Réécriture de requêtes en termes de vues en présence de contraintes de valeurs pour un système d'intégration de sources de données agricoles

Hélène Jaudoin

N° D'ORDRE : D.U. 1618

EDSPIC : 338

Université Blaise Pascal Clermont-Ferrand II

ECOLE DOCTORALE
SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

THÈSE

présentée par

Hélène JAUDOIN

DEA :
Informatique, productique, imagerie médicale

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité : INFORMATIQUE

Réécriture de requêtes en termes de vues en présence de contraintes de valeurs pour un système d'intégration de sources de données agricoles

Soutenue publiquement le 29 novembre 2005 devant le jury :

Président :	Pr. Christine Collet
Rapporteurs :	Pr. Dominique Laurent
	Pr. Mohand-Said Hacid
Examinateurs :	Pr. Michel Schneider (Directeur de thèse)
	Dr. Farouk Toumani (Co-directeur de thèse)
	M. Frédéric Vigier (Co-directeur de thèse)
	Pr. Jean-Marc Petit

Mis en page avec la classe thloria.

Remerciements

Je voudrais tout d'abord remercier Madame Christine Collet, Professeur à l'ENSIMAG de Grenoble, pour avoir accepté de présider mon jury de thèse ainsi que Monsieur Dominique Laurent, Professeur à l'Université de Cergy-Pontoise, et Monsieur Mohand-Saïd Hacid, Professeur à l'University Claude Bernard Lyon 1, pour avoir accepté de rapporter ma thèse.

Cette thèse est issue d'un projet collaboratif entre le LIMOS et le Cemagref. Aussi je voudrais remercier mon directeur de thèse Monsieur Michel Schneider, Professeur à l'Université Blaise Pascal et Monsieur Frédéric Vigier, Ingénieur de Recherche du Cemagref qui ont initié ce projet, me donnant ainsi l'opportunité d'effectuer une thèse. Au delà de ces aspects administratifs, je voudrais à nouveau remercier Michel Schneider pour son écoute, ses conseils, ses critiques et sa disponibilité tout au long de la thèse. Je remercie également Frédéric Vigier et toute l'équipe Copain qui m'ont permis de mieux comprendre les enjeux du domaine applicatif.

Je tiens également à exprimer toute ma reconnaissance et présenter mes remerciements chaleureux à Farouk Toumani, Maître de Conférence à l'ISIMA de Clermont-Ferrand, pour ses qualités tant humaines que scientifiques et pour tout le temps passé à expliquer, conseiller, critiquer et encourager. Sa rigueur scientifique et sa passion de la recherche ont été moteurs tout au long de la thèse et salvateurs dans les moments de découragement.

Je voudrais remercier Jean-Marc Petit, Professeur à l'INSA de Lyon, pour ses nombreux conseils durant la thèse ainsi que pour sa lecture et sa critique du mémoire et enfin pour son aide dans le domaine de la fouille de données.

Je souhaite également remercier le Cemagref, l'ISIMA et le LIMOS de m'avoir accueillie au sein de leurs locaux. Je remercie également le secrétariat de ces établissements pour leur aide administrative et tout le personnel pour leur sympathie. De plus je remercie le Cemagref et l'Université Blaise Pascal pour le financement de la thèse indispensable à son bon déroulement.

Enfin, je voudrais remercier mes amis et mes proches d'être et d'avoir été là : merci à François, Chak, Marie, Nadine, Laure, Fred (et merci également pour son aide dans l'implantation), Juan-Pedro, Myriam, merci aux compagnons d'infortune : les gardiens de l'ISIMA de cet été et plus généralement aux doctorants et aux jeunes docteurs du Cemagref et de l'ISIMA, merci à mon père pour m'avoir donné le goût des sciences, à ma famille et enfin à Romain pour ses encouragements.

Liste des tableaux

Table des figures

Table des matières

Introduction

Cette thèse s'est déroulée dans le cadre d'un projet régional réalisé en partenariat avec le Cemagref [1], un institut public de recherche pour l'ingénierie de l'agriculture et de l'environnement. Le travail réalisé durant cette thèse est motivé par une application dans le domaine du développement durable, dont les enjeux thématiques puis scientifiques sont détaillés ci-après.

Enjeux thématiques

La prise de conscience récente de l'importance de consommer des produits de qualité et de préserver notre environnement a amené les citoyens à s'interroger sur les pratiques agricoles. Suite aux crises de la vache folle et du sang contaminé, à l'utilisation massive de pesticides qui polluent les rivières et qui se retrouvent dans la chaîne alimentaire (comme dans le miel par exemple), les consommateurs sont demandeurs d'une transparence vis à vis des pratiques agricoles qui doivent garantir d'une part, la qualité et l'origine des produits et qui doivent relater d'autre part, les actions mises en oeuvre pour tendre vers des pratiques plus respectueuses de l'environnement.

Ces considérations ont entraîné l'apparition du concept de *traçabilité* des pratiques à laquelle les acteurs du monde agricole doivent contribuer. Ces acteurs sont également demandeurs d'une valorisation de leurs pratiques. De plus, les ministères et leurs services déconcentrés ont la tâche de vérifier si les pratiques agricoles sont conformes aux nouvelles réglementations comme par exemple la réglementation sur les épandages de boues qui fixe les prescriptions techniques applicables aux épandages de boues sur les sols agricoles (Arrêté du 8 janvier 1998 pris en application du décret n^o 97-1133 du 8 décembre 1997 modifié par l'arrêté du 3 juin 1998), ou sur l'irrigation (Article n^o 92-3 du 3 janvier 1992) qui oblige à mesurer les volumes d'eau prélevés). Les acteurs agricoles peuvent également prendre des engagements sur leurs pratiques sous forme de contrats qui par exemple, peuvent lier une coopérative et un producteur céréalier. Le producteur s'engage à ne pas dépasser des quantités de produits phytosanitaires déversés sur ses parcelles, en échange de l'achat d'une certaine quantité de céréales par la coopérative. Certains exploitants agricoles ont également la volonté d'apporter une qualification à leur exploitation. Par exemple, pour obtenir la certification "agriculture raisonnée", l'exploitation doit vérifier un certain nombre de critères comme notamment, la diminution des intrants phytosanitaires et la régulation de l'irrigation des parcelles.

Ainsi les acteurs du monde agricole se trouvent confrontés à diverses contraintes pesant sur la gestion de leur exploitation. La prise en compte de ces contraintes implique la mise en oeuvre de systèmes informatiques pour l'enregistrement, le suivi et la vérification des pratiques agricoles et des produits utilisés en élevage et culture [61]. Le développement de tels systèmes est devenu un enjeu économique en agriculture [60]. Aujourd'hui, on peut trouver foison de logiciels en relation avec le type de pratiques et le type de culture et d'élevage [61], comme par exemple Planfum [2] pour la prévision et l'enregistrement des intrants de type fumure, Agri-Trace [3] pour tracer les productions végétales. Il existe également de nombreux systèmes d'in-

[1] http ://www.cemagref.fr/

[2] http ://www.allier.chambagri.fr/logiciels/planfum.asp

[3] http ://www.prov-liege.be/servagricoles/agri-trace/

formation comme SIREME [62] pour les coopératives agricoles ayant signé un contrat ”agriconfiance” avec les exploitations, ou SIGEMO [76] qui vise à suivre les pratiques d'épandages de matières organiques.

Cependant, les enregistrements des pratiques agricoles sont pour la plupart effectués en local (sur le site de l'exploitation ou chez un partenaire agricole) avec des logiciels propriétaires. Ainsi le domaine du développement durable est formé d'un ensemble de systèmes informatiques totalement autonomes et distribués. Il est alors difficile pour les ministères et leurs services déconcentrés de contrôler efficacement l'ensemble des pratiques agricoles parce qu'ils ont besoin de regrouper et/ou coupler des informations provenant de logiciels dédiés à des pratiques différentes. Considérons par exemple, le cas des pratiques des épandages de boues et des traitements en pesticides. Les épandages de boues dans la région du Puy-de-dôme ont pu être enregistrés par des bureaux d'étude ou par des exploitants. De la même façon, l'enregistrement des traitements en pesticide a été l'objet de différents partenaires agricoles de la région. Si un service souhaite collecter des informations sur les parcelles du département de l'Allier qui ont reçu des épandages de boues et des pesticides, les personnes responsables de cette enquête doivent consulter chacun des acteurs responsables des enregistrements, et doivent finalement, saisir et coupler à la main toutes ces données.

Pour faciliter la traçabilité des pratiques, le Cemagref a été chargé de mettre en place des techniques d'analyse et d'intégration de données permettant la coopération des sources de données distribuées et hétérogènes. Ces techniques doivent être flexibles afin de tenir compte de la versatilité de l'environnement, i.e. de l'arrivée de nouvelles sources de données. Grâce au processus d'informatisation progressive du secteur agricole, de nouvelles sources de données liées au domaine du développement durable de l'environnement et de l'agriculture peuvent apparaître. En effet en France, le nombre d'exploitations ayant un accès à Internet a triplé entre 2000 et 2003 [4]. De plus, les techniques d'intégration doivent passer à l'échelle, dans le sens où elles doivent être capables de traiter un grand nombre de sources de données. Par exemple en France, plus de 367000 exploitations professionnelles [5] peuvent aspirer un jour à rendre leurs données accessibles sous forme électronique.

Contexte général

D'un point de vue général, aujourd'hui, les techniques d'analyse et d'intégration de données sont devenues des atouts majeurs pour le domaine environnemental comme pour les entreprises et les services gouvernementaux. Elles sont utilisées dans de nombreux domaines d'application comme le e-commerce, les bases de données scientifiques et l'intégration d'applications inter et intra entreprises. Ces techniques permettent en effet, un gain de temps pour regrouper et croiser l'information distribuée. Une étude récente montre notamment que le marché de l'intégration d'information a augmenté de 80% en 2003 [38] alors qu'une analyse du Gartner Group prévoit que d'ici 2006, l'intégration de données fera partie des priorités stratégiques pour les entreprises [39].

[4] http ://www.acta-informatique.fr/

[5] http ://www.agreste.agriculture.gouv.fr

L'intégration et le partage d'information provenant de sources de données distribuées et hétérogènes engendrent différentes difficultés, comme notamment :

- l'identification des sources de données pertinentes à la résolution d'une requête donnée,
- la résolution des problèmes d'hétérogénéité qui se posent à différents niveaux (par exemple, les protocoles de communication réseaux, les modèles de données et les langages d'interrogation des sources, les conflits sémantiques et syntaxiques),
- la fusion des données provenant de sources hétérogènes.

Ces difficultés sont accrues par l'augmentation permanente du volume de données disponibles sous forme électronique ainsi que par la variété des syntaxes et/ou formats de ces données.

Les problèmes de partage et d'intégration d'information ont intéressé la communauté de recherche en bases de données depuis une vingtaine d'années [74, 29] et continue aujourd'hui à faire l'objet d'actives investigations [75, 25]. Les travaux dans ce domaine ont visé à développer des techniques et des outils permettant un accès transparent à des sources de données dissiminées sur un réseau. Ils ont abouti entre autre, à la définition d'une architecture de médiation basée sur le paradigme *médiateur/wrapper* comme illustré dans la figure 1. Dans ce type d'architecture, un composant central, appelé *médiateur*, joue le rôle d'interface entre les utilisateurs et les sources de données. Le médiateur est composé d'un *schéma global*, qui offre une vue unifiée des sources de données et, d'un ensemble de *vues* décrivant le contenu des sources. Les requêtes sont alors exprimées sur le schéma global, donnant ainsi aux utilisateurs l'illusion d'interroger une base de données unique. En s'appuyant sur les informations fournies par les vues, le médiateur analyse les requêtes et les reformule en sous-requêtes exécutables par les sources de données. Avant d'être envoyée à la source de données cible, chaque sous-requête est traduite dans le langage natif de la source par le *wrapper* correspondant.

Dans la littérature, on trouve deux grandes approches de médiation qui se distinguent par le type de correspondance sémantique entre le schéma global et les sources de données. Dans la première approche, appelée Global As View (GAV), le schéma global est défini comme un ensemble de vues sur les sources de données [29] alors que dans la deuxième approche, appelée Local As View (LAV), les sources de données sont des vues sur le schéma global [63]. L'approche GAV est connue pour être peu flexible dans la mesure où l'ajout de nouvelles sources de données implique la modification du schéma global. A contrario dans l'approche LAV, l'ajout et la suppression de sources de données n'affectent pas le schéma global. Cependant, le traitement des requêtes est généralement plus difficile dans l'approche LAV que dans l'approche GAV.

Ce travail de thèse se situe dans le cadre général des systèmes de médiation suivant une approche LAV. Plus particulièrement, il s'intéresse au problème de traitement des requêtes dans ce contexte. De manière générale, la difficulté de ce problème réside dans le fait que le médiateur doit calculer les réponses à une requête donnée en utilisant uniquement les extensions des vues, autrement dit sans connaître le contenu réel des sources sous-jacentes. Un résultat fondateur dans ce domaine est donné par Abiteboul et Duschka dans [1] où la sémantique des réponses à une requête est définie formellement à travers la notion de *réponses certaines*. Intuitivement, cette notion permet de caractériser les réponses d'une requête qui sont correctes quelque soit le contenu réel des sources de données, sachant que ces réponses ont été calculées uniquement à partir des extensions des vues. Ainsi le traitement des requêtes dans un système de médiation

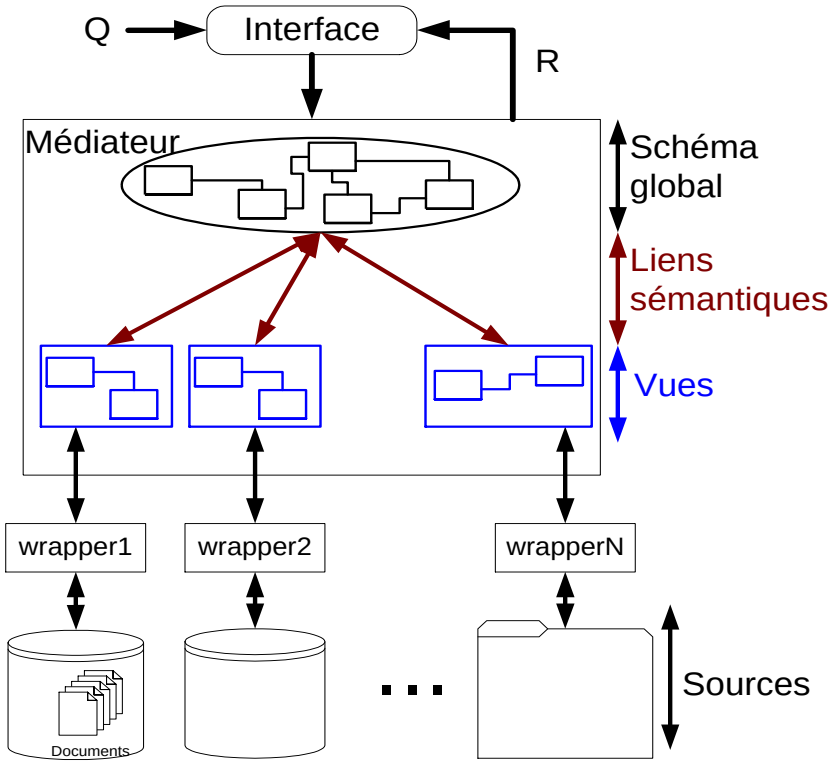

FIG. 1 – Une architecture de médiation

peut être formulé comme le problème de calculer toutes les réponses certaines d'une requête [6]. Une technique souvent utilisée pour résoudre ce problème s'appuie sur la notion de réécriture de requêtes [55, 43]. Plus précisément, le problème de la réécriture de requête en termes de vues peut s'énoncer de la façon suivante : étant donnés un ensemble de définitions de vues $\mathcal{V}$ et une requête Q, les vues et la requête étant définies sur le même schéma S, l'objectif est de reformuler Q en une expression de requête qui utilise uniquement les vues de $\mathcal{V}$ [43, 11, 7]. L'expression obtenue, appelée *réécriture*, permet d'identifier les vues, et donc les sources de données, pertinentes à la résolution de Q. Il existe plusieurs types de réécriture qui diffèrent par la relation logique définie entre la requête et ses réécritures. Une réécriture peut être équivalente à une requête Q [7] ou maximalement contenue dans Q [11] ou encore être une approximation de Q [69].

Le problème de la réécriture de requêtes en termes de vues joue un rôle important dans beaucoup d'autres applications comme, par exemple, l'optimisation de requêtes et les entrepôts de données [77, 43]. Ce problème a fait l'objet de nombreux travaux qui se distinguent par la classe de langages utilisée pour décrire le schéma global, les vues et les requêtes. La réécriture a été notamment étudiée dans le cadre des requêtes conjonctives [56, 77], en présence de contraintes arithmétiques [3], pour le cas des requêtes récursives [24], dans le cadre des lo-

[6] Ce problème est connu sous le nom de répondre à une requête en termes de vues ("query answering using views")

giques de description [7, 33, 13], pour les modèles de données semi-structurés [16, 17], et pour des langages hybrides comme CARIN [31], combinant les requêtes conjonctives et les logiques de description.

Dans ce mémoire, nous nous intéressons au problème de la réécriture de requête en termes de vues en présence de *contraintes de valeurs*. Ce problème est motivé par une application dans le domaine du développement durable dont l'objectif est de permettre l'intégration effective de sources de données agricoles hétérogènes et distribuées. Nous explicitons ci-dessous nos motivations et décrivons l'approche suivie pour résoudre ce problème.

