèle hiérarchique pour les composants de base avec le contrôle contextuel. La racine de la hiérarchie, s'appelle ń Core Component Type ż, ne contient aucune d'information sur la sémantique. ń Basic Information Entity ż et ń Aggregate Information Entity ż sont les descendants de ń Core Component Type ż décrivant les facteurs sémantiques. Le contexte représenté par les règles de contexte (rule context) modifie ń Basic Information Entity ż et ń Aggregate Information Entity ż pour obtenir les composants de base appropriés au processus commercial. Ces derniers, composants communs, constituent effectivement les documents commerciaux. (voir l'exemple dans 2.2.4)

2.2.3.4 CPP et CPA (Collaboration Protocol Profile, Collaboration Protocol Agreement)

L'échange d'informations entre deux partenaires exige chacun de connaître des collaborations commerciales supportées par l'autre, le rôle de l'autre dans les collaborations commerciales et les détails techniques avec lesquels l'autre émet et reçoit les messages. Le CPP d'un partenaire décrit (en XML) des collaborations commerciales supportées et la capacité d'échange de messages. Les informations sur les collaborations commerciales supportées viennent des BPSS dans lesquelles la sémantique des transactions, des chorégraphies est déterminée. Le CPA définit la manière avec

lesquelles les deux partenaires feront l'interaction fondée sur les collaborations commerciales choisies. Les deux partenaires utilisent des copies identiques du CPA pour configurer leurs système. Les CPPs sont stockés sur les Registres ebXML pour faciliter la recherche d'un partenaire souhaitable.

Pour former le CPA à partir des deux CPP, les deux partenaires calculent l'intersection d'informations de deux CPP. La figure 3 montre le processus de calculs pour déterminer le CPA.

2.2.4 Composition du document commercial

Dans cette sous-section, nous essayons d'examiner comment composer de façon automatique un document commercial en s'appuyant sur les éléments présentés dans les sous-sections ci-dessus. Un canevas de l'ebXML présente un mécanisme fondé sur les règles de contexte qui permet d'assembler des composants de base pour obtenir un document souhaitable.

Le processus d'assemblage d'un document est un processus qui utilise les règles pour extraire les composants de base appropriés du référentiel ebXML. L'idée principale de ce processus s'appuie sur le contexte (il vient de l'analyse du processus commercial) qui fournit un canevas permettant d'adapter des composants de base aux besoins commerciaux spécifiques en conservant la transparence du processus de transformation. En partant des composants de base et en utilisant le contexte pour dériver un composant de base dépendant du contexte, on peut assurer que l'information dans le composant générique est encore utile pour l'interaction avec un partenaire dans un contexte différent (i.e industrie, région, etc.). Une règle de contexte associe chaque catégorie contextuelle aux valeurs différentes. Elle indique également quelles actions sont effectuées si un raccord (match) est apparu. Les actions peuvent être d'ajouter l'information, de demander ou éliminer l'information.

On a deux ensembles de règles : règles d'assemblage et règles de contexte. Les deux sont représentés en XML (DTD). Les règles d'assemblage sont, par exemple, ń CreateGroup ż, ń CreateElement ż, ń Rename ż,etc. Les règles de contexte sont, par exemple, ń Condition ż, ń Add ż, ń Subtract ż, etc. Le résultat d'un processus d'assemblage est le document sous la forme suivante :

<ELEMENT Document (Taxonomy+, Assembly, ContextRules ?, Component+)>
...

où ń Taxonomy ż pointe vers le contexte spécifique qui se combine avec les règles d'assemblage et de contexte.

Voici un exemple du document assemblé :

<?xml version="1.0" ?>
<!DOCTYPE Document SYSTEM "sid.dtd">
<Document>
<Taxonomy context="Geopolitical"
ref="http ://ebxml.org/classification/ISO3166">Region
</Taxonomy>
<Assembly name="PurchaseOrder" value="Geopolitical='United States'"/>
<ContextRules name="CalAer" value="Industry='Aerospace'
Geopolitical='United States'"/>
<Component name="PurchaseOrder" sequence="FollowedBy">
<Component name="Buyer" sequence="FollowedBy">

<Component name="Address" sequence="FollowedBy">
<Group sequence="Choice">
<Component name="BuildingName"/>
<Component name="BuildingNumber"/>
</Group>
<Group sequence="Choice">
<Component name="Floor"/>
<Component name="Suite"/>
</Group>
<Component name="City"/>
<Component name="State"/>
<Component name="ZIP"/>
<Component name="Country"/>
</Component>
</Component>
<Component name="Seller"/>
<Component name="Item" occurrence="+"/>
</Component>
</Document>

Les règles d'assemblage s'utilisent comme suit :

<!ELEMENT Assembly (Assemble+)>
<!ATTLIST Assembly
...
>
<!ELEMENT Assemble (CreateElement|CreateGroup)+>
<!– the name is the name of the schema that is created –>
<!ATTLIST Assemble
...
>
<!ELEMENT CreateGroup
(CreateGroup|CreateElement|UseElement|Annotation)+ >
<?xml version="1.0" ?>
<!DOCTYPE Assembly SYSTEM "assembly.dtd">
<Assemble name="PurchaseOrder" id="PO">
<CreateGroup>
<CreateElement location="UUID" id="Buyer">
<Name>Buyer</Name>
<Type>PartyType</Type>
<CreateGroup>
<UseElement name="Name">
</UseElement>
<UseElement name="Address">
<CreateGroup id="fred">
<CreateGroup type="choice">
<UseElement name="BuildingName">
</UseElement>

```
</CreateGroup>
...
<UseElement name="City">
</UseElement>
<UseElement name="State">
</UseElement>
...
</CreateGroup>
</UseElement>
</CreateGroup>
<Condition test="$Geopolitical='United States'">
<Rename from="address" to="addressUS"/>
<Rename from="Place" to="City"/>
<Rename from="address/County" to="State"/>
<Rename from="address/PostalCode" to="ZIP"/>
</Condition>
</CreateElement>
<CreateElement id="Seller" location="UUID">
<Name>Seller</Name>
<Type>PartyType</Type>
</CreateElement>
</CreateGroup>
...
</Assemble>
```

Les règles de contextes s'utilisent comme suit :

```
<!ELEMENT ContextRules (Rule+)>
<!ATTLIST ContextRules
version CDATA #IMPLIED
id ID #IMPLIED
idref IDREF #IMPLIED
>
<!ELEMENT Rule (Taxonomy+, Condition+)>
<!ATTLIST Rule
apply (exact|hierarchical) "exact"
order CDATA #IMPLIED
id ID #IMPLIED
idref IDREF #IMPLIED
>
<!ELEMENT Taxonomy EMPTY>
<!– this ref should be a URI –>
<!ATTLIST Taxonomy
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>
```

N.B : Avec ce mécanisme d'assemblage et de contexte, on va rencontrer au moins les deux problèmes suivants :

* Il est possible que deux règles soient appropriées au contexte mais on peut obtenir deux résultats conflictuels. Par exemple, une règle exigerait que si un acheteur est de la région U.S, les produits n'incluent pas les lignes d'articles dans la facture. L'autre indique soit que si un vendeur est en France, la description des produits sera incluse.

* Il est possible que deux règles soient appropriées au contexte mais on peut obtenir deux résultats différents selon l'ordre appliqué. Par exemple, une règle exigerait que si l'industrie d'un acheteur est automobile, la catégorie des produits sera ajoutée aux lignes d'articles de la facture. L'autre indique soit que si une entité d'information catégorique de produit existe et l'industrie d'un vendeur est chimique, un attribut serait ajouté à la catégorie du produit pour indiquer la toxicité du produit dans cette

catégorie. Si la toxicité est appliquée avant, l'attribut ne sera pas introduit (car la catégorie de produit n'existe pas encore).

2.2.5 Avantages

* Utiliser l'approche orientée-objet avec les outils d'UML (UMM) dans la modélisation de données et de transactions. Cette approche permettra de capturer une partie importante de la sémantique commerciale, y incluant 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 elle s'installe dans un 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 son entreprise.

* Améliorer l'interopérabilité sémantique grâce à la conception des composants de base et le mécanisme contextuel. Les composants de base permettent également la réutilisation mais l'objectif principal est toujours l'interopérabilité. Le mécanisme contextuel est un moyen capable de résoudre des problèmes traditionnels pour un partenaire qui doit faire du commerce avec plusieurs partenaires appartenant à des industries différentes. Les problèmes sont les suivants : la même donnée (le même concept) avec des noms différents ou avec des places différentes dans le document. La solution exige souvent une analyse des vocabulaires supportés par les partenaires pour arriver à des transformations entre eux. Le mécanisme contextuel, qui décrit la dérivation de chaque vocabulaire industriel spécifique, appliqué sur les composant de base peut automatiser ces transformations en remontant vers les composants de base dont les concepts dérivent. Il est évident que ce mécanisme permet une réduction considérable sur le coût d'analyse de vocabulaires. Le deuxième type des problèmes traditionnels est la même donnée (le même concept) se situant dans des processus différents ou interprétée par des cultures différentes. Pour ce type-là on peut également trouver une solution grâce au mécanisme contextuel avec la flexibilité du XML mais parfois la performance doit être sacrifiée (la complexité du document XML s'accroît si l'on veut éviter les ambiguïtés).

* Utiliser le langage de balise XML comme un format de transport des documents échangés. Cela permet de profiter des outils du XML (XSLT) pour faciliter la traduction entre XML et un autre format utilisé par les applications commerciales.

2.2.6 Limites

* La nature de l'approche orientée-objet ne permet pas de capturer suffisamment la sémantique dont le commerce a besoin. La relation généralisation - spécialisation entre les composants de base ne suffit pas pour exprimer la sémantique. Par exemple, un concept (Taxe) se situant dans une ligne d'article aurait potentiellement une sémantique différente de celle de même concept se situant dans la tête du document. Dans ce cas-là, il exige une ń connaissance ż 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 qu'ils deviennent très similaires malgré la différence des composants de base dont ils dérivent. Dans ce cas-là, la similarité ne peut pas être détectée.

* Le mécanisme contextuel 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 entraîne les problèmes cités dans la sous-section 2.2.4. Ces problèmes, tant que ces règles s'appliquent ń naïvement ż sur les composants de base, sont insurmontables. La solution a besoin d'une intégration cohérente des informations contextuelles dans les composants de base en conservant la réutilisation.

* Malgré les efforts pour organiser les composants de base, pour introduire le mécanisme contextuel, 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, d'interprétation des documents de façon ń en ligne ż. Pour cette raison, l'ebXML n'est toujours pas approché à l'échange transparent sémantique.