Description du problème et motivations

Les contraintes de valeurs correspondent à la notion de type énuméré en bases de données. Ces contraintes sont implémentées dans la plupart des systèmes de gestion de bases de données modernes comme Oracle [37], DB2 [65] et SQLServer [64]. Elles permettent de spécifier les valeurs autorisées pour un attribut donné. Par exemple, à l'aide d'une telle contrainte on peut définir la classe des personnes presque trentenaires en fixant le domaine de valeurs de l'attribut age aux valeurs $\{27, 28, 29\}$.

Les contraintes de valeurs sont très utiles dans beaucoup d'applications, comme par exemple pour la vérification des contraintes d'intégrité ou pour exprimer une forme d'information incomplète [15]. De plus, ce type de contraintes a connu récemment un regain d'intérêt impulsé notamment par les travaux liés au web sémantique. En effet, les types énumérés font partie intégrante du langage d'ontologie du web OWL [7], un standard émergeant du web sémantique [8]. Notre intérêt pour ces contraintes est motivé par notre cadre applicatif. En effet dans l'application qui nous intéresse, les vues qui décrivent les sources de données ont souvent la même description. Elles se distinguent uniquement par les valeurs autorisées pour certains de leurs attributs. Par exemple, de nombreuses sources de données stockent des informations au sujet des *parcelles culturales* qui ont reçu des *pesticides* mais lors d'années différentes et/ou qui sont situées dans des communes différentes. Des vues associées à de telles sources de données peuvent être décrites de manière informelle comme suit :

$S_1.V_1$: *Parcelles culturales des communes 23200 ou 23400 ou 63450 qui ont reçu des pesticides en 1999 ou 2000*

$S_2.V_2$: *Parcelles culturales des communes 43258 ou 23400 ou 03125 qui ont reçu des pesticides en 1997 ou 2000 ou 2002*

Ces vues décrivent les contenus de deux sources de données S_1 et S_2 respectivement. Elles ont la même description, mais l'attribut numéro de commune de la vue V_1 est contraint par les valeurs 23200 ou 23400 ou 63450 tandis que l'attribut numéro de commune de la vue V_2 est contraint par les valeurs 43258 ou 23400 ou 03125. De même, l'attribut année de la vue V_1 et l'attribut année de la vue V_2 ne sont pas restreints par les mêmes valeurs. Ainsi la vue V_2 décrit une requête similaire à celle réalisable par la source de données S_1 mais concernant des parcelles culturales qui appartiennent à un ensemble de communes différentes et qui ont reçu des pesticides pendant des périodes différentes. Dans notre contexte applicatif, les contraintes de valeurs sont également utiles dans la description des requêtes. En effet, une requête typique Q

[7] http ://www.w3.org/2004/OWL/

[8] OWL est actuellement une recommandation du W3C (http ://www.w3.org/).

du domaine d'application consiste par exemple, à demander les parcelles culturales des numéros de communes 23200 ou 23400 qui ont reçu des pesticides durant les années 2001 ou 2002.

L'utilisation des contraintes de valeurs permet d'obtenir des descriptions de requêtes et de vues très fines et, de ce fait, d'améliorer le processus de réécriture. Par exemple, en considérant la contrainte de valeur sur l'attribut *année*, on peut déduire que la vue V_1 n'est pas intéressante pour répondre à Q. Par conséquent, les contraintes de valeurs peuvent être exploitées à deux fins : *(i)* pour réduire le nombre de vues candidates durant la réécriture des requêtes, permettant ainsi d'optimiser le processus de réécriture, et *(ii)* pour réduire le nombre de sources de données à solliciter, permettant ainsi de réduire les coûts de communication sur le réseau et par conséquent, d'améliorer l'évaluation des requêtes.

Approche adoptée

A notre connaissance, le problème de la réécriture de requêtes en présence de contraintes de valeurs n'a pas été exploré par les travaux existants, que ce soit dans le cadre des logiques de description ou dans celui de Datalog. Dans cette thèse, nous proposons une approche hybride pour résoudre ce problème. Cette approche s'appuie sur le cadre des logiques de description [6] pour formaliser et étudier la décidabilité de ce problème. Elle exploite ensuite un cadre de découverte de connaissances [57] pour développer des solutions algorithmiques permettant de résoudre ce problème efficacement en pratique.

Les logiques de description sont un formalisme de représentation des connaissances important de par les investigations théoriques dont elles ont fait l'objet et les nombreuses applications dans lesquelles elles ont été utilisées [6]. A notre connaissance, les logiques de description constituent un des rares formalismes dans lesquels les problèmes de modélisation et de raisonnement en présence des contraintes de valeurs ont été bien identifiés [73, 15]. En effet, dans le cadre des logiques de description, le constructeur $\mathcal{O}$, également connu sous la dénomination ONEOF, permet de spécifier des ensembles d'individus et peut donc être utilisé pour exprimer des contraintes de valeurs. Dans notre travail, nous utiliserons une forme restreinte de ce constructeur, notée $\mathcal{O}_v$, qui est suffisante pour capturer la sémantique des contraintes de valeurs. Le cadre des logiques de description nous permet de formaliser le problème de la réécriture de requêtes en présence de contraintes de valeurs, puis de montrer sa décidabilité.

Le deuxième cadre sur lequel repose notre approche est celui de la découverte de connaissances dans les bases de données. Ce domaine de recherche très actif vise à mettre au point des théories et des techniques pour extraire des connaissances intéressantes à partir de grand volume de données. Nous utilisons le cadre de découverte de connaissances introduit dans [57] pour caractériser le problème du calcul des réécritures engendrées par la présence des contraintes de valeurs. L'utilisation de ce cadre a un double intérêt. Premièrement, une formulation du problème du calcul des réécritures dans ce cadre nous permet de bénéficier d'algorithmes existants pour calculer effectivement les réécritures. Le deuxième intérêt de ce cadre est lié à sa vocation première à traiter de grands volumes de données. L'utilisation d'un cadre de découverte de connaissances permet alors de réutiliser/adapter des algorithmes existants permettant de calculer effectivement et efficacement les réécritures de requêtes même en présence d'un grand nombre de vues. Ceci est confirmé par nos expérimentations qui montrent qu'il est possible de

traiter par exemple 10000 vues, pour réécrire un simple fragment de requête.

Contributions

Cette thèse propose une approche hybride pour résoudre le problème de la réécriture de requêtes en termes de vues en présence de contraintes de valeurs, qui repose à la fois sur un cadre logique et sur un cadre de découverte de connaissances. A notre connaissance, c'est la première fois qu'un cadre de découverte de connaissances est utilisé pour résoudre ce type de problème. De manière plus précise, les principales contributions de cette thèse sont les suivantes :

- La recherche des réponses certaines à une requête Q par la méthode de la réécriture ne permet pas toujours d'obtenir toutes les réponses certaines de Q. Nous donnons une condition nécessaire et suffisante pour obtenir les réponses certaines d'une requête donnée lorsque le langage de requête et des vues est la logique $\mathcal{FL}_0(\mathcal{O}_v)$ ou $\mathcal{ALN}(\mathcal{O}_v)$. Autrement dit, il est possible dans ce contexte de répondre de façon complète aux requêtes des utilisateurs. Ce résultat nous permet de définir formellement l'instance du problème de réécriture à étudier.
- Nous étudions le problème de la réécriture de requêtes en termes de vues dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$. Cette logique permet les constructeurs de conjonction, de restriction de valeurs et l'expression de contraintes de valeurs. Nous nous intéressons dans un premier temps à cette logique afin d'appréhender les réelles difficultés apportées par l'introduction de $\mathcal{O}_v$ dans le cadre de la réécriture. Nous caractérisons pour cette logique la forme des réécritures et montrons qu'il existe un algorithme pour calculer les réécritures de requêtes en termes de vues dans $\mathcal{FL}_0(\mathcal{O}_v)$ en temps exponentiel [51].
- Ensuite, nous étudions le problème de la réécriture de requêtes en termes de vues dans une logique plus expressive, notée $\mathcal{ALN}(\mathcal{O}_v)$. $\mathcal{ALN}(\mathcal{O}_v)$ augmente l'expressivité de $\mathcal{FL}_0(\mathcal{O}_v)$ avec une forme restreinte de négation et des contraintes de cardinalité. L'utilisation de ce langage plus expressif est motivé par l'applicatif, intéressé par la possibilité d'exprimer des contraintes de cardinalité et une forme de négation dans les requêtes. De plus, la logique $\mathcal{ALN}$ est connue pour offrir un bon compromis entre l'expressivité et la complexité du raisonnement. Comme pour la logique $\mathcal{FL}_0(\mathcal{O}_v)$, nous caractérisons la forme des réécritures dans $\mathcal{ALN}(\mathcal{O}_v)$ [50]. Cette caractérisation montre que la forme des réécritures dans $\mathcal{ALN}(\mathcal{O}_v)$ est évidemment plus complexe que celle de $\mathcal{FL}_0(\mathcal{O}_v)$. Cette complexité additionnelle est due aux interactions entre $\mathcal{O}_v$ et les contraintes de cardinalité permises par la logique $\mathcal{ALN}$.
- Nous montrons que dans $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$, les problèmes de réécritures spécifiques à la présence de $\mathcal{O}_v$ peuvent bénéficier d'une formulation ensembliste. Ceci permet d'exprimer ces problèmes dans le cadre de découverte de connaissances dans les bases de données, introduit par Mannila et Toivonen dans [57]. Une caractérisation de ces problèmes dans ce cadre a un double intérêt : elle offre un moyen de calculer effectivement mais aussi efficacement ces cas de réécritures grâce aux algorithmes disponibles dans ce domaine. Elle permet alors d'envisager sérieusement le passage à l'échelle des algorithmes de réécriture.
- Nous proposons une implémentation d'un prototype de réécriture en termes de vues dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$. Cette implémentation repose sur un système de gestion de bases

de données afin de bénéficier des propriétés offertes en termes de capacité de stockage, de capacité de traitement des requêtes clients mais aussi de capacité de mises à jour des vues à distance. Cette implémentation utilise un algorithme par niveaux pour calculer effectivement et efficacement les réécritures engendrées par la présence des contraintes de valeurs. Nous avons réalisé des expérimentations sur des données synthétiques pour évaluer les performances du prototype. Les premiers résultats sont très prometteurs en termes de la capacité du prototype à traiter un grand nombre de vues.

Organisation du mémoire

Ce mémoire est organisé en quatre parties. La partie I présente un état de l'art sur les systèmes d'intégration de données, en se focalisant particulièrement sur le traitement des requêtes. La partie II est consacrée à la présentation du cadre formel basé sur les logiques de description, utilisé dans notre étude. Plus précisément, le chapitre 2 introduit les notions de base concernant les logiques de descriptions. Le chapitre 3 décrit les choix retenus pour la représentation des contraintes de valeurs dans le cadre des logiques de description et caractérise le raisonnement de subsomption dans ce cadre. La partie III est consacrée à l'étude du problème de la réécriture de requête en présence des contraintes de valeurs. Le chapitre 4 est dédiée à la formalisation du problème de la réécriture de requêtes en termes de vues dans les logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$. Le chapitre 5 se concentre sur les cas de réécritures engendrés par la présence des contraintes de valeurs et reformule les problèmes du calcul de ses réécritures en termes ensemblistes. Ce chapitre montre alors comment les techniques de fouilles de données permettent de caractériser ces problèmes, nous permettant ainsi de réutiliser et d'adapter des algorithmes existants et efficaces de fouille de données pour le calcul des réécritures engendrées par les contraintes de valeurs. Ensuite, la partie IV décrit une implantation d'un moteur de réécriture pour $\mathcal{FL}_0(\mathcal{O}_v)$ s'appuyant sur les résultats des chapitres 4 et 5. Puis nous donnons les résultats expérimentaux de cette implantation. Finalement, nous concluons et donnons quelques perspectives dans la dernière partie de ce mémoire.

Première partie

Cadre Général

Cette partie formée d'un seul chapitre, est consacrée au domaine de l'intégration d'information et plus particulièrement au problème de la résolution de requêtes dans les systèmes d'intégration. Notons que le domaine de l'intégration d'information est confronté à d'autres problèmes difficiles tels que la découverte de correspondances sémantiques entre les ontologies, la façon de les représenter et comment les exploiter [67]. Cependant nous n'aborderons pas ces thèmes dans cette thèse.

Le chapitre est organisé comme suit. La section 1.1 introduit le problème de l'intégration de l'information et décrit les deux principales approches pour le résoudre : l'approche matérialisée et l'approche virtuelle. La section 1.2 décrit les systèmes de médiation et présente chacune des deux approches de médiation GAV et LAV. La section 1.3 se focalise sur le problème du traitement des requêtes dans l'aproche LAV et présente les principaux travaux de réécriture de requêtes en termes de vues dans ce contexte. Enfin, la section 1.4 positionne le problème étudié dans cette thèse par rapport aux travaux existants.

Chapitre 1

Intégration d'informations et réécriture de requêtes en termes de vues

1.1 L'intégration d'informations

L'intégration d'informations se définit comme le problème de fusionner des données provenant de différentes sources de données distribuées et hétérogènes, en fournissant aux utilisateurs une *vue unifiée* de ces sources de données [55]. Les systèmes d'intégration permettent de répondre à ce problème en offrant aux utilisateurs une vue unifiée des sources de données sous la forme d'un *schéma global* unique, comme illustré par la figure 1.1. Ils permettent ainsi aux utilisateurs de se concentrer uniquement sur ce qu'ils cherchent (le *quoi*) et non pas sur comment l'obtenir. Deux types approches d'intégration ont été proposés pour réaliser les systèmes d'intégration :

- les approches matérialisées, comme les entrepôts de données,
- les approches virtuelles, comme les systèmes de médiation.

Dans la première approche décrite dans la figure 1.2, page 17, le schéma global est matérialisé. Ceci permet ainsi de constituer un entrepôt centralisant les données provenant de différentes sources. Dans cette approche, les données sont donc effectivement copiées des sources vers l'entrepôt. Les requêtes posées sur le schéma global sont alors **évaluées au sein de l'entrepôt**. L'avantage de cette approche est qu'elle permet de répondre efficacement aux requêtes puisque l'évaluation des requêtes est réalisée en local. En revanche, son inconvénient réside dans la nécessité de rafraîchir périodiquement les données. Ce processus peut être coûteux car il implique le transfert des données via le réseau, le formatage et le nettoyage des données. Il est bien connu que les approches entrepôt de données sont intéressantes dans le cadre d'applications qui ne sont pas fortement transactionnelles et qui ne nécessitent pas obligatoirement des versions de données très récentes (comme par exemple, dans le cas du traitement analytique ou OLAP [9] [20]).

Les systèmes de médiation, illustrés en figure 1.3, page 18, constituent la deuxième approche d'intégration. Contrairement à l'approche précédente, le schéma global, appelé ici *schéma de médiation*, est virtuel. Les données **restent stockées au niveau des sources de données** selon le format local. Les requêtes sont posées sur le schéma de médiation et évaluées effectivement

[9] On-line Analytical Processing

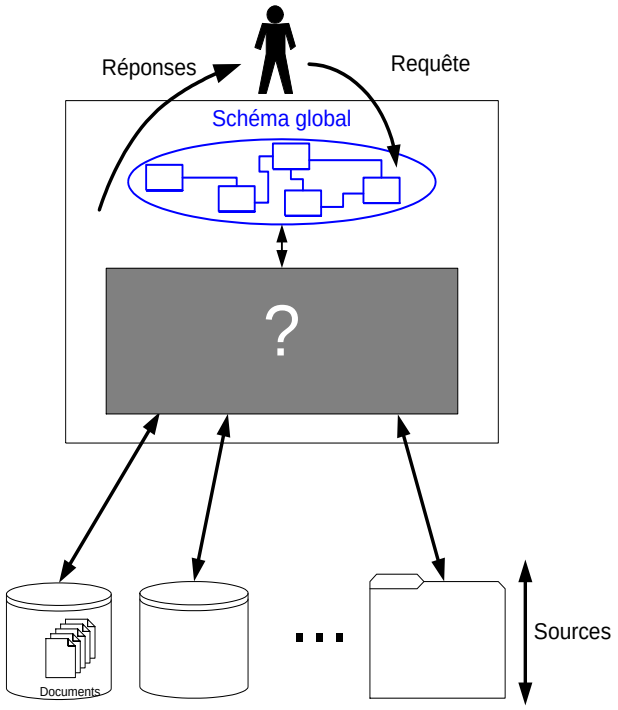

FIG. 1.1 – Un système d'intégration d'informations

au sein des sources. Pour traiter une requête, le médiateur doit être capable de la reformuler en sous-requêtes qui seront envoyées vers les sources de données pertinentes à la résolution de la requête. Dans cette architecture, les wrappers permettent de traduire les sous-requêtes exprimées dans le langage du médiateur vers le langage de la source correspondante, et après évaluation des sous-requêtes, de traduire les réponses dans le format du médiateur. L'approche de médiation est intéressante car elle ne nécessite pas le rafraîchissement des données. Son inconvénient est que l'évaluation des requêtes est coûteuse puisqu'elle implique des coûts de communication sur le réseau.

Dans la section qui suit, nous allons nous intéresser plus particulièrement aux systèmes de médiation et discuter les différentes approches de médiation.

1.2 Les systèmes de médiation

Un système de médiation peut être défini comme un triplet $< \mathcal{S}, \mathcal{V}, \mathcal{M} >$ où $\mathcal{S}$ désigne le schéma global du système de médiation, $\mathcal{V}$ l'ensemble des descriptions du contenu des sources de données, appelées *vues* et $\mathcal{M}$ les liens sémantiques entre $\mathcal{V}$ et $\mathcal{S}$ [55]. L'architecture d'un tel système est décrite par la figure 1.4. Il existe deux grands types d'approches de médiation :

– l'approche Global As View (GAV),
– l'approche Local As View (LAV).

Ces deux approches se distinguent par la façon de définir les liens sémantiques $\mathcal{M}$ ([44, 55, 71]). Dans l'approche GAV, le schéma global est défini comme un ensemble de vues sur les sources

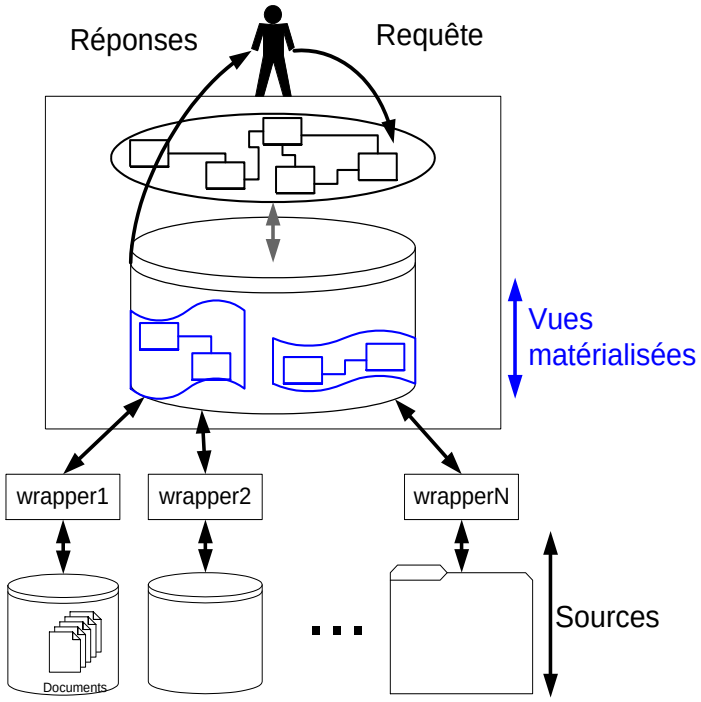

FIG. 1.2 – Une approche entrepôt de données

de données, tandis que dans l'approche LAV, les sources de données sont exprimées comme des vues sur le schéma global. Une troisième approche mixant les approches LAV et GAV, dite GLAV ([45]) a été proposée afin de profiter des deux approches précédentes.

Dans l'exemple qui suit, nous illustrons les deux approches LAV et GAV avec un système de médiation dans le domaine viticole. Pour définir les liens sémantiques entre le schéma global et les sources de données, nous utiliserons un pseudo-langage de requête.

Exemple 1

Soit le schéma global $\mathcal{S}$ illustré dans la table 1.1. Le schéma $\mathcal{S}$ est composé de 3 prédicats [10] : $Vins$, $Producteur$, $Prix$. Le prédicat $Vins$ dénote les vins du domaine ayant un nom, une $appellation$, produit une $annee$ donnée par le producteur $prod$. Le prédicat $Producteur$ spécifie les producteurs de vins $prod$ d'une $region$. Le prédicat $Prix$ dénote les vins ayant un nom, primés par le prix $intitule$.

Soient 3 sources de données décrites comme suit :

– la source 1 donne des informations sur les vins de producteurs auvergnats et est décrite au sein du médiateur par la vue V_1 : $V_1(nom, appellation, annee, prod)$.
– la source 2 donne des informations sur les vins de producteurs bordelais et est décrite au sein du médiateur par la vue V_2 : $V_2(nom, appellation, annee, prod)$.
– la source 3 donnent des informations sur les vins produits depuis 1980 qui ont reçu un prix et est décrite au sein du médiateur par la vue V_3 : $V_3(nom, intitule)$.

[10] Nous désignons par prédicat une table/relation.

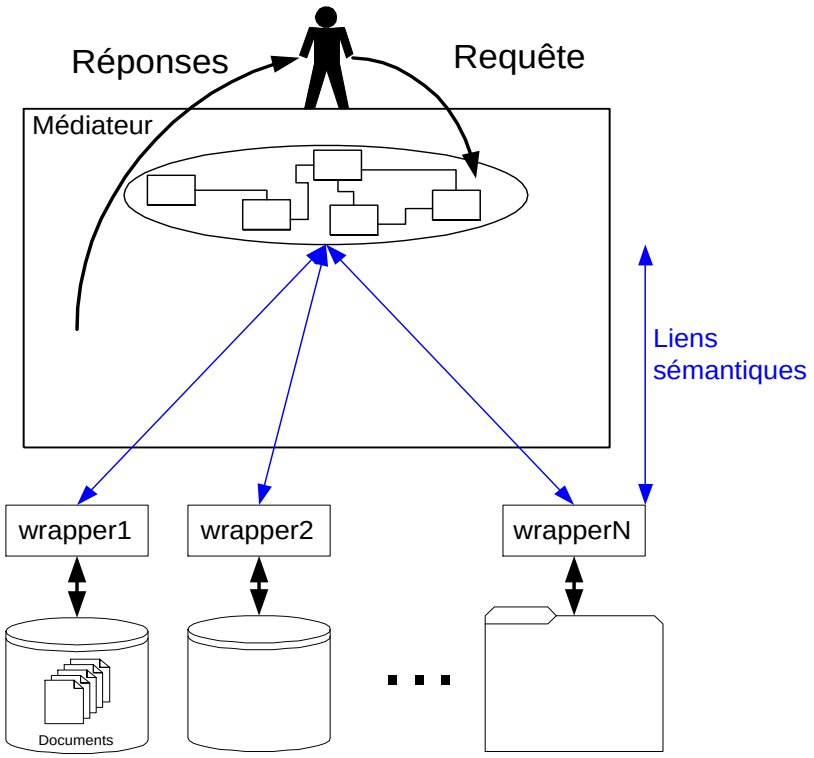

FIG. 1.3 – Approche de médiation

Schéma global $\mathcal{S}$
$Vins(nom, appellation, annee, prod)$ $Producteur(prod, region)$ $Prix(nom, intitule)$

TAB. 1.1 – Exemple de schéma global $\mathcal{S}$

Dans l'approche GAV, les liens sémantiques entre $\mathcal{S}$ et les sources sont définis en décrivant les prédicats de $\mathcal{S}$ comme des requêtes sur les vues de $\mathcal{V}$. Par exemple, comme illustrée dans la table 1.2, le prédicat $Vins$ du schéma global est défini comme étant l'union des vins se trouvant dans les sources de données 1 et 2. Les producteurs sont définis à partir de l'union des tuples $< prod,' Auvergne' >$ fournis par la vue V_1, avec les tuples $< prod,' Bordeaux' >$ fournis pas la vue V_2 par projection sur l'attribut $prod$. En effet, la vue V_1 (resp. V_2) spécifie uniquement les vins Auvergnats (resp. Bordelais).

La définition des liens sémantiques entre $\mathcal{S}$ et les sources de données, selon l'approche LAV, est donnée dans la table 1.3. On peut observer maintenant que les vues de $\mathcal{V}$ sont exprimées comme des requêtes sur $\mathcal{S}$. La vue V_1 est définie comme étant une restriction du prédicat $Vins$ de $\mathcal{S}$ aux producteurs $auvergnats$. De la même manière, V_2 est définie comme étant une restriction du prédicat $Vins$ de $\mathcal{S}$ aux producteurs $bordelais$, alors que V_3 est définie comme une restriction

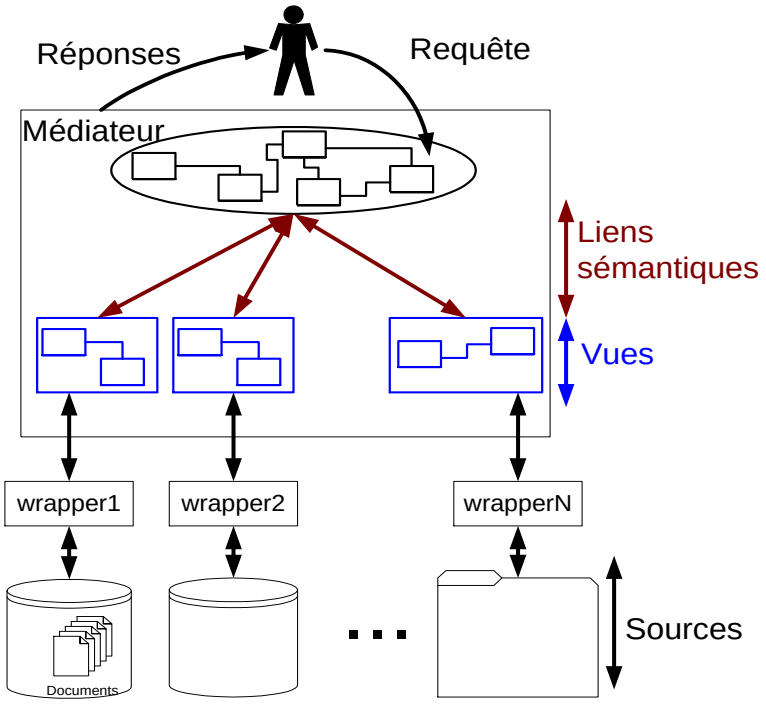

FIG. 1.4 – Système de médiation

du prédicat $Prix$ de $\mathcal{S}$ aux vins produits après 1980. ◦

Différents systèmes de médiation ont été implémentés pour les approche LAV, GAV et GLAV. Certains d'entre-eux sont donnés dans la table 1.4.

Dans un système de médiation suivant l'approche GAV, l'ajout de sources de données implique une modification du schéma global. De plus, la définition des liens sémantiques peut s'avérer difficile dans le cas où le nombre de sources de données est grand et lorsque les prédicats du médiateur induisent des interactions complexes entre les vues. L'approche LAV par contre, conduit naturellement à une définition modulaire des correspondances sémantiques. En effet, les vues sont exprimées en termes du schéma global. Ainsi l'ajout d'une source de données consiste à isoler une partie du schéma pour laquelle la source peut donner des informations, puis à la déclarer sous la forme d'une requête exprimée sur le schéma $\mathcal{S}$. Par

$\mathcal{S}$ dans l'approche GAV
$Vins(nom, appellation, annee, prod) := \{\{nom, appellation, annee, prod\}$ $\mid V_1(nom, appellation, annee, prod) \cup V_2(nom, appellation, annee, prod)\}$
$Producteur(prod, region) := \{\{prod,' Auvergne'\} \mid V_1(nom, appellation, annee, prod)\}$ $\cup\{\{prod,' Bordeaux'\} \mid V_2(nom, appellation, annee, prod)\}$
$Prix(nom, intitule) := \{\{nom, intitule\} \mid V_3(nom, intitule)\}$

TAB. 1.2 – Description de $\mathcal{S}$ dans l'approche GAV

Descriptions des vues dans l'approche LAV
$V_1(nom, appellation, annee, prod) := \{\{nom, appellation, annee, prod\}$ $\mid Vins(nom, appellation, annee, prod), Producteur(prod,' Auvergne')\}$
$V_2(nom, appellation, annee, prod) := \{\{nom, appellation, annee, prod\}$ $\mid Vins(nom, appellation, annee, prod), Producteur(prod,' Bordelais')\}$
$V_3(nom, intitule) := \{\{nom, intitule\} \mid Prix(nom, intitule),$ $Vins(nom, appellation, annee, prod), annee > 1980\}$

TAB. 1.3 – Description des vues en termes de $\mathcal{S}$ dans l'approche LAV

GAV	LAV	GLAV
TSIMMIS [29]	Information Manifold [56]	Piazza [45]
MIX [10]	OBSERVER [63]	PICSEL [31]
LeSelect [19]		XPeer [12]

TAB. 1.4 – Exemples de système de médiation

conséquent l'approche LAV facilite le passage à l'échelle en termes du nombre de vues qu'elle peut accepter. De plus, elle est plus flexible car elle permet l'évolution des sources. En revanche, la reformulation de requêtes en termes des vues est généralement aisée dans l'approche GAV que dans l'approche LAV. En effet, dans l'approche GAV, la réécriture consiste à déplier la requête en remplaçant les prédicats par leurs définitions. Le résultat obtenu est une expression en termes de vues comme illustré dans l'exemple 2.

Exemple 2
Considérons maintenant une requête Q qui demande les vins produits en 2000 qui ont reçu un prix, i.e. $Q = \{\{V\} \mid Vins(V, appellation, 2000, prod), Prix(V, intitule)\}$.
Dans l'approche GAV, pour obtenir la reformulation Q' de Q en termes des vues, il suffit généralement de remplacer les prédicats de la requête par leurs descriptions. Ainsi, à l'aide de la table 1.2, on obtient $Q' = \{V \mid V_1(V, appellation, annee, prod) \cup V_2(V, appellation, annee, prod)\ , V_3(V, intitule)\}$. ◦

Dans l'approche LAV, le problème de la réécriture est cependant plus difficile. Le médiateur est obligé d'*inférer* les prédicats de la requête en utilisant uniquement les vues, comme illustré par l'exemple 3.

Exemple 3
Soit la requête $Q = \{V \mid Vins(V, appellation, 2000, prod), Prix(V, intitule)\}$. Considérons les vues décrites dans la table 1.3.
Dans l'approche LAV pour réécrire Q, on cherche les combinaisons de vues qui impliquent logiquement Q. Ainsi les réponses obtenues à partir de cette combinaison sont justes.
D'après la table 1.3, on a V_1 (resp. V_2) qui est une restriction des $Vins$ de $\mathcal{S}$ aux producteurs $auvergnats$ (resp. $bordelais$). Ainsi V_1, respectivement V_2, impliquent logiquement le

prédicat $Vins$. On cherche cependant les vins qui ont été primés. Aussi il nécessaire de restreindre ces vues à l'aide de la vue V_3 qui donne les vins primés, produits après 1980. On obtient alors l'expression Q'' suivante : $Q' = \{V \mid V_1(V, appellation, annee, prod) \cup V_2(V, appellation, annee, prod), V_3(V, intitule)\}$. Il reste à restreindre l'année de production du vin à l'année 2000. On obtient alors la réécriture Q'' de Q qui est identique à l'expression Q'.
Ici, la réécriture de Q a été obtenue à l'aide d'inférences sur les vues. ◦

1.3 Traitement des requêtes dans l'approche LAV

Nous présentons maintenant le problème de répondre à une requête en utilisant les vues dans le cadre d'un système de médiation suivant une approche LAV et décrivons une technique pour le résoudre.

1.3.1 Répondre aux requêtes en utilisant des vues

Soit $< \mathcal{S},\mathcal{V},\mathcal{M} >$ un système de médiation. Considérons une requête Q sur le schéma $\mathcal{S}$. Le médiateur doit être capable de calculer les réponses de Q à partir des extensions des vues Ce problème est connu sous le nom de répondre à une requête en utilisant les vues (”query answering using views”) [1, 43, 32]. Pour préciser ce problème, supposons que les différentes sources de données forment une seule base de données, appelées base de données globale, notée $\mathcal{D}$ (figure 1.5). Le médiateur qui est chargé du traitement de la requête, ne connaît pas le contenu réel de $\mathcal{D}$. Il n'a en réalité qu'une vision partielle du contenu de $\mathcal{D}$ à travers les extensions des vues. Cependant, plusieurs voire une *infinité* de bases de données peuvent conduire aux mêmes extensions des vues.

Exemple 4
Soit $\mathcal{S}$ un schéma de médiation formé d'un seul prédicat $P(X, Y)$.
Sot V_1 une vue, définie comme la projection sur l'attribut X du prédicat $P(X, Y)$, i.e. $V_1(X) := P(X, Y)$.
Soit $V_1(John)$ l'extension de la vue V_1.
Alors $\mathcal{D}$ peut être formée d'un seul tuple $< John, Marie >$ ou de plusieurs comme montré dans la table 1.5, à partie du moment où il est possible d'obtenir l'extension $V_1(John)$ à partir de $\mathcal{D}$.

P(X,Y)	John	Julie
	John	Sophie
	Romain	Fabienne

TAB. 1.5 – Base de données $\mathcal{D}$

◦

De manière plus générale, $\mathcal{D}$ peut être vue comme une base de données incomplète [1, 55]. Pour répondre à une requête dans un tel contexte, une première difficulté consiste à définir la

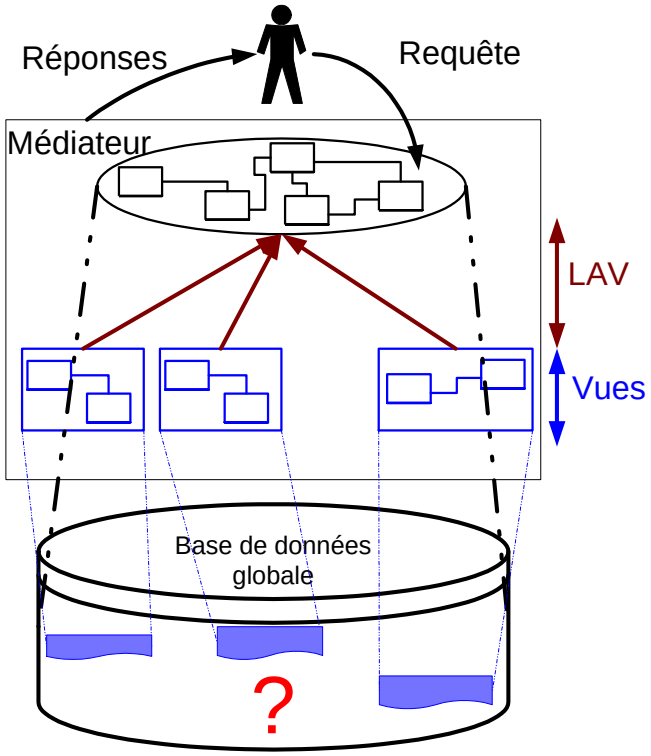

FIG. 1.5 – Répondre à une requête en ayant une vison partielle de la base de donnée

sémantique d'une réponse à une requête. Une telle sémantique est définie dans [1] via la notion de *réponses certaines*. Soit $V \in \mathcal{V}$ une vue. Nous notons par $ext(V)$ l'extension de V, par $Q(\mathcal{D})$ (resp $V(\mathcal{D})$) l'évaluation de Q (resp. de V) sur la base de données $\mathcal{D}$. Un tuple t est une réponse certaine de Q, si t appartient à l'ensemble des réponses de $Q(\mathcal{D})$ et ce quelque soit la base de données $\mathcal{D}$ conforme à l'extension des vues. Ainsi, une réponse certaine est une réponse *juste* de Q quelque soit le contenu réel de la base de données $\mathcal{D}$, à partir du moment où $\mathcal{D}$ conduit aux extensions des vues connues par le médiateur (i.e. $\mathcal{D}$ est conforme aux extensions des vues). La notion de conformité d'une base de données par rapport aux extensions des vues est définie en fonction des deux hypothèses suivantes :

- l'hypothèse du monde **fermé** (Closed World Assumption, CWA). Dans cette hypothèse, les sources de données sont supposées complètes. Dans ce cas, une base de données est conforme aux extensions des vues si $ext(V) = V(\mathcal{D})$. Dans l'exemple 1, selon cette hypothèse, la vue V_1 fournirait de manière exhaustive *tous* les vins de producteurs bordelais.
- l'hypothèse du monde **ouvert** (Open World Assumption, OWA). Dans cette hypothèse, les sources de données sont supposées incomplètes. Dans ce cas, une base de données est conforme aux extensions des vues si $ext(V) \subseteq V(\mathcal{D})$. Par exemple, selon cette hypothèse, la vue V_1 ne fournit qu'un sous-ensemble des vins de producteurs bordelais.