2.3 Modèle de langage formel

2.3.1 Motivation

Le modèle de langage formel s'est inspiré des lacunes du modèle EDI. Pour arriver à l'échange de données, deux acteurs du modèle EDI doivent négocier pour atteindre un accord d'échange (Figure ?) qui inclut la manière d'interpréter les messages. Dans ce cas, la majorité de connaissances afin d'interpréter les messages se prédéfinit dans des manuels d'utilisateurs. Ainsi, la sémantique des messages n'est pas véhiculée avec ceux-ci et cela empêche un nouvel acteur de participer au système d'échange.

Figure 4

Pour que deux acteurs d'échange se comprennent, il est nécessaire qu'ils parlent la même langue. De plus, pour valider les messages, recouvrir les activités du domaine et faire des inférences, c'est-à-dire, automatiser le traitement et la communication machine - machine, il faut formaliser ce langage-là. L'objectif de ce modèle est donc de supporter les transactions commerciales communes, permettre aux machines de composer les messages, de les transmettre et à la machine d'un nouvel acteur de les comprendre.

Un langage typique de cette direction est FLBC, développé par Steven O. Kimbrough et Scott A.Moore.

2.3.2 Base théorique

Les auteurs sont convaincus que le langage n'utilise qu'un lexique unique et publique parce que cela permet d'éviter des accords spécifiques sur la structure ou sur une interprétation propre des messages échangés. De plus, ils choisissent la logique de premier ordre (FOL) pour la base d'expression du langage.

Le langage FLBC se fonde sur cinq éléments (hypothèse) suivants :

* Speech act theory (SAT) : chaque message bien-défini, ou énoncé U, est conforme à un canevas F(P) (U(F(P)) où F est un indicateur de la force illocutoire, comme ń assert ż, ńpromiseż, ń request ż ; P est une proposition vrai - faux, s'appelle le contenu propositionnel d'une expression ayant la forme F(P) . La logique modale s'utilise pour ces expressions.

* P dans le canevas F(P) est récrit comme une phase S avec une locution nominale NP et verbale VP : P (S, S ((NP, VP)

* Event semantic s'utilise pour récrire VP sous la forme de la logique de premier ordre.

* Thematic rôles s'exprime les rôles des agents du VP.

2.3.3 Capture et représentation de la sémantique commerciale

Un lexique et un ensemble de règles de grammaire sont proposés pour le langage FLBC. La partie importante de la sémantique commerciale est déterminée dans ce lexique. Il présente les opérations normales comme ń pay ż, ń deliver ż, ń create ż, etc ; les VPs comme ń promise ż, ń assert ż, ń declare ż, ń invoice ż, etc ; les thematic rôles comme ń Addressee ż, ń Theme ż, ń K(x) : promise x is keep ż, ń T(x) : assertion x is true ż et un énorme dictionnaire (références) des objets commerciaux [voir 16].

Pour illustrer comment décrire un document commercial en utilisant ce lexique et ces règles (FLBC), on considère l'exemple suivant :

Un message EDI-simulé comme suit :

1. promise : 12345
2. date-time : 2001-11-13
3. from : s
4. to : r
5. deliver

a) goods : g
b) to : r
c) by : s
d) date-time : 2001-11-13 + day (2001-11-13 + 30)

On convertit ce message en celui de FLBC :

Promise(12345) (Speaker(12345,s) (Addressee(12345,r) (
Cul(12345,2001-11-13) (2 (? ő (K(12345) ((deliver(e) (
Agent(e, s) (Benefactive(e, r) (Sake(e, 12345) (Theme(e,g)
((Cul(e, t) (((t, +(2001-11-13, day(2001-11-13+ 30))))))
//explication ?

2.3.4 Avantages

* La formalisation avec FOL du FLBC permet de faciliter la validation des documents FLBC.

2.3.5 Limites

* Le lexique unique du FLBC est une grande difficulté pour le commerce actuel. Un bon système doit être capable de répondre à l'hétérogénéité des concepts commerciaux, des cultures (voir la section 2.2)

* L'information représentée par la logique modale du FLBC n'est pas encore exploitée suffisamment. Autrement dit, cet information n'est pas formalisée pour permettre de la traiter automatiquement.

2.4 Synthèse sur les modèles

Les trois modèles étudiés montrent les enjeux auxquels est confronté l'échange de données sous forme de document en tant que domaine de recherche. L'échange transparent sémantique pose des problèmes de représentation et d'intégration sémantique. Il s'agit d'introduire un nouveau concept (une sémantique) dans les documents par un acteur. La composition et interprétation des documents échangés par l'EDI s'appuient sur une sémantique prédéfinie par les dictionnaires et accords d'échange préalables. D'une part un acteur ne peut introduire une modification d'un concept existant ou un nouveau sans accord précédent, d'autre part le système d'échange n'est qu'accessible pour les membres du groupe qui utilisent le même dictionnaire, les mêmes accords d'échange.

L'ebXML a tiré les leçons de l'EDI et facilite la participation d'échange d'un nouvel acteur 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 facilitera la validation des documents échangés et le traitement automatique des connaissances. Celle-ci avec le mécanisme d'intégration permettra d'introduire de façon ń en ligne ż un nouveau concept dans les documents. Il s'agit de la possibilité de représentation 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, des cultures commerciaux. En outre, la représentation sémantique en reprenant des éléments de celle du langage naturel pose d'autres problèmes, par exemple, la formalisation de ces éléments.

4 Logique descriptive et ebXML

L'étude de la DL ouvre une direction de recherche qui essaie de répondre à la question : la DL peut-elle être la base formelle pour la représentation sémantique de l'ebXML permettant de franchir les problèmes cités ? Autrement dit, quel est le lien entre la base DL et l'ebXML ? La capacité expressive et les algorithmes d'inférence développés de la DL nous encouragent à rechercher la réponse dans ce contexte.

Il est évident que la question vise aux deux parties examinées de l'ebXML : processus commerciaux et composants de base. Pour la première portant la majorité de la sémantique, puisque le processus de la capture est très complexe, les connaissances acquises restent encore vagues. La formalisation de ces connaissances en utilisant la DL à partir des fiches descriptives exige donc d'identifier les concepts primitifs et les relations entre eux. Ces dernières permettent de définir les concepts complexes DL (concepts définis) grâce au langage descriptif. Des difficultés éventuelles sont la découverte, l'identification des concepts primitifs/définis DL et les relations à travers les descriptions en langage naturel dans les fiches (voir fiches 5,11) , la transformation des processus dynamiques décrits par les diagrammes d'activité ou d'état (UML) vers des concepts DL (voir fiche 8).

Pour la deuxième, grâce à des descriptions plus simples de informations dans le dictionnaire des composants de base, le problème de formalisation cité devient moins difficile. Et pourtant, les composants de base servent à composer des documents commerciaux. Nous supposons que la sortie du processus précédent (la capture de la sémantique par fiches) comporte une définition du document et des informations contextuelles bien définies en DL. Alors, la composition (interprétation) du document implique deux processus : l'identification des composants sans sémantique (Core Component Type) à l'aide de la définition du document et la mise à jour de ces composants pour obtenir les composants avec la sémantique (Basic Information Entity, Aggregate Information Entity) à l'aide des informations contextuelles.

La discussion ci-dessus se déroule dans l'hypothèse où la sémantique est bien déterminée avant d'échanger, c'est-à-dire que les processus commerciaux, les composants de base sont partagés par les acteurs. La composition et l'interprétation des documents échangés s'effectuent avec la sémantique fixe tout au long de l'échange.

Pour aller au-delà, nous essayons de généraliser le problème en citant une autre question posée dans un scénario d'échange très demandé : le modèle abordé fonctionne-t-il encore en cas d'introduction d'un nouveau concept dans les documents ou d'une modification des informations contextuelles ? Autrement dit, le modèle fonctionne avec une sémantique changeante (modifiée ou ajoutée). L'introduction d'un nouveau concept par un interlocuteur dans une conversation quotidienne est un exemple de ce scénario. La compréhension du concept est assurée par une explication qui se transmet de l'un à l'autre. En particulier, une intégration des deux bases DL sur tous les concepts définis (supposons que les interlocuteurs aient les mêmes concepts primitifs) serait exigée.

En bref, la recherche d'un modèle formel pour l'ebXML nous entraîne à procéder aux travaux suivants :

i) Proposition des principes pour organiser Tboxe et Aboxe des bases DL de l'ebXML. Ces bases devraient se représenter en XML. Ainsi, cette tâche définit des balises pour les éléments du langage descriptif : concept, rôle primitif ; concept, rôle défini ; attribut ; les constructeurs, etc. Le système CLASSIC (développé par AT&T) est une référence.

ii) Une analyse des fiches descriptives de l'ebXML pour identifier les concepts, les rôles primitifs et les relations entre eux décrites par des concepts, rôles définis. Ces descriptions sont transformées vers une base DL, éventuellement, à l'aide d'un outil. Cette tâche a pour objectif d'estimer la capacité expressive de la sémantique commerciale du model. Elle permet également les préparatifs du prototype du modèle.

iii) Automatisation du processus de composition (interprétation) des documents commerciaux en supposant que la définition du document et les informations contextuelles soient bien définies dans la base DL et qu'une base DL des composants de base soit construite. Cette tâche peut se diviser en sous-tâches comme suit :

a) recherche des composants de base (libre de contexte) à partir de la définition du document. Avec la base DL des composants partagée et une description du concept dans la définition, cette sous-tâche permettrait d'arriver à une des possibilités suivantes : l'acceptation du concept, l'acceptation du concept avec une modification du nom ou de la description, l'ajout d'un nouveau concept.

b) application d'informations contextuelles sur les composants trouvés. Cette sous-tâche implique la recherche d'une représentation des informations contextuelles (une table de catégorie de contexte-valeur et des actions correspondantes selon l'ebXML), un mécanisme d'application de ces informations via cette représentation sur les composants sans conflit de données.

iv) Intégration de deux bases DL. Cette tâche serait exigée en cas de traitement de la sémantique changeante.

Il est très probable que ces travaux font apparaître des difficultés potentielles que nous n'avons pas pu identifier pour ce moment. Une analyse de ces tâches et une solution proposée vont se présenter dans les rapports suivants.

5 Annexe

5.1 Exemple de capture sémantique commerciale

Nous présentons un exemple de l'analyse en s'appuyant sur les fiches descriptives pour le modèle commercial ń Direct to Customer Drop Ship Retail ż. Cependant, nous ne décrivons que la collaboration ń Commande d'Achat ż de ce modèle pour

illustrer comment utiliser ces fiches. Cet exemple est une version simplifiée qui est extraite de ń Business Process Analysis Worksheets and Guidelines ż [10]

5.2. Problématique de l'ebXML et logique de description

This report introduces semantic representation of ebXML and its problems. These problems are identified and analysed when we attempt to investigate the functionality specifications of ebXML. Our study about the way of representing the knowlege allows to identify used formalisms in ebXML. They are UML, lexicon and production rules. Some of them possess informal semantic. A formalization of the semantics is necessary because the functionality specifications of ebXML require manipulations on the captured semantic. Therefore, we propose a hybrid formal representation in which the semantics of these formalisms are encapsulated. In addition, data interchanges in ebXML between heterogeneous partners are based on the different lexicons. Hence, a reoganization of the lexicons forming the relevant ontologies and an integration of them are required.\\%

Description Logic is considered as expressive formalism for semantic representation. It is introduced on behalf of the unified formalism for the formalization of the semantics. Some extensions, however, are added in order to increase its expressiveness for requirements of ebXML. Finally, that formalization implies a mapping between the semantics of the formalisms used by ebXML and the extension of description logic.