L'hypothèse du monde ouvert est une hypothèse appropriée aux systèmes d'intégration, tandis que l'hypothèse du monde fermé s'applique bien dans le contexte des bases de données classiques, par exemple pour l'optimisation de requêtes. Dans les systèmes d'intégration, les sources de données peuvent être incomplètes : il n'est pas toujours possible de garantir qu'une source de données va fournir toutes les réponses qui vérifient la description de la source V. Dans

le contexte des bases de données ou des entrepôts de données, toutes les données existantes sont centralisées au sein de la base de données. Aussi l'extension d'une vue V est susceptible de contenir toutes les réponses qui vérifient V.

Illustrons maintenant la notion de réponse certaine dans le cadre de ces hypothèses.

Exemple 5
Soit $\mathcal{S}$ formé d'un seul prédicat : $Maries(X, Y)$.
Soit $\mathcal{D}$ une instance de base de données globale.
Soit $Q(X, Y) := Maries(X, Y)$.
Soient deux vues V_1 et V_2 telles que :
$V_1(X) := Maries(X, Y)$ et
$V_2(Y) := Maries(X, Y)$.
Supposons que les extensions des vues sont $V_1(Patrick)$ et $V_2(Fabienne)$.
Dans l'hypothèse du monde ouvert, on a $V_1(Patrick) \subseteq V_1(\mathcal{D})$ et $V_2(Fabienne) \subseteq V_2(\mathcal{D})$. Ainsi on ne peut rien déduire concernant l'identité du conjoint de $Patrick$. Notamment, on n'est pas sûr que le tuple $(Patrick, Fabienne)$ soit une réponse correcte de Q. Ce tuple n'est donc pas une réponse certaine de Q dans l'hypothèse du monde ouvert. En effet, on peut aisément exhiber une base de données globale $\mathcal{D}$ conforme aux extensions des vues données précédemment, dans laquelle le tuple $(Patrick, Fabienne)$ n'est pas une réponse juste. Un exemple d'une telle base de données est donnée dans la table 1.6.

$Maries(X, Y)$

Patrick	Marie
Jean	Fabienne
Eric	Sophie
Frédéric	Fabienne

TAB. 1.6 – Exemple de base de données $\mathcal{D}$

Considérons maintenant l'hypothèse du monde fermé. On a exactement $V_1(Patrick) = V_1(\mathcal{D})$ et $V_2(Fabienne) = V_2(\mathcal{D})$.
Ainsi dans une base de données globale conforme aux extensions des vues, la projection de $Maries(X, Y)$ sur X ne peut être que $Patrick$ et la projection de $Maries(X, Y)$ sur Y ne peut être que $Fabienne$. Dans ce cas, une seule base de données est possible. Elle contient comme unique tuple $(Patrick, Fabienne)$. Ainsi on peut déduire de ce cas que le tuple $(Patrick, Fabienne)$ est une réponse certaine de Q. Notons que pour inférer ce résultat, il est nécessaire d'effectuer un raisonnement sur la cardinalité des relations.
Ceci explique pourquoi le problème de calculer toutes les réponses certaines d'une requête Q est connu pour être généralement plus difficile sous l'hypothèse du monde fermé [55]. ◦

1.3.2 Différentes approches pour la réécriture de requêtes

Une approche souvent utilisée dans les systèmes de médiation pour calculer les réponses à une requête donnée, consiste à reformuler la requête en termes des vues. L'expression ainsi obtenue est ensuite transformée en sous-requêtes que les sources de données peuvent évaluer.

Le problème de reformuler une requête en termes des vues est connu sous le nom de *réécriture de requête en termes de vues* [55, 43, 32]. Informellement, une requête Q' est une réécriture d'une requête Q si (i) Q' réfère uniquement les vues de $\mathcal{V}$ et (ii) Q' est contenue dans Q (i.e. $Q'(\mathcal{D}) \subseteq Q(\mathcal{D}),\ \forall \mathcal{D}$) ([43]). De plus, si Q' est équivalente à Q (i.e. $Q'(\mathcal{D}) = Q(\mathcal{D}),\ \forall \mathcal{D}$) alors Q' est dite *réécriture équivalente* de Q [43]. Les réécritures équivalentes ont surtout été utilisées pour l'optimisation de requête. Ce type de réécriture est cependant trop restrictif dans le cadre de la médiation car, dans ce contexte, il n'est pas toujours possible de trouver une réécriture équivalente à une requête donnée [55]. De plus, à notre avis, la notion de réécriture équivalente n'a pas beaucoup d'intérêt pratique dans le contexte de l'hypothèse du monde ouvert. Notons qu'une définition plus précise de réécriture équivalente telle que donnée dans [68] définit Q' comme étant équivalente à Q si l'évaluation de Q' sur les extensions des vues donne le même résultat que l'évaluation de Q sur $\mathcal{D}$. En considérant cette définition, la notion de réécriture n'a plus beaucoup de sens dans l'hypothèse du monde ouvert. En effet, dans un tel contexte, les réécritures équivalentes ne peuvent pas exister puisque les vues sont incomplètes.

C'est pourquoi la notion de réécritures maximalement contenues est généralement utilisée à la place des réécritures équivalentes. Informellement, une réécriture Q' exprimée dans le langage $\mathcal{L}$ est dite *maximalement contenue* dans Q s'il n'existe pas de réécriture Q'' dans $\mathcal{L}$, qui fournisse un strict sur-ensemble de réponses de Q'. Ces réécritures sont intéressantes car elles permettent de trouver le maximum de réponses certaines. Selon le langage de réécriture considéré, la réécriture maximale ne garantit pas toujours qu'on puisse trouver **toutes** les réponses certaines de Q [18].

Le problème de la réécriture a été étudié pour différentes classes de langages utilisées pour décrire les requêtes, les vues et le langage de réécriture. Soient trois langages $\mathcal{L}_Q$, $\mathcal{L}_V$ et $\mathcal{L}_R$, permettant de décrire respectivement les requêtes, les vues et le langage de réécriture. Nous donnons ci-dessous un panorama des travaux dans le domaine de la réécriture, en précisant pour chacun d'entre-eux $\mathcal{L}_Q$, $\mathcal{L}_V$ et $\mathcal{L}_R$ et le type de réécriture considéré.

– Cas où $\mathcal{L}_Q$, $\mathcal{L}_V$ et $\mathcal{L}_R$ sont des sous-langages de datalog :
  [21] s'est intéressé au problème du calcul des réécritures quand les vues et les requêtes sont des requêtes conjonctives. Plus précisément, cet article montre que si les vues ou la requête contiennent un prédicat binaire alors le problème de calculer une réécriture d'une requête est NP-complet. En revanche, si les vues et la requête sont uniquement formées de prédicats unaires alors le problème du calcul des réécritures se résout en temps polynomial. [56] s'est intéressé au calcul des réécritures maximalement contenues d'une requête donnée. Dans le cas où les requêtes et les vues sont des requêtes conjonctives, il suffit de chercher des réécritures exprimées comme des unions de requêtes conjonctives. Les auteurs ont également proposé un algorithme appelé "Bucket" pour calculer ces réécritures. Les résultats donnés dans [56] s'étendent naturellement au cas où les requêtes sont des unions de requêtes conjonctives. L'approche de réécriture basée sur l'algorithme "Bucket" peut s'étendre au cas où les vues contiennent des contraintes arithmétiques, de la forme $x\theta y$ où x,y sont des variables ou des constantes et $\theta \in \{<, >, \neq, =\}$ [43]. Dans [23], il est montré que le problème du calcul des réécritures maximalement contenues lorsque la requête est un programme datalog quelconque est décidable et que les vues sont des requêtes conjonctives. Dans ce cas, les réécritures sont également des programmes datalog. Un algorithme appelé "Inverse-Rule" a été proposé pour calculer les réécritures

maximales d'une requête dans ce contexte. Le cas où les contraintes arithmétiques sont autorisées dans les requêtes a été étudié dans [3]. Plus précisément, il est montré que si les requêtes et les vues sont des requêtes conjonctives avec des prédicats de comparaison de type $X < c$ ou $X > c$ où c est une constante, alors le langage utilisé pour exprimer les réécritures doit être datalog. Dans [3], les auteurs donnent un récapitulatif des différents travaux sur la réécriture en présence des contraintes arithmétiques. Dans [2], le problème de la réécriture maximale est abordé lorsque les requêtes et les vues peuvent être des unions des requêtes conjonctives. Plus précisément si les requêtes sont un programme datalog arbitraire et que les vues sont des unions de requêtes conjonctives alors la réécriture maximale est un programme datalog non récursif étendu au prédicat de différence $\neq$.

– Cas où $\mathcal{L}_Q$, $\mathcal{L}_V$ et $\mathcal{L}_R$ sont des logiques de description :
Dans ce cas, les requêtes et les vues sont des descriptions de concepts et le test de l'inclusion entre deux requêtes revient à un test de subsomption. Le problème de la réécritures dans des langages purement terminologiques a été étudié pour deux types de réécritures : la réécriture maximalement contenue [11] et la réécriture équivalente [7]. [11] s'intéresse à la décidabilité du problème de la réécritures maximalement contenues quand les requêtes et les vues sont exprimées dans la logique $\mathcal{ALN}$ (resp. dans $\mathcal{ALCNR}$). La logique $\mathcal{ALN}$ permet une forme restreinte de négation, la conjonction, la restriction de valeurs, la restriction sur les cardinalités tandis que $\mathcal{ALCNR}$ étend $\mathcal{ALN}$ avec la négation généralisée et la conjonction de rôles. Dans ce cadre, il est montré que les réécritures sont des unions de requêtes conjonctives. De plus, dans $\mathcal{ALN}$, le problème de la réécriture est décidable en temps polynomial.
Dans [7], le problème du calcul des réécritures équivalentes, appelé réécriture de concepts en utilisant une terminologie a été étudié. L'objectif est de calculer des réécritures équivalentes d'un concept donné et optimales par rapport à un critère donné. Le critère d'optimalité utilisé dans [7] concerne la minimisation de la taille des réécritures. De plus, le problème est étudié dans un cadre un peu particulier car les réécritures peuvent être partielles, dans le sens où elles ne sont pas uniquement formées de "vues". Le problème est étudié dans le cadre des logiques de description $\mathcal{FL}_0$, $\mathcal{ALN}$, $\mathcal{ALE}$ et $\mathcal{ALC}$. Pour les logiques $\mathcal{FL}_0$, $\mathcal{ALN}$ et $\mathcal{ALE}$, le problème de l'existence d'une réécriture est NP-difficile tandis que dans $\mathcal{ALC}$, ce problème est PSpace-difficile.

– Cas des langages hybrides :
Dans [11], les auteurs s'intéressent également au problème des réécritures maximalement contenues dans le cadre du langage hybride CARIN. Le langage CARIN combine datalog avec les logiques de description permettant ainsi de décrire des concepts à l'aide des constructeurs des logiques et des règles datalog dont les sous-buts sont construits avec des descriptions de concepts. Si les requêtes sont des descriptions dans $\mathcal{ALN}$ et les vues des descriptions dans CARIN-$\mathcal{ALN}$ telles qu'elles ne contiennent pas de variables existentielles, alors il est montré que pour calculer la réécriture maximale d'une requête Q, il suffit de cherche des réécritures qui sont des unions de requêtes conjonctives. [70] étend les résultats de [11] de la manière suivante. Si les vues sont des descriptions dans $\mathcal{ALN}$ et les requêtes sont décrites dans CARIN-$\mathcal{ALN}$, alors il est toujours possible de calculer

l'ensemble fini des réécritures conjonctives maximalement contenues d'une requête.

Dans la table 1.7, nous récapitulons les principaux résultats concernant la réécriture maximalement contenue.

$\mathcal{L}_Q$	$\mathcal{L}_V$	$\mathcal{L}_R$	Résultats et références
$\mathcal{ALCNR}$ ou $\mathcal{ALN}$	$\mathcal{ALCNR}$ ou $\mathcal{ALN}$	$\{\sqcap, \sqcup\}$	Calcul d'une réécriture est décidable [11]
CARIN-$\mathcal{ALN}$	CARIN-$\mathcal{ALN}$	$\mathcal{CQ}$	Calcul d'un ensemble fini de réécritures est décidable [11] s'il n'y a pas de variables existentielles dans les vues
CARIN-$\mathcal{ALN}$	$\mathcal{ALN}$	$\mathcal{CQ}$	Calcul d'un esemble fini de réécritures est décidable [70]
$\mathcal{CQ}$	$\mathcal{CQ}$	union de $\mathcal{CQ}$	Trouver une réécriture est décidable [36]
datalog	$\mathcal{CQ}$	datalog	Trouver une réécriture est décidable [23]
$\mathcal{CQ}$ avec les comparaisons arithmétiques $(\leq, \geq)$ avec des constantes	$\mathcal{CQ}$ avec les comparaisons arithmétiques	union de $\mathcal{CQ}$ avec les comparaisons arithmétiques $(\leq, \geq)$ avec des constantes	Trouver une réécriture est décidable [3]

$\mathcal{CQ}$ désigne les requêtes conjonctives.

TAB. 1.7 – Tableau récapitulatif des travaux sur la réécriture maximalement contenue

1.4 Problème étudié

Dans ce mémoire, nous nous intéressons au problème de la réécriture de requêtes en termes de vues en présence de *contraintes de valeurs*. Nous nous plaçons dans un système de médiation suivant une approche LAV et vérifiant l'hypothèse du monde ouvert. Ce problème est à notre connaissance ouvert. Pour résoudre ce problème, nous utilisons le cadre formel des logiques de description. Nous proposons alors un moyen de représenter les contraintes de valeurs et de raisonner en leur présence dans le cadre de deux logiques de description, notées $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$. Puis nous formalisons et résolvons le problème de la réécriture dans ce contexte. Enfin nous rattachons le problème du calcul des réécritures à un cadre formel de découverte des connaissances afin de bénéficier d'algorithmes existants pour calculer effectivement et efficacement les réécritures de requête.

Deuxième partie

Représentation des contraintes de valeurs et raisonnements dans le cadre des logiques de description

Cette partie est divisée en deux chapitres. Le chapitre 2 présente les notions de base associées aux logiques de description alors que le chapitre 3 décrit les choix retenus pour la modélisation des contraintes de valeurs, ainsi que deux langages $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ qui seront utilisés dans la suite de notre étude.

Chapitre 2

Prérequis : Les Logiques de Description

Ce chapitre s'inspire principalement de la référence [6].

2.1 Introduction

Les logiques de description sont un formalisme de représentation des connaissances, basé sur un sous-ensemble des logiques du premier ordre. Elles permettent de représenter un domaine d'application en définissant les concepts pertinents du domaine, i.e. la terminologie du domaine, et des propriétés sur les objets et les individus du domaine. Les logiques de description tiennent leur origine des réseaux sémantiques et des langages à base de frames ([6], chapitres 1 et 2). Cependant contrairement à ces formalismes de représentation des connaissances, elles disposent d'une sémantique formelle garante de procédures de raisonnement rigoureuses. Les raisonnements dans le contexte des logiques de description, permettent d'inférer de nouvelles connaissances à partir des connaissances explicites des concepts. Par exemple, le test d'inconsistance d'un concept donné est un raisonnement qui permet de déduire si ce concept est incohérent (i.e. il contient des informations contradictoires), tandis que le test de la subsomption entre deux concepts permet de déterminer si un concept est plus général/spécifique qu'un autre.

Depuis une vingtaine d'années, les logiques de description ont fait l'objet de nombreux travaux qui ont abouti à des résultats théoriques importants ainsi qu'au développement de systèmes de représentation des connaissances, aussi appelés systèmes terminologiques. Les travaux théoriques ont plus précisément porté sur l'étude de la complexité des raisonnements et ont contribué à donner une vision claire du rapport entre la complexité des raisonnements et l'expressivité du langage utilisé. D'un point de vue pratique, de nombreux efforts ont été consacrés à l'optimisation des systèmes terminologiques. Les systèmes terminologiques actuels sont relativement performants malgré une complexité du raisonnement généralement élévée. Par exemple, pour la plupart des logiques, le test de subsomption est exponentiel. Les classifieurs ONTOCLASS de PICSEL [30], RACER [42], CLASSIC [15] et FaCT [46] sont des exemples de systèmes terminologiques. A titre d'exemple de performance, le système FaCT [11] est capable de classer 1200 concepts en moins de 160 secondes [48].

D'un point de vue plus applicatif, les logiques de description ont été utilisées dans différents domaines d'application comme par exemple, les bases de données ([6], chapitre 16) ou

[11] http ://owl.man.ac.uk/factplusplus/

le traitement du langage naturel ([6], chapitre 15). Aujourd'hui, elles sont également utilisées dans le cadre du web sémantique[12]. Les logiques de description sont notamment à la base du langage d'ontologie du web OWL (Web Ontology Language) [13] du $W3C$ (World Wide Web Consortium) [14].

2.2 Notions de base

Les logiques de description fournissent un ensemble de constructeurs afin de représenter un domaine d'application en termes de classes d'individus (prédicats unaires) appelés *concepts*, et de *rôles* (prédicats binaires entre individus). Un exemple de concept qu'on peut construire avec les constructeurs de conjonction $\sqcap$ et de cardinalité $(\geq\ n\ R)$ où n est un entier positif et R est un rôle, est le concept $Parent$. Ce concept qui dénote les individus du domaine qui sont des personnes et qui ont au moins un enfant est décrit par l'expression suivante :

$$Personne \sqcap \geq\ 1\ Enfant.$$

Informellement, un concept peut être *atomique* ou complexe dans le sens où il peut être formé à partir d'autres concepts connectés à l'aide de constructeurs. Ainsi le concept $Personne$ est atomique tandis que le concept $Parent$ est complexe.

Noms des Constructeurs	Constructeurs	$\mathcal{FL}_0$	$\mathcal{ALN}$
Concept Universel	$\top$	$\times$	$\times$
Concept Bottom	$\bot$		$\times$
Négation atomique	$\neg A$		$\times$
Conjonction	$C \sqcap D$	$\times$	$\times$
Restriction Universelle	$\forall R.C$	$\times$	$\times$
Restriction de cardinalité inférieure	$\leq\ nR$		$\times$
Restriction de cardinalité supérieure	$\geq n\ R$		$\times$

TAB. 2.1 – Syntaxe des constructeurs définissant $\mathcal{FL}_0$ et $\mathcal{ALN}$

Il existe différentes logiques de description, chacune définie par un ensemble de constructeurs donné. Dans cette thèse, nous nous sommes intéressés aux logiques de description $\mathcal{FL}_0$ et $\mathcal{ALN}$ dont les constructeurs sont spécifiés dans la table 2.1, en colonne 3 et 4 respectivement. Ces constructeurs sont décrits plus précisément ci-dessous :

- le concept universel, noté $\top$, spécifie l'ensemble de tous les individus du domaine.
- le concept bottom, noté $\bot$, spécifie l'ensemble vide, i.e. un concept inconsistant.
- la négation atomique d'un concept atomique A, notée $\neg A$, dénote l'ensemble des individus du domaine qui ne sont pas des instances de A.
- la conjonction entre deux concepts C et D, noté $C \sqcap D$, spécifie la classe des individus qui sont à la fois des instances des concepts C et D.

[12] http ://www.w3.org/2001/sw/
[13] http ://www.w3.org/2004/OWL/
[14] http ://www.w3.org/

– le quantificateur universel, noté $\forall R.C$, désigne les individus dont les images de R sont nécessairement des instances de C ainsi que les individus qui n'ont pas d'images par le rôle R.
– les restrictions de cardinalité inférieure et supérieure, notées $\leq\ nR$ et respectivement $\geq nR$ spécifient les individus du domaine qui sont restreints par au plus n, respectivement au moins n, individus, via le rôle R.

2.2.1 Syntaxe de $\mathcal{FL}_0$ et $\mathcal{ALN}$

Les descriptions de concepts dans la logique $\mathcal{FL}_0$ sont formées selon la syntaxe définie ci-dessous.

Définition 2.2.1 (Syntaxe de $\mathcal{FL}_0$)
Soient $\mathcal{C}$ un ensemble de noms de concepts et $\mathcal{R}$ un ensemble de noms de rôles. Soient $A \in \mathcal{C}$ et $R \in \mathcal{R}$. Les descriptions de concepts (ou tout simplement les concepts) dans $\mathcal{FL}_0$ sont définies suivant les règles suivantes :

$$
\begin{array}{rll}
C, D \ \rightarrow & A \mid & \text{(concept atomique)} \\
& \top \mid & \text{(concept universel)} \\
& C \sqcap D \mid & \text{(intersection)} \\
& \forall R.C & \text{(restriction de valeurs)}
\end{array}
$$

L'exemple suivant illustre le domaine du développement durable à l'aide de description de concepts dans la logique $\mathcal{FL}_0$.

Exemple 6

– *ParcelleCulturale* dénote l'ensemble des individus du domaine qui sont des parcelles culturales.
– *ZoneTraitée* $\sqcap$ *ZoneIrriguée* spécifie la classe des individus qui sont à la fois des instances de ZoneTraitée et de ZoneIrriguée.
– *ParcelleCulturale* $\sqcap \forall$*traiteeP.Pesticide* désigne les parcelles culturales qui ne sont pas traitées ou qui ont uniquement été traitées en pesticides.
– *ParcelleCulturale* $\sqcap \forall$*NumCommune.Entier* spécifie la classe des parcelles culturales dont les numéros de commune, s'ils existent, sont nécessairement des entiers.

$\circ$

En plus des constructeurs fournis par $\mathcal{FL}_0$, la logique $\mathcal{ALN}$ contient le constructeur $\bot$, les restrictions de cardinalités $(\leq\ n\ R)$ et $(\geq\ n\ R)$ où n est un entier positif et R un nom de rôle, et un forme de négation $\neg A$. La définition ci-dessous donne les règles de construction des concepts dans $\mathcal{ALN}$.

Définition 2.2.2 (Syntaxe de $\mathcal{ALN}$)
Soient $\mathcal{C}$ un ensemble de noms de concepts et $\mathcal{R}$ un ensemble de noms de rôles. Soient $A \in \mathcal{C}$ et $R \in \mathcal{R}$. Soit n un entier positif.

Les descriptions de concepts (ou tout simplement les concepts) dans $\mathcal{ALN}$ sont définies comme suit :

$C, D \rightarrow$	$A \mid$	(concept atomique)
	$\neg A \mid$	(négation atomique)
	$\top \mid$	(concept universel)
	$\bot \mid$	(concept bottom)
	$C \sqcap D \mid$	(intersection)
	$\forall R.C \mid$	(restriction de valeurs)
	$(\leq\ n\ R) \mid$	(restriction de cardinalité inférieure)
	$(\geq\ n\ R)$	(restriction de cardinalité supérieure)

Notons que la négation dans $\mathcal{ALN}$ s'applique uniquement sur des concepts atomiques et non sur des descriptions complexes. Dans $\mathcal{ALN}$, on peut par exemple, décrire la classe des instances de parcelles culturales qui ont été au moins une fois traitées en pesticides, comme suit :

$$ParcelleCulturale \sqcap (\forall traitee.Pesticide) \sqcap (\geq\ 1\, traitee).$$

2.2.2 Sémantique de $\mathcal{FL}_0$ et $\mathcal{ALN}$

Afin de définir une sémantique formelle pour les logiques $\mathcal{FL}_0$ et $\mathcal{ALN}$, nous considérons dans ce mémoire, la sémantique déclarative des logiques de description [6]. Cette sémantique repose sur la notion d'*interprétation* définie comme suit.

Une interprétation $\mathcal{I}$ est un couple $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ où $\Delta^{\mathcal{I}}$ est un ensemble non vide, appelé *domaine d'interprétation* et $.^{\mathcal{I}}$ est une *fonction d'interprétation*. La fonction d'interprétation $.^{\mathcal{I}}$ associe tout concept atomique A à un sous-ensemble de $\Delta^{\mathcal{I}}$, noté $A^{\mathcal{I}}$ et tout rôle R à une relation binaire $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. La fonction d'interprétation s'étend aux concepts décrits dans $\mathcal{FL}_0$ et $\mathcal{ALN}$. Tout concept est alors interprété à l'aide de la sémantique des constructeurs de $\mathcal{FL}_0$ et $\mathcal{ALN}$ donnée dans la table 2.2, où n dénote un entier positif, P un nom de concept, A un concept atomique et le symbôle $|.|$ dénote la cardinalité d'un ensemble.

Constructeurs	*Sémantique*	$\mathcal{FL}_0$	$\mathcal{ALN}$
$\top$	$\Delta^{\mathcal{I}}$	$\times$	$\times$
$\bot$	$\emptyset$		$\times$
P	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$\times$	$\times$
$\neg A$	$\Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}$		$\times$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	$\times$	$\times$
$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y : (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$	$\times$	$\times$
$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert\{y \mid (x,y) \in R^{\mathcal{I}}\}\rvert \leq n\}$		$\times$
$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid \lvert\{y \mid (x,y) \in R^{\mathcal{I}}\}\rvert \geq n\}$		$\times$

TAB. 2.2 – Sémantique des logiques $\mathcal{FL}_0$ et $\mathcal{ALN}$

Etant donné la sémantique introduite précédemment, les notions de satisfiablilité, d'équivalence et de subsomption entre deux concepts sont définis comme suit :

Définition 2.2.3 (Satisfiabilité, subsomption et équivalence)
- *Un concept C est satisfiable, noté $C \not\equiv \bot$, ssi il existe une interprétation $\mathcal{I}$ telle que $C^{\mathcal{I}} \neq \emptyset$. On dit alors que $\mathcal{I}$ est une interprétation valide ou un modèle pour C. Un concept C est dit inconsistant, noté $C \equiv \bot$, ssi il n'admet pas de modèle.*
- *Un concept C est subsumé par un concept D, noté $C \sqsubseteq D$, ssi $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}\ \forall \mathcal{I}$.*
- *Un concept C est équivalent à un concept D, noté $C \equiv D$, ssi $C^{\mathcal{I}} = D^{\mathcal{I}}\ \forall \mathcal{I}$.*

Notons qu'il existe d'autres sémantiques que la sémantique déclarative, pour les logiques de description. Un autre exemple de sémantique est celle des points fixes qui est utile en présence de concepts cycliques [52].

2.2.3 Terminologie

Les systèmes terminologiques permettent de représenter les descriptions intentionnelles d'un domaine d'application à l'aide d'une *terminologie*, aussi appelée $TBox$. Il existe plusieurs types de terminologies qui diffèrent par le type d'*axiomes* terminologiques qu'elles admettent.

Soit une logique $\mathcal{L}$ telle que $\mathcal{L}$ désigne $\mathcal{FL}_0$ ou $\mathcal{ALN}$, un axiome terminologique (ou simplement axiome) dans $\mathcal{L}$ a une des formes suivantes :

(1) $A \sqsubseteq D$ Spécification primitive de concept
(2) $A \equiv D$ Définition de concept

où A est un nom de concept et D est un concept dans $\mathcal{L}$.

Les axiomes de type (1) permettent des spécifications incomplètes dans le sens où ils donnent les conditions nécessaires pour que des individus soient dans l'extension du concept A, tandis que les axiomes de type (2) permettent des spécifications complètes dans le sens où ils donnent les conditions nécessaires et suffisantes pour que des individus soient dans l'extension de A. De manière plus précise, la sémantique des axiomes terminologiques est définie comme suit :

Définition 2.2.4 (Sémantique des axiomes terminologiques)
- *Une interprétation $\mathcal{I}$ satisfait une spécification primitive $A \sqsubseteq D$ ssi $A^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.*
- *Une interprétation $\mathcal{I}$ satisfait une définition $A \equiv D$ si et seulement si $A^{\mathcal{I}} = D^{\mathcal{I}}$.*

Une terminologie formée de spécifications primitives est appelée terminologie *primitive* tandis qu'une terminologie formée de spécifications primitives et de définitions est appelée terminologie *simple*. Dans une terminologie simple, la partie gauche d'une définition (resp. d'une spécification primitive) s'appelle un *concept défini* (resp. un *concept primitif*). Un nom de concept $A \in \mathcal{C}$ qui n'apparaît pas en partie gauche d'un axiome terminologique, que l'axiome soit une définition ou une spécification primitive, est appelé *concept atomique*.

La sémantique des terminologies peut être obtenue en étendant la notion d'interprétation comme suit.

Définition 2.2.5 (Sémantique des terminologies)
Une interprétation $\mathcal{I}$ est une interprétation valide pour une terminologie $\mathcal{T}$ si et seulement si $\mathcal{I}$ satisfait tous les axiomes de $\mathcal{T}$.

Les terminologies permettent de définir les concepts et les relations entre les concepts du domaine d'application. Autrement dit, elles spécifient le vocabulaire du domaine d'application. Un exemple de terminologie inspiré de notre domaine d'application, est donné dans la table 2.3. Cette terminologie est formée des définitions des trois concepts $Parcelletraitee$, $ParcelleBio$ et $CommuneEpandue$. La terminologie contient également la spécification primitive du concept $ParcelletraiteeAllier$. Le concept $Parcelletraitee$ dénote les parcelles culturales qui ont reçu des pesticides d'une certaine catégorie. Le concept $ParcelleBio$ spécifie les parcelles culturales qui ont une certification agriculture biologique. Le concept $CommuneEpandue$ dénote les communes qui font partie d'au moins une campagne d'épandage, tandis que le concept $ParcelletraiteeAllier$ représente un sous-ensemble des parcelles traitées dans la région $Auvergne$.

Terminologie $\mathcal{T}$
$ParcelleTraitee \equiv ParcelleCulturale \sqcap \forall arecu.(\forall CategoriePesticide.String)$
$ParcelleBio \equiv ParcelleCulturale \sqcap \geq 1\ aCertificationBio$
$CommuneEpandue \equiv Commune \sqcap \geq 1\ appartient \sqcap \forall appartient.Campagne$
$ParcelletraiteeAllier \sqsubseteq ParcelleTraitee \sqcap \forall nomRegion.Auvergne$

TAB. 2.3 – Terminologie $\mathcal{T}$

Une terminologie contient des *cycles* si un nom de concept fait référence à lui-même, directement ou indirectement dans sa spécification. Une terminologie qui ne contient pas de cycle est dite *acyclique*. Dans ce mémoire, on suppose que les terminologies sont acycliques. De même on suppose que les noms de concepts apparaissent au plus une *seule* fois en partie gauche d'une définition ou d'une spécification primitive dans une terminologie.

2.3 Raisonnement

L'intérêt des logiques de description est qu'elles permettent de faire des raisonnements comme par exemple, inférer des connaissances implicites à partir des connaissances représentées explicitement dans une terminologie. Plusieurs types de raisonnements peuvent être définis par rapport à une terminologie $\mathcal{T}$ comme le test de la *satisfiabilité* d'un concept, le test de *subsomption* entre deux concepts et le test d'*équivalence* entre deux concepts. Ces raisonnements sont sont définis comme suit :

Définition 2.3.1 (Satisfiabilité, subsomption et équivalence par rapport à une terminologie)
- *Un concept C est satisfiable par rapport à $\mathcal{T}$, noté $C \not\equiv_{\mathcal{T}} \bot$, ssi il existe un modèle $\mathcal{I}$ de $\mathcal{T}$ tel que $C^{\mathcal{I}} \neq \emptyset$.*
- *Un concept C est subsumé par un concept D par rapport à $\mathcal{T}$, noté $C \sqsubseteq_{\mathcal{T}} D$ ssi $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$.*

– *Un concept C est équivalent au concept D par rapport à $\mathcal{T}$, noté $C \equiv_{\mathcal{T}} D$ ssi $C^{\mathcal{I}} = D^{\mathcal{I}}$ pour tout modèle $\mathcal{I}$ de $\mathcal{T}$.*

Un raisonnement par rapport à une terminologie $\mathcal{T}$ peut se ramener à un raisonnement sur les descriptions concernées sans tenir compte de la terminologie, i.e. un raisonnement par rapport à une terminologie vide. Pour cela, il est nécessaire de passer par une phase de transformation de la terminologie de manière à supprimer les spécifications primitives. La terminologie obtenue après une telle transformation est appelée terminologie *régulière*. Les terminologies régulières sont ensuite *aplaties* de telle sorte que les axiomes terminologiques ne comportent que des concepts atomiques dans leur partie droite. Ces transformations sont décrites ci-après.

Soit une terminologie simple $\mathcal{T}$. Une terminologie régulière $\overline{\mathcal{T}}$ est obtenue à partir de $\mathcal{T}$ en remplaçant les spécifications primitives $A \sqsubseteq D$ en définitions $A \equiv \overline{A} \sqcap D$, où $\overline{A}$ est un nouveau concept atomique. $\overline{\mathcal{T}}$ est appelé *normalisation* de $\mathcal{T}$. Informellement, l'ajout du concept $\overline{A}$ dans l'axiome $A \equiv \overline{A} \sqcap D$ permet de spécifier les contraintes supplémentaires que doivent vérifier les individus de D pour être des individus de A.

L'aplatissement d'une terminologie $\mathcal{T}$ consiste à déployer chaque définition de $\mathcal{T}$ en son *expansion*, c'est-à-dire à remplacer de manière récursive les concepts définis par leur définition, dans la partie droite des axiomes.

Les processus de normalisation et d'aplatissement d'une terminologie $\mathcal{T}$ sont des transformations qui préservent la sémantique de $\mathcal{T}$. Autrement dit tout modèle d'une terminologie primitive $\mathcal{T}$ est un modèle pour la terminologie régularisée et aplatie $\overline{\mathcal{T}}$ de $\mathcal{T}$ et inversement. Cependant, la phase d'aplatissement d'une terminologie peut être coûteuse dans le sens où la taille de la terminologie aplatie peut être exponentielle par rapport à la taille de la terminologie initiale $\mathcal{T}$ [66]. Cette complexité reste théorique car les configurations menant à une explosion de la taille de la terminologie sont très rares en pratique [66].