\\%

\\%

\textbf{Keywords :} ebXML, UML, commerce semantic, production rules, hybrid representation, semantic transparency, description logic.

\end{abstract}

\section{Introduction}

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\footnote{Electronic Data Interchange} montre des difficultés infranchissables pour les acteurs dont les moyens sont modérés \cite{1}. 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 plus riches sur Internet. Dans ce contexte, l'ebXML\footnote{Electronic Business XML} est créé et il semble qu'il répondrait à ces besoins. Et pourtant, nous n'avons pas tous les outils nécessaires pour réaliser les spécifications de l'ebXML. L'un des outils manquants est un mécanisme capable de représenter la sémantique commerciale impliquée par les formalismes utilisés. Un tel mécanisme devrait faciliter la réalisation de ces spécifications, par exemple la composition des documents commerciaux, des manipulations sur les processus commerciaux, l'échange de données transparent sémantique, etc.\\%

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\footnote{Unified Modeling Language} 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 l'UMM\footnote{UN/CEFACT Modeling Methodology} , 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 cette limite 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 de formalismes utilisés n'est pas établie automatiquement. Il existe des efforts de l'ebXML pour surmonter ces difficultés, par 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 partager, les difficultés se multiplieront.\\%

Il existe déjà des manières de représenter la sémantique de langage, par exemple, le réseau sémantique, le graphe conceptuel, etc. mais aucune parmi elles ne donne des services d'inférence efficaces sur la sémantique capturée. La logique de description (DL), qui est inspirée du réseau sémantique et considérée comme un fragment décidable de la FOL\footnote{First Order Logic} , montre une capacité expressive suffisante (moyennant une extension) par le fait qu'elle représente la sémantique du modèle entité - relation \cite{2}. Ainsi, une extension de la DL pour la sémantique du modèle orienté - objet est en cours de recherche. Cela nous permet de penser à développer des ontologies de l'ebXML en utilisant la DL pour la formalisation de la sémantique. Ces ontologies avec un mécanisme d'intégration établissent une base sur laquelle l'échange de données de façon transparente sémantique se fonde.\\%

En sachant que nous n'arrivons jamais à capturer \textit{totalement} une sémantique impliquée par un langage (langage naturel, langage de modélisation, etc.), la formalisation d'une sémantique vise toujours à un ensemble de services d'inférences dont les spécification ebXML ont besoin. Cependant, notre approche se concentre sur les formalismes utilisés (UML, règles de production, etc.), la formalisation obtenue sera encore utilisée par d'autres applications dans lesquelles ces formalismes et services d'inférences sont exigés.\\%

Le reste de ce rapport sera organisé en sections. Dans la deuxième section, nous commençons par présenter la représentation des connaissances commerciales de l'ebXML et un exemple illustrant comment découvrir, définir un processus commercial à partir des modèles de données UMM et comment composer un document commercial à partir de composants de base et des informations contextuelles. Une analyse sur les formalismes utilisés et sur la relation entre eux achève la première sous-section. La deuxième sous-section présente un scénario d'échange avec l'ontologie partagée ebXML dans lequel le premier problème se pose : la nécessité d'une représentation hybride pour la sémantique ebXML. Le deuxième scénario d'échange qui introduit des ontologies locales dans le système fait apparaître le deuxième problème :

échanger de façon transparente sémantique pour les acteurs. Une petite synthèse sur les problèmes cités termine la section.\\%

La troisième section commence par la présentation de la DL. Nous proposons également quelques facteurs ajoutés à la DL afin de répondre à la représentation de la sémantique de diagrammes UML. Le reste de la section se concentre sur la transformation de la sémantique UML vers des expressions DLs. Une brève discussion sur la transformation des règles de production vers des expressions DLs achève la section.

\section{Problématique de l'ebXML}
\subsection{Sémantique de l'ebXML}
\subsubsection{Composants de l'ebXML portant la sémantique commerciale}

Une description brève des composants de l'ebXML portant la sémantique commerciale est présentée dans \cite{1}. La description en détail sur ces composants se trouve dans \cite{10}, \cite{12} et \cite{13}. Puisque notre objectif est de découvrir les formalismes utilisés par ebXML, nous ne fournissons qu'une vision sur la relation au niveau de la représentation de connaissances entre l'UML, UMM et ebXML. Une version de cette vision est présentée par la figure \textit{Fig. 1}.

\begin{figure}[h]
\begin {center} \includegraphics{d :/travaux/pub01-21.eps}
\end{center}
\caption{\textit{Relation entre UML, UMM et ebXML}}
\end{figure}

L'ebXML propose un ensemble de fiches descriptives (\textit{worksheet}), les diagrammes d'état et d'activité conformément à l'UMM permettant de représenter plus pratiquement les connaissances commerciales (\textit{flèche 1}). A partir de ces fiches, l'utilisateur peut définir ses processus commerciaux décrivant ses affaires. L'ebXML construit deux parties importantes portant la sémantique. La première partie, appelée \textit{Business Process Specification Schema} (BPSS), décrit un sous-ensemble de la sémantique (\textit{flèche 3}). BPSS a pour objectif de fournir des informations de configurations du système d'échange en durée d'exécution (\textit{flèche 8}). L'utilisateur peut extraire des fiches descriptives des éléments et relations (processus commerciaux, modèle d'informations) conformément à BPSS afin de les introduire aux \textit{Business Process Specification} (BSP). La deuxième partie, appelée \textit{composants de base} (\textit{Core Component} CC), décrit les composants libres de contexte permettant la réutilisation pour la composition de documents (\textit{flèche 5}). Ces composants sont dérivés du lexique et du méta-modèle de l'UMM (\textit{flèche 4}). Le document commercial se compose de composants de base et d'informations contextuelles extraites de fiches descriptives (\textit{flèche 5,6,7}). Les connaissances suivant les \textit{flèches 6,7} sont représentées par les règles de production.\\%

Une autre version plus lisible au niveau de la participation des formalismes est présentée par la figure \textit{Fig.2}.\\%

Le formalisme UML ne peut que capturer une partie des connaissances commerciales. Il existe des travaux, \cite{7} et \cite{8}, qui proposent un formalisme ayant l'ambition de représenter la totalité des connaissances commerciales mais ils

sont confrontés à plusieurs problèmes à franchir au niveau de la théorie. Une contribution de l'ebXML est de définir BPSS, un sous-ensemble sémantique de l'UMM, permettant aux utilisateurs à la fois de configurer des interfaces, logiciels du système d'échange en durée d'exécution (chorégraphie des transactions commerciales, etc.) et de définir BPS conformément à ses affaires. BPSS est décrit par deux versions : l'une en diagrammes de classes UML, l'autre en XML. En outre, une autre contribution de l'ebXML est de construire les composants de base réutilisables CC. Ces composants de base avec les règles de production portant les informations contextuelles permettent de composer les documents commerciaux. Ces règles résultent du modèle d'information UMM (fiches descriptives).\\%

\begin{figure}[h]
\begin {center} \includegraphics{d :/travaux/pub01-22.eps}
\end{center}
\caption{\textit{Formalismes utilisés par ebXML}}
\end{figure}

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 se limite par la spécification ebXML aux catégories contextuelles autorisées. Rien n'assure que ces catégories soient exhaustives.

\subsubsection{Exemple sur la capture sémantique commerciale}

Nous considérons un exemple sur le modèle commercial : \textit{Direct-To-Customer-Drop-Ship-Retail-Model} montrant comment l'ebXML capture et représente la sémantique d'un processus commercial (\textit{voir l'annexe}). Cet exemple illustrera également l'utilisation des informations extraites du processus commercial pour la composition/interprétation du document commercial. Pour éviter des détails lourds, nous ne présentons que des fiches simplifiées.\\%

Dans un premier temps, on utilise la \textit{fiche 1, Fig. 6} pour décrire le type de processus. Chaque type de processus est une série de processus formant un secteur commercial. L'identification de ces processus s'effectue dans les fiches : \textit{Gestion de commandes d'achat, Commande de vente, Livraison de marchandises}, etc. Il y a 5 acteurs participant aux processus d'échange : \textit{Détaillant, DSVendeur, MétierTranport, AutoritéCrédit} et \textit{Consommateur}. La figure \textit{Fig. 9} indique les processus d'échange entre les acteurs et leur rôle. Nous allons examiner le processus \textit{Gestion de commandes d'achat} effectué par \textit{Détaillant} et \textit{DSVendeur} (\textit{Fig. 8}). S'il existe une commande de vente effectuée par un consommateur, le processus sera déclenché. Un processus commercial peut se constituer de plusieurs collaborations. Mais le processus \textit{Gestion de commandes d'achat} ne comprend qu'une collaboration \textit{Create-Vendor-Purchase-Order}. La \textit{fiche 2}, la \textit{Fig. 10} et la \textit{Fig. 11} s'utilisent pour décrire la collaboration. Dans cette phase de la capture de connaissances, l'aspect de temps entre en compte. Cela entraînerait à un ajout de l'aspect de temps au formalisme utilisé. Cet aspect est présent dans les diagrammes d'états et d'activités d'UML.\\%

La collaboration possède la transaction \textit{Create-Vendor-Purchase-Order}. La \textit{fiche 3}, la \textit{Fig. 12} et la \textit{Fig.13} s'utilisent pour décrire la transaction \textit{Create-Vendor-Purchase-Order}. Chaque transaction correspond

toujours à un document échangé. Dans ce cas, les deux acteurs qui participent à la transaction sont : \textit{Détaillant} et \textit{DSVendeur}. Le document qui est envoyé du détaillant au DSVendeur décrit les informations sur l'acheteur, le vendeur, les produits demandés par l'acheteur, etc.\\%

A partir des connaissances commerciales acquises qui sont contenues dans les fiches, la composition du document commence par une proposition d'un squelette du document (\textit{fiche 5, Fig. 15, Fig. 16}) et par la recherche des composants de base appropriés. Les informations contextuelles sont également déterminées (\textit{fiche 4, Fig. 14}). Ces informations sont représentées par des règles de production, par exemple :\\%

\begin{itemize}

\item[]

i) Si un acheteur est de la région U.S., les descriptions des produits n'incluront pas les lignes d'articles dans la commande.

\item[]

ii) Si un vendeur est en France, la description des produits sera incluse dans la commande.

\item[]

iii) Si l'industrie d'un acheteur est l'automobile, la catégorie des produits sera ajoutée aux lignes d'articles de la commande.

\item[]

iv) Si une entité d'informations catégorique de produits 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.

\end{itemize}

Après avoir appliqué ces règles sur les composants de base, on obtiendra les compsants avec la sémantique déterminée. Le document final se compose de ces composants.\\%

L'application des règles contextuelles sur les composants de base trouvés peut poser des problèmes, par exemple, un conflit sur le résultat (si l'on applique les règles i et ii sur une ligne de produit dans le document) ou le résultat obtenu dépend de l'ordre appliqué des règles (si l'on applique les règles iii et iv sur une ligne de produit dans le document). Si la règle iv est appliquée avant, l'attribut de toxicité ne sera pas introduit car la catégorie de produit n'existe pas encore.

\subsubsection{Synthèse sur les formalismes utilisés}

A travers la présentation des composants de l'ebXML dans les sous-sections précédentes, nous nous rendons compte que la sémantique commerciale de l'ebXML se compose de celle de diagrammes UML, d'une ontologie incluant le CC. 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, un diagramme de classes UML s'utilise pour décrire le méta-modèle du CC permettant d'y introduire les informations contextuelles. Les composants de base avec la sémantique sont instanciés à partir de ces diagrammes. BPSS est également représenté par un diagramme de classes dans lequel la sémantique des collaborations, des transactions d'un processus commercial est capturée. Les diagrammes d'états et d'activités s'utilisent pour exprimer les comportements d'objets et les interactions entre eux.

Une ontologie définit tous les termes utilisés dans ces diagrammes. En outre, la composition des documents commerciaux se basant sur ces connaissances représentées et sur le mécanisme de contexte exigerait sans doute un autre formalisme pour représenter les règles de production. Chaque règle se représente en deux parties dont la conclusion inclut une \textit{action}. Le formalisme souhaité pour ces règles devrait être capable de franchir les problèmes cités (conflit de résultats, dépendance de l'ordre appliqué) dans l'exemple.\\%

Ces formalismes se montrent de quelques avantages en permettant de capturer la majorité des connaissances. Néanmoins, ils nous mettent face à une grande difficulté : la sémantique décrite par le langage naturel. La formalisation de cette sémantique est indispensable car les spécifications de fonctionnalités de l'ebXML nécessitent des manipulations sur la sémantique, c'est-à-dire les services d'inférences, par exemple, gestion des processus commerciaux, composition des documents commerciaux, intégration des lexiques, etc. De plus, une sémantique formelle partagée et compatible à celles de formalismes utilisés est exigée pour répondre à l'hétérogénéité de ces formalismes. Il est évident que l'on ne peut pas capturer une sémantique en totalité, la formalisation de la sémantique abordée doit viser toujours aux fonctionnalités concrètes et bien définies, par exemple, les spécifications de fonctionnalités de l'ebXML mentionnées.\\%

Dans la sous-section suivante, nous essayons d'analyser encore plus les problèmes cités en mettant l'accent sur la transparence sémantique dans le contexte d'échange de données ebXML entre deux acteurs.

\subsection{Scénario d'échange avec l'ontologie partagée : problème de la représentation hybride}

Nous présentons un scénario simple d'échange de documents utilisés dans un système supportant plusieurs acteurs. Ce scénario correspond à la sémantique statique qui ne change pas tout au long du processus d'échange.

Les deux composants de l'ebXML : les processus commerciaux et composants de base couvrant la sémantique commerciale s'appuient sur une ontologie partagée (\textit{Fig. 3}).

\begin{figure}[h]
\begin {center} \includegraphics{d :/travaux/pub01-31.eps}
\end{center}
\caption{\textit{Scénario d'échange avec une sémantique identique et statique}}
\label{fig 3-1}
\end{figure}

L'ebXML utilise ce scénario et propose un mécanisme contextuel pour résoudre partiellement le problème des différences sémantiques impliquées par plusieurs vocabulaires : un vocabulaire neutre sous forme des composants de base engendre différents vocabulaires en y réunissant des informations contextuelles. L'essentiel de ce scénario est d'atteindre les mêmes connaissances ou la même sémantique sur les processus concernés avant le premier échange et cette sémantique ne change pas tout au long de l'échange. Par exemple, l'acteur A envoie à l'acteur B une commande d'achat qui contient un produit \textit{Ordinateur-Superbe-Qualité}. Dans ce scénario ce terme doit être défini dans l'ontologie partagée (composants de base avec des informations contextuelles) pour que les deux acteurs se comprennent. \\%

Afin de supporter différents vocabulaires, l'ebXML construit un diagramme de classes UML sous la forme de structure hiérarchique. Les composants de base qui sont instanciés à partir de la racine de ce diagramme sont les définitions des vocabulaires sans information contextuelle, appellés \textit{Core Component Type}. Chaque fois le processus d'échange est défini avec la disponibilité des informations contextuelles, les descendants \textit{Basic Information Entity} et \textit{Aggregate Basic Information Entity} sont déterminés par ces informations contextuelles. Ainsi, les vocabulaires appropriés à ce contexte se produisent. Par exemple, un \textit{Core Component Type} : \textit{adresse} est défini sans champs \textit{etat}. Une fois qu'un partenaire d'échange est déterminé comme étant des Etat-Unis, le champs \textit{etat} y est rajouté. L'ebXML propose d'utiliser 5 types de contextes : \textit{Processus, Produit, Géographie, Industrie, Culture}, c'est-à-dire que les modifications possibles sur les vocabulaires doivent se limiter à ces catégories. \\%

Par ailleurs, les spécifications ebXML demandent un mécanisme capable de gérer les processus commerciaux qui sont définis comme l'ensemble des diagrammes UML et le lexique des termes. En particulier, le système devrait permettre de vérifier si un processus existe dans le système, un processus est défini à partir des processus existants, etc. C'est pourquoi la réalisation de ces spécifications exige de manipuler la sémantique des formalismes.

Pour répondre à cet exigence, l'ontologie partagée doit se fonder sur un formalisme générique unifiant l'UML, le lexique, les règles de production ensemble. Autrement dit, une représentation hybride sera la solution à tel système. Et pourtant, une telle représentation implique un problème important : la définition de la sémantique commune et cohérente, qui est partagée par les formalismes utilisés.\\%

En bref, cette analyse nous amène à tenir compte des tâches suivantes pour le système ebXML : formaliser la sémantique ebXML qui vient de celles des formalismes : UML, règles de production ; introduire des services d'inférences sur la sémantique représentée supportant les fonctionnalités attendues. Une approche proposée pour ces tâches est de choisir un formalisme générique dans lequel le formalisme UML s'immerge. D'autres formalismes s'y rajoutent au fur et à mesure. Nous revenons à ce sujet dans la section suivante.

\subsection{Scénario d'échange avec l'ontologie locale : problème de la transparence sémantique}

L'eCommerce 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 de partenaires. L'ambition de l'ebXML est de répondre aux besoins du marché à tel niveau. Il sera infaisable si tous les acteurs se restreignent à utiliser le même vocabulaire avec des compréhensions identiques sur les termes dans 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 \textit{"en ligne"}. C'est pourquoi le fait de supporter un scénario d'échange avec la sémantique dynamique est indispensable pour l'ebXML (\textit{Fig. 4}).

\begin{figure}[h]
\begin {center} \includegraphics{d :/travaux/pub01-32.eps}

\end{center}
\caption{\textit{Scénario d'échange avec une sémantique dynamique}}
\label{fig 3-2}
\end{figure}

L'essentiel de ce scénario est d'introduire une ontologie locale pour chaque acteur permettant de définir les termes propres. La relation entre les ontologies locales et l'ontologie partagée (elle contient les termes de base) pose des problèmes essentiels de ce scénario. On reprend l'exemple ci-dessus. Le produit \textit{Ordinateur-Superbe-Qualité} est défini dans l'ontologie OA en XML comme suit (OP : Ontologie Partagée, OA : Ontologie de A ; OB : Ontologie de B)\\%

\textit{
\tab <OA : Ordinateur-Superbe-Qualité>\\%
\hugetab <OP : Processeur> " OA : Processeur-Performant "\\%
\hugetab </OP : Processeur>\\%
\hugetab <OP : Mémoire> > 256Mo </OP : Mémoire>\\%
\hugetab <OP : Disque> SCSI </OP : Disque>\\%
\hugetab <OA : Ecran> " OA : Résolution- Elevée " </OA : Ecran>\\%
\tab </OA : Ordinateur-Superbe-Qualité>\\%
}

Quand l'acteur B reçoit le terme \textit{Ordinateur-Superbe-Qualité}, B ne le comprend pas car ce terme n'est défini ni dans l'OB ni dans l'OP (B ne peut comprendre " OA : \textit{Processeur-Performant}" ou " OA : \textit{Résolution-Elevée}"). Un appariement entre l'OA, l'OB avec la référence vers l'OP est exigé pour déterminer ce terme dans l'OB. Ce fait implique deux phases : construction du terme (à partir des informations externes : OA, OP) et appariement entre cette construction et l'OB pour obtenir la définition du terme dans l'OB. Il est probable que les deux définitions du terme sont différentes au niveau de la structure mais elles doivent être identiques ou ne pas être conflictuelles au niveau de la sémantique. Par exemple, l'OA définit le terme \textit{Processeur-Performant} comme suit :\\%

\textit{
\tab <OA : Processeur-Performant>\\%
\hugetab <OP : Marque> Intel-Pentium </OP : Marque>\\%
\hugetab <OP : Vitesse> > 1GHz </OP : Vitesse>\\%
\hugetab <OP : Cache> > 1Mo </OP : Cache>\\%
\hugetab <OP : NbdePro> > 1 </OP : NbdePro>\\%
\tab </OA : Processeur-Performant>\\%
}

OB définit les termes \textit{Processeur-De-Qualité-Elevée, Processeur-Multi, Processeur-rapide} comme suit :\\%

\textit{
\tab < OB : Processeur-De-Qualité-Elevée>\\%
\hugetab <OP : Marque> Intel-Pentium </OP : Marque>\\%
\hugetab <OP : Vitesse> >1GHz </OP : Vitesse>\\%
\hugetab <OP : Cache> >1Mo </OP : Cache>\\%
\tab </OB : Processeur-De-Qualité-Elevée>\\%
\\%

\tab <OB : Processeur-Multi>\\%
\hugetab <OP : Marque> Intel-Pentium </OP : Marque>\\%
\hugetab <OP : NbdePro> >1 </OP : NbdePro>\\%
\tab </OB : Processeur-Multi>\\%
\\%
\tab <OB : Processeur-Rapide>\\%
\hugetab <OP : Vitesse> >1GHz </OP : Vitesse>\\%
\tab </OB : Processeur-Rapide>
}\\%