Le tableau 2.4 donne la terminologie régularisée et aplatie de la terminologie donnée dans le tableau 2.3.

Terminologie régularisée et aplatie $\overline{\mathcal{T}}$
$ParcelleTraitee \equiv ParcelleCulturale \sqcap \forall arecu.(\forall CategoriePesticide.String)$ $ParcelleBio \equiv ParcelleCulturale \sqcap \geq 1 aCertificationBio$ $CommunesEpandues \equiv Communes \sqcap \geq 1 appartient \sqcap \forall appartient.Campagne$
$ParcelletraiteeAllier \equiv ParcelleCulturale \sqcap \forall arecu.(\forall CategoriePesticide.String)$ $\sqcap \forall nomRegion.Auvergne \sqcap \overline{ParcelletraiteeAllier}$

TAB. 2.4 – Normalisation et déploiement d'une terminologie $\mathcal{T}$

Soient C et D deux définitions dans $\mathcal{T}$. Soient C' et D', les expansions de C et D respectivement. Les raisonnements sur C et D dans $\mathcal{T}$ peuvent alors être réduits à des raisonnements sur C et D sans terminologie comme spécifié ci-dessous :

– C est satisfiable selon $\mathcal{T}$ ssi C' est satisfiable.
– $C \sqsubseteq_{\mathcal{T}} D$ ssi $C' \sqsubseteq D'$.
– $C \equiv_{\mathcal{T}} D$ ssi $C' \equiv D'$.

On suppose dans la suite du mémoire que toute terminologie $\mathcal{T}$ a été transformée en terminologie régulière $\overline{\mathcal{T}}$ et qu'elle est aplatie. Pour simplifier les notations, on notera par $\mathcal{T}$, une

terminologie régulière et aplatie.

2.4 Les algorithmes de subsomption

Il existe deux types d'approches pour développer des algorithmes de subsomption : l'approche structurelle et l'approches sémantique.

L'approche structurelle utilise des algorithmes qui s'appuient sur la comparaison syntaxique des concepts pour décider la subsomption entre concepts. Ces algorithmes fonctionnent en deux étapes. Premièrement les concepts sont *normalisés* afin d'obtenir des descriptions canoniques de concepts. Dans la deuxième étape, les algorithmes comparent la structure syntaxique des concepts normalisés. Ces algorithmes sont souvent efficaces bien qu'ils ne soient complets que pour quelques logiques peu expressives. L'approche structurelle est notamment utile pour décider de la subsomption entre deux concepts pour des logiques qui n'admettent pas le constructeur de négation généralisée.

L'approche sémantique utilise des algorithmes basés sur la méthode des tableaux pour tester la satisfiabilité d'un concept E [22]. Ces algorithmes fonctionnent comme suit. Ils s'appuient sur un ensemble de règles, appelées règles de propagation, dont l'objectif est de construire une interprétation valide de E. S'il est possible de générer une telle interprétation, alors le concept E est satisfiable, sinon l'algorithme garantit qu'il n'est pas possible de construire une interprétation valide pour E et donc E est inconsistant. Cette approche peut également être utilisée pour tester la subsomption entre deux concepts pour des langages fermés par la négation. En effet, tester si C est subsumé par D revient à tester si le concept $C \sqcap \neg D$ est insatisfiable, i.e. $(C \sqsubseteq D) \Leftrightarrow (C \sqcap \neg D \equiv \bot)$. Ainsi, si C est subsumé par D, un algorithme basé sur l'approche sémantique garantit qu'il n'est pas possible de construire une interprétation valide du concept $(C \sqcap \neg D)$. Ces algorithmes se sont montrés très utiles pour l'étude de la complexité des tests de satifisfiabilité et de subsomption des concepts. Ils permettent de décider de la subsomption dans des logiques bénéficiant d'une puissance d'expression élevée, qui autorisent par exemple, la négation des concepts et la disjonction.

Dans le contexte de la réécriture, l'existence des algorithmes de subsomption pour une logique donnée sont une base pour l'étude de la décidabilité du problème de la réécriture dans cette logique. Plus précisément, leur existence est une condition nécessaire (mais non suffisante) pour garantir la décidabilité du poblème de réécriture.

2.5 Raisonnement en présence des individus

Dans cette section, nous nous intéressons à l'incidence sur le raisonnement, de l'utilisation des noms d'individus dans les concepts. Dans les applications basées sur les logiques de description, il peut être intéressant de spécifier des noms d'individus dans les concepts afin d'exprimer des *types énumérés*. Cependant, les logiques $\mathcal{FL}_0$ et $\mathcal{ALN}$ ne permettent pas de représenter ces types de données. Dans le cadre des logiques de description, il est possible de décrire les types énumérés à l'aide du constructeur appelé ONEOF, noté $\mathcal{O}$, qui construit un concept à partir d'un ensemble de noms d'individus. Ainsi si on considère l'ensemble des numéros de communes $\{63456, 63677, 63788\}$ comme étant un ensemble d'individus, on peut spécifier le

domaine de l'attribut *numéro de commune*. On peut alors décrire le concept des parcelles Culturales dont le numéro de commune est uniquement 63456 ou 63677 ou 63788. La suite de cette section est dédiée à la présentation du constructeur $\mathcal{O}$ et aux difficultés à raisonner en présence de ce constructeur. Cette section s'appuie sur les articles [15, 73, 47].

2.5.1 Le constructeur $\mathcal{O}$

Pour définir la syntaxe du constructeur $\mathcal{O}$, un nouvel alphabet $\mathcal{N}$ des noms des individus, est considéré. Un ensemble d'individus est alors dénoté $\{i_1, i_2, ..., i_n\}$ où $i_j \in \mathcal{N}, \forall j \in \{1, ..., n\}$.

Le constructeur $\mathcal{O}$ s'emploie comme un concept et peut restreindre un rôle. Par exemple, $\{PuydeDome, Allier, Cantal, HauteLoire\}$ dénote le concept qui admet comme instances possibles les départements de la région Auvergne, tandis que le concept $\forall travaille.\ \{Public, Prive, Associatif\}$ dénote les individus qui s'ils travaillent, travaillent soit dans le public, soit dans le privé, ou en milieu associatif.

La fonction d'interprétation $.^{\mathcal{I}}$ est étendue aux individus de telle sorte que $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ pour tout $i \in \mathcal{N}$. De plus, l'hypothèse de nom unique est supposée vérifiée, ainsi pour tout $i, j \in \mathcal{N}$, si $i \neq j$ alors $i^{\mathcal{I}} \neq j^{\mathcal{I}}$.

La sémantique de $\{i_1, i_2, \ldots, i_n\}$ est alors définie par :

$$\{i_1, i_2, \ldots, i_n\}^{\mathcal{I}} = \{i_1^{\mathcal{I}}, i_2^{\mathcal{I}}, \ldots, i_n^{\mathcal{I}}\}$$

et celle de $\forall r.\{i_1, i_2, \ldots, i_n\}$ par :

$$\forall r.\{i_1, i_2, \ldots, i_n\}^{\mathcal{I}} = \{x \in \Delta^{\mathcal{I}} \mid (x, y) \in r^{\mathcal{I}} \Rightarrow y \in \{i_1^{\mathcal{I}}, i_2^{\mathcal{I}}, \ldots, i_n^{\mathcal{I}}\}\}.$$

2.5.2 Difficultés à raisonner en présence du $\mathcal{O}$

Il est connu que le constructeur $\mathcal{O}$ introduit des difficultés supplémentaires au niveau du raisonnement [15, 73]. Par exemple, l'utilisation du $\mathcal{O}$ induit des hypothèses implicites de cardinalité dont il faut tenir compte dans les processus de raisonnement. Nous illustrons ce cas dans l'exemple ci-dessous.

Exemple 7
Soit $C \equiv \forall travaille.\{63, 43, 03\}$.
Les instances de C, d'après sa description, ne peuvent pas avoir plus de trois images par le rôle $travaille$.
En effet, si x est une instance de C et (x, y) appartient à $travaille^{\mathcal{I}}$ alors on a forcément $y = 63$ ou $y = 43$ ou $y = 03$.
Ainsi x peut avoir au maximum trois images distinctes par le rôle $travaille$. Par conséquent, C est subsumé par le concept $\leq\ 3\ travaille$. ∘

De manière plus générale, la conjonction des concepts $\forall R.\{i_1, \ldots, i_n\}$ et $(\geq\ m\ R)$ où $m > n$, conduit à l'inconsistance. Notons que la connaissance implicite de cardinalité du $\mathcal{O}$ peut affecter la subsomption dans les logiques autorisant $\mathcal{O}$. Même dans une logique ne fournissant pas de constructeurs de cardinalité (i.e. $(\leq nR)$ et $(\geq nR)$) , comme $\mathcal{FL}_0(\mathcal{O})$, la présence du constructeur $\mathcal{O}$ conduit à des cas non triviaux de subsomption comme illustré dans

l'exemple qui suit.

Exemple 8
Considérons les concepts C et D suivants :
$C \equiv \{a\} \sqcap \forall r.\{a\} \sqcap \forall r.r.P$
$D \equiv \forall r.P$
Alors on a $C \sqsubseteq D$.
L'explication de ce cas de subsomption est la suivante.
Si C n'est pas satisfiable et $C^{\mathcal{I}} = \emptyset$ alors $C \sqsubseteq D$. Sinon C est satisfiable et $\{a\} = C^{\mathcal{I}}$. Nous allons vérifier que $\{a\}$ appartient également à l'extension de D.
Le concept D décrit tous les individus x du domaine d'interprétation tels que
– ils n'ont pas d'image par r
– ils sont restreints par r et pour tout $y \in \Delta^{\mathcal{I}}$ tel que $(x, y) \in r^{\mathcal{I}}$ alors $y \in P^{\mathcal{I}}$.
Dans l'extension de C,
– soit $\{a\}$, individu du domaine, n'est pas restreint par le rôle r. Alors $\{a\}$ appartient à l'extension de $D \equiv \forall r.P$. On a alors $C \sqsubseteq D$.
– soit $\{a\}$ est restreint par le rôle r, pour tout y du domaine d'interprétation tel que $(a, y) \in r^{\mathcal{I}}$ alors $y = a$ par la restriction de valeur de r sur $\{a\}$ dans C. Ainsi on a $(a, a) \in r^{\mathcal{I}}$ et $\{a\}$ est également dans l'extension de $\forall r.P$. Ainsi on a $C \sqsubseteq D$.
Ce cas de subsomption est du au fait que la cardinalité de l'extension de C est 1 et que $\{a\}$ a un seul successeur par le rôle r, $\{a\}$ lui même.

◦

Avec les algorithmes de subsomption structurelle qui s'attachent à comparer la structure syntaxique des concepts, il est difficile de déceler de tels cas de subsomption. Il peut être judicieux dans ce cas d'utiliser la méthode des tableaux pour raisonner en présence des individus. En effet, des algorithmes basés sur la méthode des tableaux, ont été proposés pour des logiques étendues au constructeur $\mathcal{O}$ comme, $\mathcal{ALCO}$ ([73]) et $\mathcal{SHOQ}(D)$ ([47]).

Cependant, l'approche sémantique n'est pas adaptée au traitement du problème de réécriture de requêtes en termes de vues. Avec ce type d'approche, on est obligé de parcourir tout l'espace de recherche des réécritures, cet espace étant formé de toutes les combinaisons de vues possibles. De telles approches ne sont pas envisageables dans des applications comportant beaucoup de vues. A contrario, avec l'approche syntaxique, il est possible de caractériser les critères syntaxiques que les vues doivent vérifier pour réécrire effectivement une requête donnée. Il est alors possible d'isoler dans l'espace de recherche des réécritures, un sous-ensemble de vues réellement pertinentes à la réécriture de la requête.

Dans la sous-section qui suit nous présentons la logique CLASSIC [15] qui adopte une approche différente pour traiter les individus en offrant une sémantique non standard au constructeur $\mathcal{O}$. L'intérêt de CLASSIC est qu'il existe pour cette logique un algorithme de subsomption structurelle, qui se prête mieux au problème de la réécriture comme discuté auparavant.

2.5.3 Une sémantique non standard de $\mathcal{O}$

Afin de contourner la difficulté de raisonnement en présence des individus dans les concepts, les auteurs de [15] proposent une sémantique modifiée de $\mathcal{O}$, appelée *sémantique non standard*.

Les individus dans CLASSIC sont associés à des sous-ensembles disjoints du domaine d'interprétation au lieu d'être associés à des éléments du domaine d'interprétation. Plus précisément, un individu i est interprété comme un sous-ensemble du domaine, i.e. $i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$. La modification de la sémantique de $\mathcal{O}$ s'accompagne également d'un changement de la notion de cardinalité des ensembles afin de capturer l'information implicite sur la taille de l'ensemble d'individus, imposée par $\mathcal{O}$. Comme illustré dans les exemples 7 et 8, page 39, cette information est essentielle dans des logiques permettant les contraintes de cardinalité sur les rôles. Ainsi dans CLASSIC, deux individus du domaine sont dits *congruents* s'ils appartiennent à l'extension du même individu ou s'ils sont identiques. La cardinalité d'un ensemble du domaine est alors donnée par le nombre d'éléments de l'ensemble modulo la relation de congruence. Par exemple dans CLASSIC, la subsomption entre les concepts C et D décrits dans l'exemple 7, est décelée car la cardinalité de $\{63, 43, 03\}$ modulo la relation de congruence est de 3.

Notons que dans CLASSIC, il existe un autre moyen d'insérer des noms d'individus dans les concepts quand ces noms d'individus sont vus comme des *valeurs*, ou des *types de données*, distinctes du domaine d'interprétation habituel. D'un point de vue sémantique, dans CLASSIC, le domaine d'interprétation est divisé en deux domaines : Δ_C, le domaine des individus dits "Classic", et Δ_V, le domaine des *valeurs* dites "host values". Le domaine d'interprétation d'une description de concept est un sous-ensemble de Δ_C ou de Δ_V, tandis que le domaine d'interprétation d'un rôle est $\Delta_C \times \Delta_C$ ou $\Delta_C \times \Delta_V$. Le constructeur $\mathcal{O}$, dénoté $\{i_1, i_2, ..., i_n\}$, spécifie des ensembles d'individus qui peuvent être des individus ou des valeurs. Plus formellement, l'interprétation de $\{i_1, i_2, ..., i_n\}$ est :

- $\cup_k i_k^{\mathcal{I}}$, i.e. l'union des interprétations des individus i_j, $j \in \{1, ..., n\}$ si les individus i_j, $j \in \{1, ..., n\}$ sont des individus Classic.
- $\{i_1, i_2, ..., i_n\}$ si les individus i_j, $j \in \{1, ..., n\}$ sont des valeurs.

Lorsque le constructeur $\mathcal{O}$ spécifie un ensemble de valeurs, il est uniquement utilisé afin de restreindre le domaine de valeur des rôles. Plus précisément, un ensemble de valeurs ne peut pas être utilisé pour définir un concept quelconque. Les valeurs, contrairement aux individus Classic, ne peuvent pas être restreintes par des rôles.

Notons que l'algorithme de subsomption proposé pour CLASSIC dans [15], est complet par rapport à la sémantique non standard de $\mathcal{O}$, mais est incomplet par rapport à la sémantique standard de $\mathcal{O}$. Par exemple, l'algorithme ne décèle pas le cas de subsomption entre les concepts C et D décrits dans l'exemple précédent 8.

Après avoir présenté les notions de base autour des logiques de description et les difficultés de raisonnement liées à la présence de individus dans les concepts, nous présentons maintenant nos choix de modélisation des contraintes de valeurs, à l'aide des logiques de description.

Chapitre 3

Modélisation des contraintes de valeurs

Dans ce chapitre, nous introduisons le constructeur $\mathcal{O}_v$ qui permet de modéliser les contraintes de valeurs dans le cadre des logiques de description (section 3.1). Nous présentons ensuite les logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ obtenues à partir des logiques $\mathcal{FL}_0$ et $\mathcal{ALN}$ en les étendant avec le constructeur $\mathcal{O}_v$ (section 3.2). Puis nous donnons une caractérisation de la subsomption structurelle dans ces deux logiques (sections 3.3 et 3.4). Finalement, nous discutons nos choix de modélisation des contraintes de valeurs (section 3.5).

3.1 Représentation des contraintes de valeurs avec $\mathcal{O}_v$

Dans cette section, nous nous intéressons à la modélisation des contraintes de valeurs à l'aide des logiques de description. Dans ce mémoire, une valeur désigne les individus qui ne peuvent pas être restreints par des rôles. Une contrainte de valeurs est l'ensemble des valeurs autorisées pour un attribut donné, i.e. elle est un cas particulier de type énuméré. Comme discuté dans le chapitre 2, il est possible de modéliser les types énumérés et donc les contraintes de valeurs avec le constructeur $\mathcal{O}$ (chapitre 2). Cependant pris dans sa sémantique standard, il n'existe pas d'algorithme complet de subsomption (structurelle) pour les logiques autorisant le constructeur $\mathcal{O}$. Aussi nous introduisons un constructeur noté par la suite $\mathcal{O}_v$, qui permet de représenter des ensembles de *valeurs*. Le constructeur $\mathcal{O}_v$ est bien adapté à notre contexte applicatif car les types énumérés auxquels nous nous intéressons sont uniquement des contraintes de valeurs. Ce constructeur n'est pas nouveau puisqu'il est proche de la notion des "host values", utilisée dans CLASSIC [15].