L'établissement de la définition de \textit{Processeur-Performant} dans l'OB nécessite des inférences sur les connaissances de l'OB, par exemple, la subsomption, la conjonction (basé sur l'OP partagée),etc. Ces services d'inférences permettent de découvrir les mêmes concepts portant des noms différents et de définir un nouveau concept à partir de concepts plus génériques. Avec ces services, il existe deux candidats dans l'OB pour \textit{Processeur-Performant} : \textit{Processeur-De-Qualité-Elevée} et un nouveau concept défini par l'intersection entre \textit{Processeur-Multi} et \textit{Processeur-Rapide}.\\%

Une solution plus générique est l'intégration des deux ontologies l'OA et l'OB en utilisant l'approche \textit{assertionnelle}. Cette approche s'appuie sur les informations structurelles, par exemple, l'équivalence ou la subsomption des concepts, la transformation entre des concepts avec des niveaux abstraits différents. Il existe sur l'ontologie de la logique de description des algorithmes développés très récemment permettant des inférences non-standard. Ces inférences facilitent l'intégration assertionnelle de deux ontologies mais il nous reste beaucoup de travail à faire pour que les problèmes essentiels soient vraiment résolus.\\%

Nous nous rendons compte facilement que ce scénario implique des échanges des données de façon transparente sémantique pour les acteurs. La nécessité de la sémantique formelle qui est affirmée par le premier scénario se justifie également dans le deuxième scénario. En effet, l'intégration d'ontologies exige de manipuler la sémantique formalisée pour découvrir la subsomption, l'équivalence entre les concepts ou transformer l'un en l'autre. Il existe actuellement deux approches pour le problème d'échange transparent sémantique. La première approche se fonde sur un vocabulaire unique avec la sémantique associée bien déterminée comme dans \cite{7} et \cite{8}. Le travail restant à faire est de développer un langage formel, qui porte cette sémantique, permettant de représenter presque toutes les connaissances commerciales. Alors tous les acteurs parlent la même langue avec la même base de connaissances pour décrire le monde, donc ils se comprennent. Cette approche fait face aux difficultés suivantes : formalisation des connaissances sur le nécessaire, le probable et le contingent ; adaptation du formalisme choisi à la nature de la représentation hybride de connaissances en conservant cohérence et efficacité ; la construction d'un vocabulaire unique à partir de vocabulaires divers existants, etc. La deuxième approche ne traite que les connaissances prévisibles, sans répudiation dans les transactions commerciales. Elle utilise plusieurs formalismes avec une sémantique formelle commune. Des services d'inférences doivent se développer pour manipuler les connaissances représentées et

apparier des ontologies différents. Cette approche est confrontée également aux problèmes suivants : transformation de plusieurs formalismes vers un formalisme unifié portant la sémantique commune ; intégration des ontologies, etc.\\%

En bref, nous avons identifié et analysé les problèmes posés par les spécifications ebXML pour arriver à proposer les directions de recherche appropriées. Nous mettons l'accent sur les idées principales par le résumé suivant :

\begin{itemize}

\item[$\bullet$]

L'ebXML utilise l'UML (diagrammes de classes, d'états et d'activité), le lexique et l'ensemble de règles de production pour représenter des connaissances commerciales. Les spécifications ebXML ont proposé les fonctionnalités exigeant des manipulations sur la sémantique de ces formalismes. Cela nous entraîne à procéder aux tâches suivantes :

\begin{itemize}

\item[\labelitemii]

choisir et développer un formalisme générique, dont la sémantique est formelle, capable d'encapsuler les formalismes utilisés par l'ebXML.

\item[\labelitemii]

transformer la sémantique des diagrammes de classes UML vers celle de formalisme choisi.

\item [\labelitemii]

transformer la sémantique des diagrammes d'états et d'activités UML vers celle de formalisme choisi.

\item[\labelitemii]

concevoir l'ontologie correspond au lexique en utilisant le formalisme choisi.

\item[\labelitemii]

transformer la sémantique de l'ensemble de règles de production vers celle de formalisme choisi.

\end{itemize}

Ce problème et ces tâches sont inspirés du premier scénario d'échange.

\end{itemize}

\begin{itemize}

\item[$\bullet$]

Le deuxième scénario implique le problème d'échange transparent sémantique. Cela nous entraîne à développer des techniques permettant d'intégrer les ontologies. Il est évident que les ontologies sont conçues par le formalisme choisi dans l'étape précédente.\\%

\end{itemize}

\section{Logique de description pour la sémantique de l'ebXML}

Une description générale sur la DL est présentée dans \cite{1}. Dans ce rapport nous allons aborder une extension existante de la DL, $\mathcal{ALCQI}$, sur laquelle le formalisme proposé, $\mathcal{DLX}$, sera développé. Ce formalisme est présenté dans l'espoir de pouvoir simplifier la formalisation de la sémantique hétérogène de l'ebXML en conservant l'essentiel de chaque formalisme utilisé. Nous allons également expliquer comment ce formalisme générique proposé avec sa sémantique formelle répond à la réalisation des fonctionnalités décrites par ebXML. Puisque le

langage correspondant à la logique de description a une sémantique définie formellement, la formalisation d'une sémantique d'un formalisme utilisé par ebXML implique la représentation de cette sémantique en ce langage. Les sous-sections suivantes seront donc consacrées à décrire respectivement, $\mathcal{DLX}$ en tant que le formalisme générique, la transformation de diagrammes UML vers $\mathcal{DLX}$, et finalement la transformation de règles de production vers $\mathcal{DLX}$.

\subsection{Syntaxe et sémantique de $\mathcal{ALCQI} \textbf{ - Formalisme } \mathcal{DLX}$}

Si l'on dénote \textit{A, P, C, }et \textit{R} respectivement, à un concept atomique, un rôle atomique, un concept arbitraire et un rôle arbitraire, le langage correspondant à $\mathcal{ALCQI}$ est défini comme suit :\\%

\\%

$

\begin{array}{rll}

C,C'& \longrightarrow & \textit{A } | \enspace \neg\textit{C }

| \textit{ C } \sqcap \textit{ C' } | \textit{ C } \sqcup \textit{ C' } | \\%

& & \forall \textit{R.} \textit{C } | \enspace \exists \textit{R.} \textit{C }|

\enspace \exists ^{\geq n}\textit{R.} \textit{C } | \enspace \exists

^{\leq n}\textit{R.} \textit{C} \\%

\textit{R} & \longrightarrow & \textit{P } | \textit{ P}^{-}

\end{array}

$\\%

Nous utilisons également les abréviations suivantes :\\%

$

\begin{array}{rll}

\bot & \textup{pour} & \textit{A} \sqcap \neg\textit{A} \\%

\top & \textup{pour} & \textit{A} \sqcup \neg\textit{A} \\%

\exists ^{=n}\textit{R.} \textit{C} & \textup{pour} &

\exists ^{\geq n} \textit{R.} \textit{C}\enspace \sqcap \enspace

\exists ^{\leq n} \textit{R.} \textit{C} \\%

\exists ^{\geq n} \textit{R} & \textup{pour} &

\exists ^{\geq n} \textit{R.} \top

\end{array}

$\\%

La sémantique de $\mathcal{ALCQI}$ est définie par l'\textit{interprétation} $\mathcal{I} = (\Delta^{\mathcal{I}}, ^{.\mathcal{I}}) $ où $ \Delta^{\mathcal{I}}$ est un ensemble fini, non vide (domaine de $\mathcal{I}$) et $^{.\mathcal{I}}$ est une fonction (fonction d'interprétation de $\mathcal{I}$). Si les ensembles des concepts et celui des rôles atomiques sont respectivement dénotés par $\mathcal{A}$ et $\mathcal{P}$, les propriétés suivantes de l'interprétation $\mathcal{I}$ sont vérifiées :\\%

\\%

$

\begin{array}{rcl}

(\neg \textit{C}) ^{\mathcal{I}} & = & \Delta^{\mathcal{I}}\setminus

\textit{C}^{\mathcal{I}} \\%

```
( C _{1} \sqcap C _{2})^{\mathcal{I}} & = & \textit{C}^{\mathcal{I}}_{1}
\cap \textit{C} ^{\mathcal{I}}_{2} \\%
( C _{1} \sqcup C _{2})^{\mathcal{I}} & = & \textit{C}^{\mathcal{I}}_{1}
\cup \textit{C} ^{\mathcal{I}}_{2} \\%
(\forall \textit{R.}\textit{C})^{\mathcal{I}}& = & \{ \textit{o}\in
\Delta^{\mathcal{I}} \enspace | \enspace \forall \textit{o'.}( \textit{o,o'} ) \in
\textit{R}^{\mathcal{I}} \rightarrow \textit{o'} \in \textit{C}
^{\mathcal{I}} \} \\%
(\exists \textit{R.}\textit{C})^{\mathcal{I}}& = & \{ \textit{o}\in
\Delta^{\mathcal{I}} \enspace | \enspace \exists \textit{o'.}( \textit{o,o'} )
\in \textit{R}^{\mathcal{I}} \wedge \textit{o'} \in \textit{C}
^{\mathcal{I}} \} \\%
(\exists ^{\geq n} \textit{R.} \textit{C} )^{\mathcal{I}}& = &
\{ \textit{o}\in \Delta^{\mathcal{I}} \enspace | \enspace \sharp
\{ \textit{o'} \enspace | \enspace ( \textit{o,o'} ) \in \textit{R}
^{\mathcal{I}} \wedge \textit{o'} \in \textit{C} ^{\mathcal{I}} \}
\geq \textit{n}\} \\%
(\exists ^{\leq n} \textit{R.} \textit{C} )^{\mathcal{I}}& = &
\{ \textit{o}\in \Delta^{\mathcal{I}} \enspace | \enspace \sharp
\{ \textit{o'} \enspace | \enspace ( \textit{o,o'} ) \in \textit{R}
^{\mathcal{I}} \wedge \textit{o'} \in \textit{C} ^{\mathcal{I}} \}
\leq \textit{n}\} \\%
(\textit{P}^{-} )^{\mathcal{I}} & = & \{ ( \textit{o,o'} ) \in
\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \enspace | \enspace
( \textit{o',o} ) \in \textit{P}^{\mathcal{I}} \}\\%
\end{array}
$\\%
Le formalisme $\mathcal{DLX}$ est défini en rajoutant les facteurs suivants à
$\mathcal{ALCQI}$ :\\%
\\%
$
\begin{array}{rll}
( \textit{i :C} ) ^{\mathcal{I}} & = & \{ \textit{t}\in
\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \enspace | \enspace
\textit{t}[\textit{i}] \in \textit{C} ^{\mathcal{I}} \} \\%
( \leq \textit{k} [ \textit{i} ] \textit{R} ) ^{\mathcal{I}} & = &
\{ \textit{a}\in \Delta^{\mathcal{I }} \enspace | \enspace \sharp
\{ \textit{t} \in \textit{R}^{\mathcal{I}} \enspace | \enspace
\textit{t}[\textit{i}] = \textit{a} \} \leq \textit{k} \}
\end{array}
$\\%
```

Un petit exemple a pour but d'expliquer quelques propriétés ci-dessus : supposons que le concept \textit{Homme} et le rôle \textit{enfant} sont définis.

$\forall enfant.Homme $ signifie l'ensemble d'instances \textit{Homme} dont tous les enfants sont les garçons.\\%

$\exists enfant.Homme $ signifie l'ensemble d'instances qui a au moins un garçon.\\%

$\exists ^{\leq 2} enfant ^{-} $ signifie quelqu'un qui a au plus deux parents.\\%

Une \textit{base de connaissances} $\mathcal{DLX}$ est un ensemble fini d'\textit{assertions} sous la forme suivante :\\%

$
\begin{array}{rll}
\textit{C}_{1} & \sqsubseteq & \textit{C}_{2}\\%
\textit{R}_{1} & \sqsubseteq & \textit{R}_{2}
\end{array}
$
\\%

Il existe les services d'inférences principaux sur la base $\mathcal{DLX}$: \textit{validabilité de la base de connaissances} s'il existe un modèle (interprétation) pour la base de connaissances $\mathcal{K}$, \textit{validabilité de concept} si $\mathcal{K}$ a un modèle $\mathcal{I}$ tel que $\textit{C}^{\mathcal{I}} \neq \varnothing $, et \textit{subsomption} si $\mathcal{K}$ implique une assertion d'inclusion $\textit{C}_{1} \enspace \sqsubseteq \enspace \textit{C}_{2}$ et pour toutes les interprétations $\mathcal{I} : \textit{C}_{1}^{\mathcal{I}} \sqsubseteq \textit{C}_{2}^{\mathcal{I}}$.\\%

\textbf{Exemple de la base $\mathcal{K}$ pour \textit{Core Component}}\\%

Dans CC, \textit{produit} est défini comme une \textit{Aggregate Information Entity} constituée de \textit{Basic Information Entity} : \textit{quantité} et \textit{prix} comme ses attributs. Supposons que les concepts \textit{Produit, Description, Quantité, Prix} existent dans la base $\mathcal{K}$ partagée. Les relations (rôles) \textit{attribut} entre \textit{Produit} et \textit{Quantité} : $\textit{R}_{1}$, \textit{Produit} et \textit{Prix} : $\textit{R}_{2}$ sont définies en utilisant la notation présentée ci-dessus :\\%

\\%

$
\begin{array}{rll}
\textit{R}_{1}& \sqsubseteq & (1 :\textit{Produit}) \sqcap (2 :\textit{Quantité})\\%
\textit{Produit} & \sqsubseteq & ((\leq 1 [1] \textit{R}_{1}) \sqcap (\leq 1 [1] \textit{R}_{1}))\\%
\end{array}
$

La dernière assure que pour chaque instance \textit{Produit} il existe uniquement une instance \textit{Quantité} par le biais $\textit{R}_{1}$.\\%

\\%

$
\begin{array}{rll}
\textit{R}_{2}& \sqsubseteq & (1 :\textit{Produit}) \sqcap (2 :\textit{Prix})\\%
\textit{Produit} & \sqsubseteq & ((\leq 1 [1] \textit{R}_{2}) \sqcap (\leq 1 [1] \textit{R}_{2}))\\%
\end{array}
$

$
La dernière assure que pour chaque instance \textit{Produit} il existe uniquement une instance \textit{Prix} par le biais $\textit{R}_{2}$.\\%
\\%
Maintenant, l'acteur A souhaite composer un document \textit{commande} pour envoyer à l'acteur B. Ce document-là contient un \textit{ProduitA} qui est \textit{localement} défini sur A en rajoutant l'attribut \textit{Description} pour décrire, par exemple la fragilité du produit :\\%
\\%
$
\begin{array}{rll}
\textit{ProduitA}& \sqsubseteq & \textit{Produit} \\%
\textit{R}_{3}& \sqsubseteq & (1 :\textit{ProduitA}) \sqcap (2 :\textit{Description})\\%
\textit{ProduitA} & \sqsubseteq & ((\leq 1 [1] \textit{R}_{3}) \sqcap (\leq 1 [1] \textit{R}_{3}))
\end{array}
$
\\%
Pour des raisons de simplicité, supposons qu'il n'existe pas d'ontologie locale sur l'acteur B. Même si le concept \textit{ProduitA} n'est pas défini dans la base partagée, l'acteur B peut interpréter ce concept en s'appuyant sur les connaissances existantes. Effectivement, le service d'inférence \textit{subsomption} de la base $\mathcal{K}$ permettra de découvrir que le concept \textit{ProduitA} est défini du \textit{Produit}, donc il interprétera \textit{ProduitA} comme un \textit{Produit}. En outre, puisque le concept \textit{Description} existe dans la base partagée, l'acteur B peut le traiter si nécessaire. L'idée de l'algorithme du calcul \textit{subsomption} est extraite de \cite{5} comme suit :
\begin{itemize}
\item[$\bullet$]
transformer les concepts en formes normales
\item[$\bullet$]
propager les contraintes et reconnaître les irrationalités s'il existe
\item[$\bullet$]
comparer structuralement les expression obtenues
\end{itemize}
\subsection{$\mathcal{DLX}$ et diagrammes UML}
Une classe de l'UML constituée de trois composants : identification (nom de classe), les attributs et les signatures des méthodes est naturellement représentée par un concept de $\mathcal{DLX}$. Dans ce rapport, nous ne traitons pas la transformation de ces signatures en $\mathcal{DLX}$.

Nous définissons les assertions de même façon pour les \textit{attributs} et les relations \textit{agrégation} entre des classes dans un diagramme de classes :\\%
$
\begin{array}{rll}
\textit{R}& \sqsubseteq & (1 :\textit{C}_{1}) \sqcap (2 :\textit{C}_{2}) \\%

\textit{C}_{1} & \sqsubseteq & ((\leq n_{u} [1] \textit{R}) \sqcap (\leq n_{v} [1] \textit{R}))\\%
\end{array}
$
\\%
où $\textit{C}_{1}$ contient $\textit{C}_{2}$; n_{u}, n_{v} sont la multiplicité de l'agrégation \textit{R}.\\%

La transformation d'une association entre deux classes $\textit{C}_{1}, \textit{C}_{2}$ vers $\mathcal{DLX}$ est plus sophistiquée car il nous faut capturer ses significations suivantes :

\begin{itemize}
\item[] i) définir un concept \textit{A} pour l'association \textit{A} et deux rôles $\textit{r}_{1}, \textit{r}_{2}$ pour les composants de \textit{A}. Chaque rôle $\textit{r}_{i}$ a \textit{A} comme le premier composant et $\textit{C}_{i}$ comme le deuxième.

\item[] ii) assurer que le concept \textit{A} possède exactement un composant de chaque $\textit{r}_{i}$.

\item[] iii) assurer que chaque instance du concept \textit{A} détermine uniquement l'association correspondante.
\end{itemize}

\begin{figure}[h]
\begin {center} \includegraphics{d :/travaux/diag1.eps}
\end{center}
\caption{\textit{Association dans UML}}
\end{figure}

Il faudra probablement rajouter de nouveaux facteurs à $\mathcal{DLX}$ pour capturer toutes ces significations.\\%

Pour ce qui concerne la généralisation, on peut le transformer en l'assertion suivante :\\%

\hugetab $\textit{C}_{i} \sqsubseteq \textit{C}$ \\%

Cependant, comment capturer la sémantique du mécanisme de l'héritage et du polymorphisme dans la relation de généralisation ? Nous ne traitons pas ce problème dans ce rapport !\\%

Quant aux diagramme d'états UML, les spécifications ebXML exigent les manipulations suivantes sur les processus commerciaux : vérifier si un processus est équivalent à l'autre ou vérifier si l'un est inclus dans l'autre. La question sur la représentation sémantique des processus nous fait penser qu'il faudra améliorer l'expressivité de $\mathcal{DLX}$. En effet, en sachant que le diagramme d'états décrit des comportements d'une instance d'un concept, les états et les transitions du diagramme peuvent être définis respectivement comme un concept \textit{état} et les \textit{fonctions} d'un ensemble de concepts dans un autre concept. Ces fonctions permettent de déterminer la relation entre deux états consécutifs. Dans cette hypothèse, un diagramme d'états d'un concept \textit{C} peut se traduire en un \textit{transducteur} dans la théorie des automates.\\%

Il est évident qu'une extension de $\mathcal{DLX}$ en y rajoutant des définitions sur la \textit{relation n-arité} et sur la \textit{fonction} sera utile. Dans ce

cas-là, nous espérons que des algorithmes développés sur les transducteurs permettent d'arriver aux spécifications mentionnées.

\subsection{$\mathcal{DLX}$ et règles de production}

Les règles de production est un formalisme important pour la représentation de la sémantique ebXML car les informations contextuelles se traduisent sous la forme de ces règles. En plus, l'assemblage des documents commerciaux subit également une application de règles sur les composants de base.

Une règle de production peut être représentée comme suit :\\%
$
\begin{array}{rll}
regle & : := & <condition> \enspace \Rightarrow \enspace <conclusion> \\%
<condition> & : := & <predicat> (<variable>,<constant>) \\%
<conclusion> & : := & <action> (<variable>,<constant>) \\%
<predicat> & : := & \textit{same } |\textit{ notsame }| ... \\%
<action> & : := & \textit{add } |\textit{ remove }| ... \\%
\end{array}
$\\%

On rend compte facilement que la sémantique de cette représentation est capturée en $\mathcal{DLX}$ sans difficulté sauf \textit{action}. En effet, on peut considérer \textit {<condition>, <conclusion>, <variable>} comme les \textit{concepts}, et \textit{predicat, same, notsame} comme les \textit{rôles}. En outre, si l'on considère l'action \textit{add} comme l'ajout d'un attribut \textit{C'} à un concept C, cela est spécifié par l'assertion suivante :\\%
$
\begin{array}{rll}
\textit{R}& \sqsubseteq & (1 :\textit{C}) \sqcap (2 :\textit{C'}) \\%
\end{array}
$\\%

Cependant, on ne peut pas transformer de la même façon l'action \textit{remove} en $\mathcal{DLX}$. Il existe des travaux, par exemple \cite{19}, qui traite de la transformation des règles Horn vers DL mais celles-ci ne contiennent pas le concept \textit{action}. La formalisation de ce concept en DL exige donc plus de travail.