Soit $\mathcal{N}$ l'ensemble des noms de valeurs. La syntaxe de $\mathcal{O}_v$ est la suivante : $\{o_1, \ldots, o_n\}$ où $o_i \in \mathcal{N}$. Nous détaillons à présent les restrictions syntaxiques qui portent sur l'utilisation du constructeur $\mathcal{O}_v$ ainsi que sur la sémantique de $\mathcal{O}_v$.

Nous supposons sans perte de généralité, que l'ensemble des noms de rôles est séparé en deux ensembles disjoints $\mathcal{R}_C$ et $\mathcal{R}_V$. Ceci permet de distinguer les rôles classiques, appartenant à $\mathcal{R}_C$, et les rôles appartenant à $\mathcal{R}_V$ qui prennent leurs valeurs dans $\mathcal{N}$ [15]. Le constructeur $\mathcal{O}_v$ est donc réservé à la restriction de rôles appartenant à $\mathcal{R}_V$.

[15] Une distinction similaire a été proposée dans [47] pour distinguer dans $\mathcal{SHOQ}$(D) les noms de rôles dont le domaine de valeurs est celui des domaines concrets et ceux dont le domaine de valeurs est celui des concepts.

L'exemple qui suit illustre la notion de contrainte de valeurs portant sur un attribut numéro de commune décrite à l'aide des logiques de description.

Exemple 9
Soient $numCommune$ un nom de rôle appartenant à $\mathcal{R}_V$, et $\{63450, 63250, 43555\}$ un ensemble de numéros de commune.
Le concept $\forall numCommune.\{63450, 63250, 43555\}$ dénote l'ensemble des individus du domaine qui n'ont pas de numéro de commune, ou bien qui ont un numéro de commune égal à 63450 ou 63250 ou 43555. Le constructeur $\mathcal{O}_v$ a permis d'exprimer la contraintes de valeurs portant sur le rôle $NumCommune$.

○

Pour définir la sémantique du constructeur $\mathcal{O}_v$, nous considérons que le domaine d'interprétation $\Delta^{\mathcal{I}}$ est partionné en deux ensembles disjoints : δ_C décrivant l'ensemble des individus du domaine et δ_V décrivant l'ensemble des valeurs. Un concept est interprété comme étant un sous-ensemble de $\Delta^{\mathcal{I}}$. Un rôle est interprété comme un sous-ensemble de $\delta_C \times \Delta^{\mathcal{I}}$. Autrement dit les individus de δ_V ne peuvent pas être restreints par des rôles. Ainsi le constructeur $\mathcal{O}_v$ est une restriction du constructeur $\mathcal{O}$ dans le sens où $\mathcal{O}_v$ ne permet pas la création directe de concepts à partir d'ensembles d'individus de δ_C. Le constructeur $\mathcal{O}_v$ permet uniquement de restreindre le domaine de valeurs des rôles.

Soit $\mathcal{N}$, l'ensemble des noms de valeurs. L'interprétation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ associe également chaque valeur $I_i \in \mathcal{N}$ à un élément $I_i^{\mathcal{I}} \in \delta_V$ tel que $I_i \neq I_j$ implique que $I_i^{\mathcal{I}} \neq I_j^{\mathcal{I}}$ (i.e. l'interprétation respecte l'hypothèse de l'unicité des noms).

Nous focalisons à présent sur les deux logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ obtenues respectivement, en étendant les logiques $\mathcal{FL}_0$ et $\mathcal{ALN}$ avec le constructeur $\mathcal{O}_v$.

3.2 $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$

Les logiques de description $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ peuvent être vues comme des sous-langages de CLASSIC [54] dans le sens où elles utilisent un sous-ensemble des constructeurs de CLASSIC. Nous donnons ci-dessous une nouvelle caractérisation de la subsomption dans ces deux logiques basée sur une forme normale appropriée. Cette caractérisation fournit les conditions nécessaires, exprimées sous forme de contraintes syntaxiques, qui doivent être vérifiées par les réécritures.

La logique $\mathcal{FL}_0(\mathcal{O}_v)$

Les constructeurs de $\mathcal{FL}_0(\mathcal{O}_v)$ sont donnés dans le tableau 3.1. Cette logique contient en plus des constructeurs permis par $\mathcal{FL}_0$ les constructeurs suivants :

- $\top_C$ et $\top_V$ qui représentent respectivement l'ensemble de tous les individus de δ_C et l'ensemble de tous les individus de δ_V. L'union de $\top_C$ et $\top_V$ est le concept $\top$
- la restriction de valeurs $\forall R_v.\{o_1, ..., o_n\}$ qui spécifie les individus du domaine dont les images par R_V sont prises dans l'ensemble $\{o_1, \ldots, o_n\}$.

Notons que si le quantificateur universel est restreint par un rôle de $\mathcal{R}_C$, il a la sémantique usuelle introduite en section 2.

Les descriptions dans $\mathcal{FL}_0(\mathcal{O}_v)$ sont interprétées selon la sémantique des constructeurs donnée dans la table 3.2, page 46.

L'exemple ci-dessous illustre des concepts dans $\mathcal{FL}_0(\mathcal{O}_v)$, caractéristiques du domaine du développement durable.

Exemple 10

Le concept $ParcelleCulturale \sqcap \forall NumCommune.\{63200, 63250, 63450\}$ spécifie les parcelles culturales dont le numéro de commune quand il existe, est 63200 ou 63250 ou 63450.
Le concept $Epandage \sqcap \forall effectuePar.Exploitation \sqcap$
$\forall aPerimetreEpandage.dansCommune.\{63200, 63250, 63450, 03200, 03400\}$ spécifie les épandages qui ont été réalisés par une exploitation sur une zone d'épandage située sur les communes 63200 ou 63250 ou 63450 ou 03200 ou 03400. ◦

Noms des Constructeurs	Constructeurs	$\mathcal{FL}_0(\mathcal{O}_v)$	$\mathcal{ALN}(\mathcal{O}_v)$
Concept Top$_C$	$\top_C$	×	×
Concept Top$_V$	$\top_V$	×	×
Concept Top	$\top$	×	×
Concept Bottom	$\bot$		×
Négation atomique	$\neg A$		×
Conjonction	$C \sqcap D$	×	×
Restriction Universelle	$\forall R_c.C$	×	×
Restriction de valeurs	$\forall R_v.\{o_1, ..., o_n\}$	×	×
Restriction de cardinalité inférieure	$\leq nR_c$		×
	$\leq nR_v$		×
Restriction de cardinalité supérieure	$\geq nR_c$		×
	$\geq nR_v$		×

TAB. 3.1 – Syntaxe des constructeurs définissant $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$

Nous nous intéressons à présent à la logique $\mathcal{ALN}(\mathcal{O}_v)$.

La logique $\mathcal{ALN}(\mathcal{O}_v)$

La logique $\mathcal{ALN}(\mathcal{O}_v)$ contient en plus des constructeurs de $\mathcal{ALN}$ et de $\mathcal{FL}_0(\mathcal{O}_v)$, les constructeurs de cardinalité suivants :

– $\leq nR_v$
– $\geq nR_v$

Ces constructeurs ne sont que la généralisation des constructeurs de cardinalité usuels aux rôles de $\mathcal{R}_V$. L'ensemble des constructeurs caractérisant $\mathcal{ALN}(\mathcal{O}_v)$ est donné dans la table 3.1, colonne 4. Leur sémantique est donnée dans la table 3.2.

Des exemples de concepts dans $\mathcal{ALN}(\mathcal{O}_v)$ sont donnés ci-dessous.

Constructeurs	*Sémantique*
$\top_C$	δ_C
$\top_V$	δ_V
$\top$	$\Delta^{\mathcal{I}} = \delta_C \cup \delta_V$
$\bot$	$\emptyset$
P	$P^{\mathcal{I}}$
$\neg A$	$\Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}$
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
$\forall R_c.C$	$\{x \in \delta_C \mid \forall y : (x,y) \in {R_c}^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
$\forall R_v.\{o_1, \ldots, o_n\}$	$\{x \in \delta_C \mid \forall y : (x,y) \in {R_v}^{\mathcal{I}} \rightarrow y \in \{o_1^{\mathcal{I}}, \ldots, o_n^{\mathcal{I}}\} \subseteq \delta_V\}$
$\leq n R_c$	$\{x \in \delta_C \mid \lvert\{y \mid (x,y) \in {R_c}^{\mathcal{I}}\}\rvert \leq n\}$
$\leq n R_v$	$\{x \in \delta_C \mid \lvert\{y \mid (x,y) \in {R_v}^{\mathcal{I}}\}\rvert \leq n\}$
$\geq n R_c$	$\{x \in \delta_C \mid \lvert\{y \mid (x,y) \in {R_c}^{\mathcal{I}}\}\rvert \geq n\}$
$\geq n R_v$	$\{x \in \delta_C \mid \lvert\{y \mid (x,y) \in {R_v}^{\mathcal{I}}\}\rvert \geq n\}$

TAB. 3.2 – Sémantique de la logique $\mathcal{ALN}(\mathcal{O}_v)$

Exemple 11

- $ParcelleCulturale \sqcap \forall aRecuP.CategoriedePesticide.\{C_1, C_2, C_3, C_{10}, C_{12}, C_{14}, C_{15}\}$ spécifie les parcelles culturales qui ont reçu des pesticides de catégorie C_1 ou C_2 ou C_3 ou C_{10} ou C_{12}ou C_{14} ou C_{15}.
- $ParcelleCulturale \sqcap \forall aRecuE.\neg EffluentElevage \sqcap \geq 1\, aRecuE$ spécifie les parcelles culturales qui ont reçu au moins un épandage qui n'était pas un effluent d'élevage.

○

Les sections qui suivent, donnent successivement une caractérisation de la subsomption dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$ puis dans $\mathcal{ALN}(\mathcal{O}_v)$.

3.3 Caractérisation de la subsomption dans $\mathcal{FL}_0(\mathcal{O}_v)$

Afin de caractériser la subsomption dans $\mathcal{FL}_0(\mathcal{O}_v)$, nous présentons d'abord une forme normale pour décrire les concepts dans $\mathcal{FL}_0(\mathcal{O}_v)$ puis les règles qui permettent de transformer les concepts dans leur forme normale. La forme normale $\widehat{C}$ d'une description C permet de rendre explicite la connaissance implicite de C et de supprimer les éventuelles redondances apparaissant dans C. Elle fournit également une représentation canonique des concepts.

Dans la suite, nous utilisons les lettres $A, B, A_1, \ldots$ pour dénoter des concepts atomiques et $E, E', E_1, \ldots$ pour dénoter des ensembles de valeurs. E peut être l'ensemble vide. P et P' spécifient soit un concept atomique (A) ou bien un ensemble de valeurs (E). D et D' spécifient des concepts quelconques.

La forme normale $\widehat{C}$ d'un concept C est soit le concept $\top$, soit une conjonction (non vide) de descriptions de la forme $\forall R_1.(\ldots \forall R_m.P)$ avec $m \geq 0$, où $R_1, \ldots, R_m$ sont des rôles (non nécessairement distincts).

La description $\forall R_1 \dots R_m.P$ est un raccourci pour exprimer le concept $\forall R_1.(\dots \forall R_m.P)$. $R_1 \dots R_m$ est considéré comme un mot, noté w, sur l'alphabet $\mathcal{R}_\mathcal{C} \cup \mathcal{R}_\mathcal{V}$ des rôles. On dira alors que $w \in \mathcal{R}^*$. Plus précisément, $R_1 \dots R_{m-1}$ est un mot dans $\mathcal{R}_c^*$ et R_m appartient à $\mathcal{R}_c \cup \mathcal{R}_v$. Plus simplement on dira que $w \in \mathcal{R}^*$. Si $m = 0$, alors on a un mot vide ϵ. Autrement dit $\forall \epsilon.P$ est une écriture équivalente de P. Si w et u sont deux mots de $\mathcal{R}^*$, wu dénote le mot obtenu par la concaténation de w et u. Par conséquent tout concept dans $\mathcal{FL}_0(\mathcal{O}_v)$ peut être exprimé dans sa forme normale comme une conjonction de concepts de la forme $\forall w.P$, appelés par la suite *atome*.

La forme normale d'un concept dans $\mathcal{FL}_0(\mathcal{O}_v)$ se décline à partir des règles données dans la table 3.3. La règle (1) est appliquée récursivement jusqu'à saturation. Ensuite la règle (2) est également appliquée jusqu'à saturation. La règle (1) peut être vue comme une règle de distribution par rapport au mot w tandis que la règle (2) permet de rendre explicite la connaissance cachée par les contraintes de valeurs.

(1)	$\forall w.(D \sqcap D') \rightarrow \forall w.D \sqcap \forall w.D'$
(2)	$\forall w.E_1 \sqcap \forall w.E_2 \rightarrow \forall w.E$ tel que $E = E_1 \cap E_2$

TAB. 3.3 – Règles de normalisation dans $\mathcal{FL}_0(\mathcal{O}_v)$

Pour spécifier qu'un atome $\forall w.P$ est dans la description de C, on écrira par la suite $\forall w.P \in C$.

L'exemple suivant illustre la normalisation d'un concept dans $\mathcal{FL}_0(\mathcal{O}_v)$.

Exemple 12

Considérons le concept C suivant :

$C \equiv ParcelleCulturale \sqcap \forall appartient.(Exploitation \sqcap$
$\forall numCommune.\{63250, 63003, 03028, 03250\})$
$\sqcap \forall appartient.numCommune.\{63250, 63003, 43008\}$
$\sqcap \forall sousCertification.AgriRaisonnee.$

Par application de la première règle de normalisation, on obtient :

$C \equiv ParcelleCulturale \sqcap \forall appartient.Exploitation$
$\sqcap \forall appartient.numCommune.\{63250, 63003, 03028, 03250\})$
$\sqcap \forall appartient.numCommune.\{63250, 63003, 43008\}$
$\sqcap \forall sousCertification.AgriRaisonnee.$

Par application de la deuxième règle de normalisation, on obtient la forme normale $\widehat{C}$ de C qui est :

$\widehat{C} \equiv ParcelleCulturale \sqcap \forall appartient.Exploitation$
$\sqcap \forall appartient.numCommune.\{63250, 63003\}$
$\sqcap \forall sousCertification.AgriRaisonnee.$

Les atomes de $\widehat{C}$ sont :

– $ParcelleCulturale$,
– $\forall appartient.Exploitation$,
– $\forall appartient.numCommune.\{63250, 63003\}$,
– $\forall sousCertification.AgriRaisonnee$

○

Pour caractériser la subsomption dans $\mathcal{FL}_0(\mathcal{O}_v)$, nous empruntons les notations introduites dans [9] qui permettent de spécifier et de catégoriser les contraintes présentes dans un concept. Ces contraintes sont exprimées sous forme d'ensembles de mots, notés $V_C(A)$ et $V_C(E)$. L'ensemble $V_C(A)$ permet d'expliciter tous les mots d'un concept C donné, qui sont contraints par le concept A. L'ensemble $V_C(E)$ permet d'expliciter tous les mots d'un concept C qui sont contraints par l'ensemble de valeurs E. Plus formellement, on obtient :

Définition 3.3.1
Soit un concept C dans $\mathcal{FL}_0(\mathcal{O}_v)$.
- $V_C(A) = \{w | \forall w.A \in C\}$
- $V_C(E) = \{w | \forall w.E' \in C$ *and* $E' \subseteq E\}$

Les ensembles $V_C(A)$ et $V_C(E)$ d'un concept C sont illustrés ci-dessous.

Exemple 13
Soient $C \equiv \forall u.A \sqcap \forall uv.\{a,b\} \sqcap \forall uwv.\{a,b\}$ et les ensembles de rôles $\mathcal{R}_C = \{u,w\}$ et $\mathcal{R}_V = \{v\}$.
Alors
$V_C(A) = \{u\}$,
$V_C(\{a,b\}) = \{uv, uwv\}$,

○

Le théorème 3.3.1 donné ci-dessous, permet de caractériser la relation de subsomption entre deux concepts dans $\mathcal{FL}_0(\mathcal{O}_v)$ exprimés dans la forme normale introduite précédemment. La preuve de ce théorème est donnée en annexe A.1, page 117.

Théorème 3.3.1 (Subsomption)
Soient C, D deux descriptions de concepts dans leur forme normale.
$C \sqsubseteq D$ si et seulement si une des conditions suivantes est vérifiées :
1) $D \equiv \top$ ou bien
2) $w \in V_C(P)$ pour tout $\forall w.P \in D$.

La section qui suit s'intéresse à une caractérisation de la subsomption dans $\mathcal{ALN}(\mathcal{O}_v)$.

3.4 Caractérisation de la subsomption dans $\mathcal{ALN}(\mathcal{O}_v)$

Comme pour $\mathcal{FL}_0(\mathcal{O}_v)$, afin de caractériser la subsomption dans $\mathcal{ALN}(\mathcal{O}_v)$, nous présentons une forme normale pour décrire les concepts dans $\mathcal{ALN}(\mathcal{O}_v)$ puis les règles qui permettent de transformer les concepts dans leur forme normale.

Dans la suite, les lettres $A, B, A_1, \ldots$ désignent des concepts atomiques et $E, E', E_1, \ldots$ dénotent des ensembles de valeurs. E peut être l'ensemble vide. P et P' spécifie soit un concept atomique (A) ou sa négation ($\neg A$) ou bien un ensemble de valeurs (E) ou encore une restriction de cardinalité (($\leq nR$) ou ($\geq nR$)), ou le concept $\bot$. D et D' spécifient des concepts quelconques.