\section{Conclusion}

Ce rapport examine la capture et la représentation sémantique commerciale de l'ebXML. A travers cette étude, les formalismes avec ses problèmes sont identifiés . Ces problèmes se posent après avoir analysé les spécifications ebXML qui exigent des manipulations sur cette sémantique. Nous essayons de proposer un formalisme, nommé $\mathcal{DLX}$ pour la formaliser. Ces premiers efforts ouvrent une nouvelle direction de recherche pour des problèmes traditionels. Cependant, il nous reste beaucoup de travail à faire pour obtenir les résultats attendus. Pour cette raison, notre travail suivant va consacrer aux tâches principales comme suit : compléter la transformation des diagrammes de classes UML, représenter des règles de production en $\mathcal{DLX}$. Une extension de $\mathcal{DLX}$ sera également étudiée pour les diagrammes d'états UML.

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. Les deux sections suivantes du chapitre se concentrent sur les nouveaux services d'inférences développés et l'architecture du système ONDIL. Dans la Section 6.5, nous présentons l'application de ONDIL à la conception et à la maintenance des ontologies correspondant aux taxonomies de bcXML. Une discussion sur le système ONDIL et une présentation de perspectives terminent le chapitre.

6.1. Systèmes d'aide à la conception et à la gestion des ontologies(3-4)

6.1.1. FaCT[1]. Le système FaCT est développé à *University of Manchester* par *Ian Horrocks* [**Hor, 1997**]. 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 [**Hor, 1997**] mais les plus importantes sont :

– Normalisation et codage. Cette technique se base sur la syntaxe de descriptions de concept pour détecter toutes les insatisfiabilités évidentes et les structures

[1]Fast Classification of Terminology

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 constraintes 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 programation 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 inférentes 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 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 le requête HTTP POST. Le corps d'une requête doit être un message encodé en XML. Le serveur DIG

[2]Description Logic Implementation Group

utilise l'élément de racine d'un message pour déterminer le type du message. Les requêtes Ask et Tell sont envoyées par client et le requête Reponse est retournée par serveur.

6.1.2. RACER[3]. Le système FaCT est développé à *University of Hamburg* (2001). Comme FaCT, le moteur d'inférence de RACER utilise les algorithmes de tableaux optimisés. RACER autorise au langage L.D très expressif $\mathcal{SHIQ}$. En dehors des services offerts par FaCT, RACER supporte le ABox , requêtes sur ABox et peut gérer plusieurs TBox en même temps. De plus, RACER autorise à modifier un TBox soumis.

6.2. Représentation de connaissances dans ONDIL

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, resctriction existentielle. La base terminomogique $\Delta_{\mathcal{T}}$ 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)

EXAMPLE 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**.

[3]Renamed ABox and Concept Expression Reasoner

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.

EXAMPLE 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 compréhention 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 acylique (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).

EXAMPLE 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 $\Delta_{\mathcal{R}}$ continet 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 ?), 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 Petit Subsumeur Commun (lcs). L'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 une définition du concept "la plus petite" qui satisfait toutes les relations à l'aide d'un service d'inférence permetant 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.

L'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 [**?**] étend l' algorithme de calcul *lcs* au langage $\mathcal{ALEN}$. Notons que le *lcs* de descriptions de concept dans le 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. Echange 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 [**BKM, 2000**] 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 [**BKM, 2000**].

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 implémanté 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 $O_{\mathcal{ALC}}$ utilisentles 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 réçu 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 réçu. 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 traité dans ONDIL.

6.3.4. Echange 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 projectés dans les ontologies spécifiques selon ces informations contextuelles. Ainsi, les vocabulaires appropriés à ce contexte sont créés. L'approche proposée dans ONDIL inspirée de ebXML utilise les règles de contexte comme un formalisme supplémentaire pour représenter les informations contextuelles elles-mêmes et pour appliquer ces informations à l'ontologie de base. 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 context 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 necessite 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 sont attachés à chaque ontologie dérivée. Ils servent à réviser l'ontologie dans la phase de maintenance de l'ontologie. Plus précisément, les deux services de révision suivants sont intégrés dans l'ontologie parmi les autres :

$$\text{OUBLIER}(C, Exp) \text{ et } \text{DIRE}(C, Exp)$$

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 égalment un service d'inférence spécifique pour le formalisme hybride basé sur l'Algorithme 4.5.9 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. Plus précisément, //

6.4. Architecture Technique de ONDIL(2-3)

Base de connaissances (TBox + ABox + Règles) //
Moteur d'inférence FaCT //
Couche de gestion et de maintenance //
Corba, HTTP (Servlet, Applet ...) //

6.5. Application de ONDIL

6.5.1. Aide à la conception des Ontologies avec ONDIL (2). Construction //

Ces contextes sont catégorisés dans la Section ?

6.5.2. Maintenance des Ontologies avec ONDIL (3). Conversion XMI-TBox //
Visualisation //
Interface basée sur Web //

6.5.3. Logiciels développés et scénario d'application : le cas bcXML (IFC)(2). Secteur de la construction //
IFC, bcXML=> Ontologie de base => Ontologies dérivées //
Scénario d'échange //

6.6. Conclusion et Perspective (1)

Base de données pour ABox //
Langages L.D expressifs avec le constructeur de cardinalité et de disjonction //

Bibliographie

[Baa, 1996] Baader, F. (1996). Logic-based Knowledge Representation. *Journal of KI*-96.

[Bor, 1994] Borgida, A. (1994). On the relationship between description logic and predicate logic. *Proceedings of CIKM-94*.

[CLN, 1999] Calvanese, D., Lenzerini, M. et Nardi, D. (1999). Unifying class-based representation formalismes. *Journal of Artificial Intelligence Research*.

[ebX, 2001a] ebXML (2001). ebXML Technical Architecture Specification. Voir *http ://www.ebxml.org*.

[ebX, 2001b] ebXML (2001). ebXML Business Process Specification Schema. Voir *http ://www.ebxml.org*.

[ebX, 2001c] ebXML (2001). ebXML Business Process and Business Information Analysis. Voir *http ://www.ebxml.org*.

[ebX, 2001d] ebXML (2001). ebXML Business Process Analysis Worksheets & Guideline. Voir *http ://www.ebxml.org*.

[ebX, 2001e] ebXML (2001). ebXML Core Component Overview. Voir *http ://www.ebxml.org*.

[ebX, 2001f] ebXML (2001). ebXML Core Component Discovery and Analysis. Voir *http ://www.ebxml.org*.

[ebX, 2001g] ebXML (2001). ebXML Core Components, Document Assembly and Context Rules. Voir *http ://www.ebxml.org*.

[ebX, 2001h] ebXML (2001). ebXML Core Components, Context and Reutisability of Core Components. Voir *http ://www.ebxml.org*.

[Cla, 2000] Clark, C.J. (2000). The UN/CEFACT Unified Modeling Methodology - An Overview. *UNCEFACT/TMWG/N097*.

[UC, 2001] UN/CEFACT (2001). Introduction, Business Modeling Business Requirements and Design of UMM. *UNCEFACT/TMWG/N097*.

[Ome, 2002] Omelayenko, B. (2002). Integrating Vocabularies : Discovering and Representing Vocabulary Maps. *Proceeding of the First International Semantic Web Conference*.

[WH, 2002] Wache, H. et Vögele, T. (2002). Ontology-Based Integration of Information A Survey of Existing Approches. *Proceeding of the First International Semantic Web Conference*.

[Kim, 1998] Kimbrough, S.O. (1998). Formal Language for Business Communication : Sketch of basic theory. *International Journal of Electronic Commerce*.

[Kim, 2000] Kimbrough, S.O. (2000). EDI, XML, and the Transparency Problem in Electronic Commerce. *INFORMS, San Antonio*.

[KM, 1997] 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*.

[Moo, 1997] Moore, S. O. (1997). A foundation for flexible automated electronic comunication. *Information System Research*.

[Fen, 2001] Fensel, D. (2001). Ontologies : A Silver Bullet for Knowledge Management and Electronic Commerce. *Springer*.

[Gru, 1993] Grubert, T. (1993). A Translation approach to portable ontologies. *Knowledge 7.*

[Gru, 1995] Grubert, T.(1995). Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum. Comput. Stud. 43.*

[BHL, 2001] Berners-Lee, T., Hendler, J. et Lassila, O. (2001). The semantic Web. *Scientific American.*

[LB, 1985] Levesque, H. J. et Brachman, R. J. (1985). A Fundamental Tradeoff in Knowledge Representation and Reasoning. *Readings in Knowledge Representation. Morgan Kaufmann.*

[AGM, 1985] 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.

[Gär, 1992] Gärdenfors, P. (1992). Belief Revision. *Cambridge University Press.*

[Ant, 2000] Anthony, J.R. (2000). Belief Revision-An overview.

[Neb, 1990a] Nebel, B. (1995). Reasoning and Revision in Hybrid Representation Systems. *Lecture Notes in Computer Science*, *Springer-Verlag.*

[Neb, 1990b] Nebel, B. (1990). Terminological reasoning is inherently intractable. *Artificial Intelligence*, *43.*

[PA, 1991] Pollock, J. L. et Anthony, S.G. (1991). Belief Revision and Epistemology.

[SS, 1991] Schmidt-Schauß, M. et Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence*, *48.*

[LR, 1998] Levy, A.Y. et Rousset, M-C. (1998). Combining Horn Rules and Description Logics in CARIN, *Artificial Intelligence, 104.*

[LR, 1996] 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.*

[DLN$^+$, 1998] Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W. et Schaerf, A. (1998). An Epistemic Operator for Description Logics, *Artificial Intelligence, 100.*

[DLN$^+$, 1996] 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.*

[DHL$^+$, 1992] 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.*

[DLN$^+$, 1998b] Donini, F.M., Lenzerini, M., Nardi, D. et Schaerf, A. (1998) AL-log : Integrating Datalog and Description Logics. *Journal of Intelligence Information Systems, 10.*

[Küs, 2001] Küsters, R. (2001). Non-Standard Inferences in Description Logics. *PhD. Thesis, Springer.*

[BKM, 1999] 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*).

[KM, 2001] Küsters, R. et Molitor, R. (2001). Computing least common subsumer for $\mathcal{ALEN}$. *Proceeding of IJCAI01, pages 219-224, Morgan Kaufman.*

[BKM, 2000] Baader, F., Küsters, R. et Molitor, R. (2000). Rewriting Concepts Using Terminologies. *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning* (KR2000).

[BKT, 2002a] Brandt, S., Küsters, R. et Turhan, A. Y. (2002). Approximation and Difference in Description Logics. *Proceeding of KR2002.*

[BKT, 2002b] Brandt, S., Küsters, R. et Turhan, A.Y. (2002). Approximation ALCN-Description Logics. *Proceeding of DL'02.*

[BT, 2002] Baader, F. et Turhan, A.Y. (2002). On the problem of Computing Small Representations of Least Common Subsumers. *Proceeding of KI2002.*

[CCD+, 2002] 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).*

[CDL, 2001] 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).*

[BCD, 2003] Berardi, D., Calvanese, D. et De Giacomo, G. (2003). Reasoning on UML Class Diagrams is EXPTIME-hard. *Dans Proc. of Int. Workshop on Description Logics (DL 2003).*

[SHG+, 2001] 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.*

[GOR, 1997] 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).*

[Mor, 1975] Mortimer, M. (1975). On languages with two variables. *Zeitschr. f. math. Logik und Grundlagen d. Math., 21.*

[BCM+, 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (2003). The Description Logic Handbook-Theory, Implementation and Applications, *Cambridge University Press.*

[TGL+, 1988] 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.*

[HU, 1979] Hopcroft, J.E., Ullman, J.D. (1979). Introduction to Automata Theory, Languages and Computation. *Addison Wesley.*

[Wil, 1986] Wilf, H.S. (1986). Algorithmes et Complexité. *Prentice Hall.*

[Ber, 1966] Berger, R. (1966). The undecidability of the dominoe problem. *Mem. Amer. Math. Soc., 66* (1-72).

[Hor, 1997] Horrocks, I. (1997). Optimising Tableaux Decision Procedures for Description Logics. *PhD thesis, University of Manchester.*

[Min, 1975] Minsky, M. (1975). A framework for representing knowledge. The Psychology of Computer Vision. *McGraw-Hill.*

[Hay, 1979] Hayes, P.J. (1979). The logic of frames. *Frame Conception and Text Understanding. Walter de Gruyter and Co..*

[Qui, 1968] Quillian, M. (1968). Semantic memory. Semantic Information Processing. *MIT Press, Cam.*

[Bra, 1977] Brachman, R. J. (1977). What's a concept : Structural foundations for semantic networks. *International Journal of Man-Machine Studies.*

[Bra, 1978] Brachman, R.J. (1978). Structured inheritance networks. *Research in Natural Language Understanding.*

[Sow, 1984] Sowa, J. F. (1984). Conceptural structures : Information Processing in Mind and Machine. *Addison Wesley.*

[BMT, 1999b] 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).

[LL, 2002] Le Duc, C., Le Thanh, N. (2002). Problématique de l'ebXML et logiques de description. *Rapport de recherche à l'I3S.*

[LL, 2003] 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.*

Conclusion

Bibliographie

[Baa, 1996] Baader, F. (1996). Logic-based Knowledge Representation. *Journal of KI*-96.

[Bor, 1994] Borgida, A. (1994). On the relationship between description logic and predicate logic. *Proceedings of CIKM-94*.

[CLN, 1999] Calvanese, D., Lenzerini, M. et Nardi, D. (1999). Unifying class-based representation formalismes. *Journal of Artificial Intelligence Research*.

[ebX, 2001a] ebXML (2001). ebXML Technical Architecture Specification. Voir *http ://www.ebxml.org*.

[ebX, 2001b] ebXML (2001). ebXML Business Process Specification Schema. Voir *http ://www.ebxml.org*.

[ebX, 2001c] ebXML (2001). ebXML Business Process and Business Information Analysis. Voir *http ://www.ebxml.org*.

[ebX, 2001d] ebXML (2001). ebXML Business Process Analysis Worksheets & Guideline. Voir *http ://www.ebxml.org*.

[ebX, 2001e] ebXML (2001). ebXML Core Component Overview. Voir *http ://www.ebxml.org*.

[ebX, 2001f] ebXML (2001). ebXML Core Component Discovery and Analysis. Voir *http ://www.ebxml.org*.

[ebX, 2001g] ebXML (2001). ebXML Core Components, Document Assembly and Context Rules. Voir *http ://www.ebxml.org*.

[ebX, 2001h] ebXML (2001). ebXML Core Components, Context and Reutisability of Core Components. Voir *http ://www.ebxml.org*.

[Cla, 2000] Clark, C.J. (2000). The UN/CEFACT Unified Modeling Methodology - An Overview. *UNCEFACT/TMWG/N097*.

[UC, 2001] UN/CEFACT (2001). Introduction, Business Modeling Business Requirements and Design of UMM. *UNCEFACT/TMWG/N097*.

[Ome, 2002] Omelayenko, B. (2002). Integrating Vocabularies : Discovering and Representing Vocabulary Maps. *Proceeding of the First International Semantic Web Conference*.

[WH, 2002] Wache, H. et Vögele, T. (2002). Ontology-Based Integration of Information A Survey of Existing Approches. *Proceeding of the First International Semantic Web Conference*.

[Kim, 1998] Kimbrough, S.O. (1998). Formal Language for Business Communication : Sketch of basic theory. *International Journal of Electronic Commerce*.

[Kim, 2000] Kimbrough, S.O. (2000). EDI, XML, and the Transparency Problem in Electronic Commerce. *INFORMS, San Antonio*.

[KM, 1997] 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*.

[Moo, 1997] Moore, S. O. (1997). A foundation for flexible automated electronic comunication. *Information System Research*.

[Fen, 2001] Fensel, D. (2001). Ontologies : A Silver Bullet for Knowledge Management and Electronic Commerce. *Springer*.

[Gru, 1993] Grubert, T. (1993). A Translation approach to portable ontologies. *Knowledge 7.*

[Gru, 1995] Grubert, T.(1995). Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum. Comput. Stud. 43.*

[BHL, 2001] Berners-Lee, T., Hendler, J. et Lassila, O. (2001). The semantic Web. *Scientific American.*

[LB, 1985] Levesque, H. J. et Brachman, R. J. (1985). A Fundamental Tradeoff in Knowledge Representation and Reasoning. *Readings in Knowledge Representation. Morgan Kaufmann.*

[AGM, 1985] 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.

[Gär, 1992] Gärdenfors, P. (1992). Belief Revision. *Cambridge University Press.*

[Ant, 2000] Anthony, J.R. (2000). Belief Revision-An overview.

[Neb, 1990a] Nebel, B. (1995). Reasoning and Revision in Hybrid Representation Systems. *Lecture Notes in Computer Science, Springer-Verlag.*

[Neb, 1990b] Nebel, B. (1990). Terminological reasoning is inherently intractable. *Artificial Intelligence, 43.*

[PA, 1991] Pollock, J. L. et Anthony, S.G. (1991). Belief Revision and Epistemology.

[SS, 1991] Schmidt-Schauß, M. et Smolka, G. (1991). Attributive concept descriptions with complements. *Artificial Intelligence, 48.*

[LR, 1998] Levy, A.Y. et Rousset, M-C. (1998). Combining Horn Rules and Description Logics in CARIN, *Artificial Intelligence, 104.*

[LR, 1996] 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.*

[DLN$^+$, 1998] Donini, F.M., Lenzerini, M., Nardi, D., Nutt, W. et Schaerf, A. (1998). An Epistemic Operator for Description Logics, *Artificial Intelligence, 100.*

[DLN$^+$, 1996] 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.*

[DHL$^+$, 1992] 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.*

[DLN$^+$, 1998b] Donini, F.M., Lenzerini, M., Nardi, D. et Schaerf, A. (1998) AL-log : Integrating Datalog and Description Logics. *Journal of Intelligence Information Systems, 10.*

[Küs, 2001] Küsters, R. (2001). Non-Standard Inferences in Description Logics. *PhD. Thesis, Springer.*

[BKM, 1999] 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*).

[KM, 2001] Küsters, R. et Molitor, R. (2001). Computing least common subsumer for $\mathcal{ALEN}$. *Proceeding of IJCAI01, pages 219-224, Morgan Kaufman.*

[BKM, 2000] Baader, F., Küsters, R. et Molitor, R. (2000). Rewriting Concepts Using Terminologies. *Proceedings of the Seventh International Conference on Knowledge Representation and Reasoning* (KR2000).

[BKT, 2002a] Brandt, S., Küsters, R. et Turhan, A. Y. (2002). Approximation and Difference in Description Logics. *Proceeding of KR2002.*

[BKT, 2002b] Brandt, S., Küsters, R. et Turhan, A.Y. (2002). Approximation ALCN-Description Logics. *Proceeding of DL'02.*

[BT, 2002] Baader, F. et Turhan, A.Y. (2002). On the problem of Computing Small Representations of Least Common Subsumers. *Proceeding of KI2002.*

[CCD+, 2002] 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).*

[CDL, 2001] 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).*

[BCD, 2003] Berardi, D., Calvanese, D. et De Giacomo, G. (2003). Reasoning on UML Class Diagrams is EXPTIME-hard. *Dans Proc. of Int. Workshop on Description Logics (DL 2003).*

[SHG+, 2001] 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.*

[GOR, 1997] 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).*

[Mor, 1975] Mortimer, M. (1975). On languages with two variables. *Zeitschr. f. math. Logik und Grundlagen d. Math., 21.*

[BCM+, 2003] Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P. (2003). The Description Logic Handbook-Theory, Implementation and Applications, *Cambridge University Press.*

[TGL+, 1988] 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.*

[HU, 1979] Hopcroft, J.E., Ullman, J.D. (1979). Introduction to Automata Theory, Languages and Computation. *Addison Wesley.*

[Wil, 1986] Wilf, H.S. (1986). Algorithmes et Complexité. *Prentice Hall.*

[Ber, 1966] Berger, R. (1966). The undecidability of the dominoe problem. *Mem. Amer. Math. Soc., 66* (1-72).

[Hor, 1997] Horrocks, I. (1997). Optimising Tableaux Decision Procedures for Description Logics. *PhD thesis, University of Manchester.*

[Min, 1975] Minsky, M. (1975). A framework for representing knowledge. The Psychology of Computer Vision. *McGraw-Hill.*

[Hay, 1979] Hayes, P.J. (1979). The logic of frames. *Frame Conception and Text Understanding. Walter de Gruyter and Co..*

[Qui, 1968] Quillian, M. (1968). Semantic memory. Semantic Information Processing. *MIT Press, Cam.*

[Bra, 1977] Brachman, R. J. (1977). What's a concept : Structural foundations for semantic networks. *International Journal of Man-Machine Studies.*

[Bra, 1978] Brachman, R.J. (1978). Structured inheritance networks. *Research in Natural Language Understanding.*

[Sow, 1984] Sowa, J. F. (1984). Conceptural structures : Information Processing in Mind and Machine. *Addison Wesley.*

[BMT, 1999b] 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).

[LL, 2002] Le Duc, C., Le Thanh, N. (2002). Problématique de l'ebXML et logiques de description. *Rapport de recherche à l'I3S.*

[LL, 2003] 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.*