Tout concept C dans $\mathcal{ALN}(\mathcal{O}_v)$ est transformé en un concept équivalent, noté $\widehat{C}$, donné dans sa forme normale, tel que $\widehat{C}$ est soit le concept $\top$, soit une conjonction (non vide) de concepts de la forme $\forall R_1 \ldots R_m.P$ avec $m \geq 0$, où $R_1, \ldots, R_m$ sont des rôles (non nécessairement distincts). On peut remarquer qu'un concept inconsistant dans $\mathcal{ALN}(\mathcal{O}_v)$ doit posséder un atome de la forme : $\forall \epsilon.\emptyset$.

Pour simplifier l'ensemble des règles permettant de normaliser une description $\mathcal{ALN}(\mathcal{O}_v)$, nous supposons que les descriptions de la forme $(\geq 0R)$, $\forall R_C.\top_C$ ou $\forall R_V.\top_V$ sont transformées en $\top_C$, tandis que les descriptions de la forme $P \sqcap \top_C$ sont transformées en P. La forme normale $\widehat{C}$ d'un concept C dans $\mathcal{ALN}(\mathcal{O}_v)$ est obtenue en appliquant tout d'abord les règles (1) et (2) données dans la table 3.4, sur C, récursivement jusqu'à saturation. Puis la règle (3) est appliquée une fois.

(1)	$\forall w.D \sqcap \forall w.D' \rightarrow \forall w.(D \sqcap D')$
(2)	$E_1 \sqcap E_2 \rightarrow E$ tel que $E = E_1 \cap E_2$
(3)	$\forall R_v.E \rightarrow \forall R_v.E \sqcap (\leq kR_v)$ où $\lvert E\rvert = k$
(4)	$\leq 0R \sqcap \forall R.D \rightarrow \leq 0R$
(5)	$A \sqcap \neg A \rightarrow \bot$
(6)	$(\geq nR) \sqcap (\leq mR) \rightarrow \bot$ si $n > m$
(7)	$(\geq nR) \sqcap (\geq mR) \rightarrow (\geq max(n,m)R)$
(8)	$(\leq nR) \sqcap (\leq mR) \rightarrow (\leq min(n,m)R)$
(9)	$P \sqcap \bot \rightarrow \bot$
(10)	$\forall R.\bot \rightarrow \leq 0R$
(11)	$\forall w.(D \sqcap D') \rightarrow \forall w.D \sqcap \forall w.D'$

TAB. 3.4 – Règles de normalisation dans $\mathcal{ALN}(\mathcal{O}_v)$

Ensuite, les règles (4) à (10) sont appliquées récursivement jusqu'à saturation. Notons que les règles (7) et (8) s'appuient sur deux fonctions $min(n,m)$ et $max(n,m)$ qui prennent en entrée deux entiers n et m. $min(n,m)$ retourne n si $n \leq m$, et m dans le cas contraire. $max(n,m)$ retourne m si $n \leq m$, et n dans le cas contraire. Finalement, la règle (11) est récursivement appliquée au concept obtenu. Les règles (4),(5),(6),(9) permettent d'expliciter les inconsistances tandis que la règle (2) permet d'expliciter les connaissances implicites sur les contraintes de valeurs. Les règles (3),(7),(8) permettent d'expliciter les connaissances implicites sur les contraintes de cardinalité.

L'exemple suivant décrit le processus de normalisation d'un concept dans $\mathcal{ALN}(\mathcal{O}_v)$.

Exemple 14

Soit une description de concept C tel que
$C \equiv \forall R_1.(\forall S.E_1 \sqcap \forall R_2.(\neg A \sqcap Q) \sqcap \forall R_2.A) \sqcap \forall R_1.\forall S.E_2$
Alors par application de la règle 1, on a
$C \equiv \forall R_1.(\forall S.E_1 \sqcap \forall R_2.(\neg A \sqcap Q \sqcap A)) \sqcap \forall R_1.\forall S.E_2$
Puis
$C \equiv \forall R_1.(\forall S.(E_1 \sqcap E_2) \sqcap \forall R_2.(\neg A \sqcap Q \sqcap A))$.
Par application de la règle 2, on obtient
$C \equiv \forall R_1.(\forall S.E \sqcap \forall R_2.(\neg A \sqcap Q \sqcap A))$ où $E = E_1 \cap E_2$.

Par application de la règle 3, on obtient
$C \equiv \forall R_1.(\forall S.E \sqcap \leq k\ S\ \sqcap \forall R_2.(\neg A \sqcap Q \sqcap A))$ où $|E| = k$.
Par application des règles 5 et 9, on obtient
$C \equiv \forall R_1.(\forall S.E \sqcap \leq k\ S\ \sqcap \forall R_2.(\bot))$.
Par application de la règle 10, on obtient
$C \equiv \forall R_1.(\forall S.E \sqcap \leq k\ S \sqcap \leq 0\ R_2)$.
Enfin par la règle 11, on obtient la forme normale $\widehat{C}$ du concept C :
$\widehat{C} \equiv \forall R_1.S.E \sqcap \forall R_1. \leq k\ S\ \sqcap \forall R_1. \leq 0\ R_2$.

○

Par conséquent toute description dans $\mathcal{ALN}(\mathcal{O}_v)$ peut être exprimée dans sa forme normale comme une conjonction d'atomes $\forall w.P$. On supposera dans la suite du mémoire que tout concept C est donné dans sa forme normale.

Afin de permettre la caractérisation de la subsomption basée sur la forme normale introduite juste au-dessus, nous complétons les notations introduites dans la définition 3.3.1, en section 3.2 avec celles données dans [9], i.e. $V_C(P)$ et $E(C)$. Les ensembles $V_C(P)$, où $P \in \{A, \neg A, (\geq nR), (\leq nR), E\}$ permettent d'expliciter tous les mots sur lesquels il existe des contraintes (de type $\forall w.P$ pour $V_C(P)$) imposées par le concept C, tandis que les ensembles $E(C)$ permettent d'expliciter les mots sur lesquels il existe des contraintes $\forall v.(\leq 0R)$. Ces ensembles sont explicités dans la définition 3.4.1 qui suit.

Définition 3.4.1
Soit un concept C dans $\mathcal{ALN}(\mathcal{O}_v)$.
- $V_C(A)$ = $\{w | \forall w.A \in C\}$
- $V_C(\neg A)$ = $\{w | \forall w.\neg A \in C\}$
- $V_C((\geq nR))$ = $\{w | \forall w.(\geq mR) \in C \wedge m \geq n\}$
- $V_C((\leq nR))$ = $\{w | \forall w.(\leq mR) \in C \wedge m \leq n\}$
- $V_C(E)$ = $\{w | \forall w.E' \in C$ *and* $E' \subseteq E\}$
- $E(C)$ = $\{w |$ *il existe un préfixe* $w' = vR$ *de* w *tel que* $\forall v.(\leq 0R) \in C\}$

Exemple 15
Soient $C \equiv \forall u.A \sqcap \forall uv.\{a, b\} \sqcap \forall w(\leq 0\ u) \sqcap \forall uwv.\{a, b\}$ et les ensembles de rôles $\mathcal{R}_C = \{u, v, w\}$ et $\mathcal{R}_V = \{v\}$.
Alors
$V_C(A) = \{u\}$,
$V_C(\{a, b\}) = \{uv, uwv\}$,
$\mathrm{E}(C) = \{wu(u|v|w|\epsilon)^*\} = \{wu, wuv, wuu, wuw, wu...\}$

○

Le théorème 3.3.1 suivant donne une caractérisation de la subsomption entre deux concepts dans $\mathcal{ALN}(\mathcal{O}_v)$. Il s'appuie sur la forme normale et les ensembles de mots définis ci-dessus.

Théorème 3.4.1 (Subsomption)
Soient C, D deux concepts, exprimés dans leur forme normale.
$C \sqsubseteq D$ si et seulement si l'une des conditions suivantes est vérifiée :
(1) $C \equiv \bot$ ou $D \equiv \top_c$, ou

(2) $w \in V_C(P) \cup E(C)$ *pour chaque* $\forall w.P$ *dans* D.

La preuve de ce théorème, dérivée de la caractérisation de la subsomption structurelle de CLASSIC [54], est donnée en annexe A.1, page 117.

Dans la section qui suit, nous discutons des alternatives possibles de la modélisation des contraintes de valeurs, i.e. :
- dans le cadre des logiques de description
- en considérant ces contraintes en dehors de la description des concepts.

La discussion autour de ces alternatives permet de motiver nos travaux portant sur la réécriture de requêtes en présence de contraintes de valeurs.

3.5 Discussion

Nous commençons par discuter de la possibilité de simuler le constructeur $\mathcal{O}_v$ avec des logiques ne permettant pas $\mathcal{O}_v$.

3.5.1 Simuler $\mathcal{O}_v$ avec d'autres logiques

Une alternative à la modélisation des valeurs avec $\mathcal{O}_v$ serait de simuler chaque valeur avec nouveau concept à intégrer dans la terminologie. Cependant pour capturer la sémantique du $\mathcal{O}_v$, il est nécessaire de disposer des logiques qui permettent d'exprimer la disjonction (constructeur $\sqcup$) et de capturer l'hypothèse de l'unicité des noms, i.e. de spécifier que les concepts nouvellement ajoutés sont disjoints deux à deux. Il n'est donc pas possible de simuler le constructeur $\mathcal{O}_v$ avec les logiques $\mathcal{FL}_0$ et $\mathcal{ALN}$ introduites précédemment.

Cependant, nous nous sommes intéressés aux alternatives de représentation de $\mathcal{O}_v$ avec les logiques de description, afin d'identifier et de positionner nos travaux par rapport aux résultats déjà obtenus dans le contexte de la réécriture de requêtes. Nous illustrons ci-dessous deux possibilités de simulation du constructeur $\mathcal{O}_v$: *i)* la première avec des langages autorisant les constructeurs de négation généralisée ($\neg C$) et de disjonction ($\sqcup$), *ii)* la deuxième avec des langages autorisant les constructeurs de restriction cardinalité sur les rôles ($\leq nR$, $\geq nR$) et de disjonction. Dans l'exemple 16, nous donnons une première approche pour simuler $\mathcal{O}_v$ à l'aide des constructeurs de négation généralisée et de disjonction.

Exemple 16
Soit $\{o_1, o_2, o_3, o_4\}$ l'ensemble des valeurs dans δ_V. Pour chaque valeur $o_i \in \delta_V$, un nouveau concept atomique O_i est créé. On introduit également dans la terminologie des axiomes spécifiant l'hypothèse de l'unicité des noms portant sur les individus, comme suit :
$O_2 \sqsubseteq \neg O_1$
$O_3 \sqsubseteq \neg O_1$
$O_3 \sqsubseteq \neg O_2$
$O_4 \sqsubseteq \neg O_1$
$O_4 \sqsubseteq \neg O_2$
$O_4 \sqsubseteq \neg O_3$.

Ainsi les concepts O_i, O_j avec $i \neq j$ sont disjoints deux à deux. Dans une terminologie régulière, ils peuvent être décrits comme suit :
$O_2 \equiv \overline{O_2} \sqcap \neg O_1$
$O_3 \equiv \overline{O_3} \sqcap \neg O_2 \sqcap \neg O_1$
$O_4 \equiv \overline{O_4} \sqcap \neg O_3 \sqcap \neg O_2 \sqcap \neg O_1$
Ensuite, étant donnée cette modélisation, tous les ensembles de valeurs $\{o_{i_1}, ..., o_{i_n}\}$, apparaissant dans les concepts sont remplacés par des disjonctions $O_{i_1} \sqcup ... \sqcup O_{i_n}$. Par exemple, le concept $\forall r.\{o_1, o_2, o_3\}$ est représenté par $\forall r.(O_1 \sqcup O_2 \sqcup O_3)$.

○

Il existe une seconde approche pour simuler $\mathcal{O}_v$ à l'aide des constructeurs des restrictions de cardinalité. Cette approche est donnée dans l'exemple 17.

Exemple 17
Soit $\{o_1, o_2, o_3, o_4\}$ l'ensemble des valeurs dans δ_V.
Comme dans l'exemple ci-dessus, chaque individu o_i conduit à un nouveau concept O_i dans la terminologie.
Pour capturer l'hypothèse de l'unicité des noms des valeurs, on introduit un nouveau rôle, noté R_O qui est utilisé pour décrire les nouveaux concepts comme suit.
$O_1 \equiv \leq\ 1\ R_O \sqcap \geq\ 1\ R_O$
$O_2 \equiv \leq\ 2\ R_O \sqcap \geq\ 2\ R_O$
$O_3 \equiv \leq\ 3\ R_O \sqcap \geq\ 3\ R_O$
$O_4 \equiv \leq\ 4\ R_O \sqcap \geq\ 4\ R_O$
On peut vérifier que les concepts représentant les individus sont disjoints deux à deux, i.e. $O_i \sqcap O_j \equiv \bot$ pour tout i, j tel que $i \neq j$. Par exemple, $O_1 \sqcap O_2 \equiv \leq\ 1\ R_O \sqcap \geq\ 1\ R_O \sqcap \leq\ 2\ R_O \sqcap \geq\ 2\ R_O$ et par conséquent, on a $O_1 \sqcap O_2 \equiv \bot$.
Pour modéliser l'ensemble de valeurs $\{o_1, o_2, o_3, o_4\}$, 4 axiomes ont été ajoutés dans la terminologie. Il faut donc autant d'axiomes que de valeurs. ○

Cependant ces deux approches de simulation du $\mathcal{O}_v$ posent deux problèmes. Premièrement, le nombre de valeurs dans δ_V pouvant être très grand, cette modélisation du $\mathcal{O}_v$ augmente considérablement la taille de la terminologie. Dans notre contexte, où le nombre de communes en France est de 30000 environ, une telle modélisation des valeurs implique la création d'un nombre important de concepts. De plus, cette modélisation peut s'avérer coûteuse lors de l'aplatissement de la terminologie.

Cette modélisation pose un deuxième problème sérieux. Elle ne capture pas totalement la sémantique de $\mathcal{O}_v$. En effet l'information sur la contrainte de cardinalité inhérente au constructeur $\mathcal{O}_v$ n'est pas considérée comme le montre l'exemple ci-dessous.

Exemple 18
Considérons le concept $C \equiv \forall R.\{o_1, o_2, o_3\}$ et sa simulation $C' \equiv \forall R.(O_1 \sqcup O_2 \sqcup O_3)$ où les concepts O_i sont définis à l'aide de l'une des approches précédentes.
Comme expliqué dans le chapitre 2, section 2.5, le concept C est subsumé par $(\leq 3R)$ alors que

cette relation n'est pas vérifiée pour C' et $(\leq 3R)$. ◦

Dans le cadre des logiques ne permettant pas la restriction de cardinalité sur les rôles, cette modélisation est sans conséquence. En revanche dans le cadre de logiques permettant les restrictions de cardinalité sur les rôles, les algorithmes de subsomption sont incomplets car ils manquent des relations de subsomption entre les restrictions de rôles sur des valeurs et les contraintes de cardinalité sur les rôles.

Conséquences sur les approches de réécritures existantes

Nous nous intéressons maintenant aux travaux portant sur la réécriture dans les logiques pouvant capturer, même partiellement, la sémantique de $\mathcal{O}_v$, comme par exemple $\mathcal{ALC}$ et $\mathcal{ALCNR}$.

Dans le cadre des logiques autorisant les constructeurs de restrictions de cardinalités sur les rôles, comme les algorithmes de subsomption sont incomplets vis à vis de la sémantique de $\mathcal{O}_v$, les algorithmes de réécriture sont par conséquent incomplets. En revanche, des travaux portant sur la réécriture dans des logiques ne permettant pas les constructeurs de restrictions de cardinalités, comme $\mathcal{ALC}$, peuvent être réutiliser pour calculer les réécritures dans $\mathcal{FL}_0(\mathcal{O}_v)$ en modélisant le problème avec l'approche donnée dans l'exemple 16. Cependant les travaux connus pour cette logique, présentés dans [8], concernent le problème du calcul des réécritures équivalentes partielles et ne peuvent pas être aisément adaptés au problème du calcul des réécritures maximales. Dans [11], le problème de calculer les réécritures maximales d'une requête a été exploré pour la logique $\mathcal{ALCNR}$. Cependant, seule la décidabilité du problème a été prouvée. De plus, comme souligné plus haut, les algorithmes de subsomption envisagés, pour résoudre le problème de la réécriture, sont incomplets. En effet, la logique $\mathcal{ALCNR}$ ne permet de simuler que partiellement le constructeur $\mathcal{O}_v$.

Une solution pour simuler correctement $\mathcal{O}_v$, serait d'utiliser un constructeur de contraintes de cardinalité sur les concepts [5]. Cependant à notre connaissance, aucune étude n'a porté sur la réécriture en présence d'un tel constructeur. De plus pour des logiques permettant ce constructeur, il n'existe pas d'algorithme de subsomption structurelle. Ainsi le problème de la réécriture de requêtes en présence de $\mathcal{O}_v$ reste un problème ouvert.

Nous avons vu qu'il est possible de trouver une alternative à l'utilisation de $\mathcal{O}_v$. Nous nous intéressons maintenant à une alternative de modélisation des contraintes de valeurs, en dehors du cadre formel des logiques de description.

3.5.2 Alternatives par rapport à la spécification des contraintes de valeurs

Une alternative possible à la spécification des contraintes de valeurs dans les concepts serait de traiter les contraintes en dehors des descriptions de concepts. Autrement dit, il n'est plus possible de former des concepts, i.e. des vues et des requêtes, avec le constructeur $\mathcal{O}_v$. Dans ce cas, les contraintes de valeurs ne sont pas prises en compte dans le processus de réécriture. Aussi supposons qu'une requête demande les parcelles culturales qui ont reçu les produits P_1 ou P_2 ou P_5 : $Q \equiv ParcelleCulturale \sqcap \forall aRecu.AlphaNum$. La contrainte de valeurs peut être spécifiée en dehors du concept Q de la façon suivante : $ContrainteV(aRecu) = \{P_1, P_2, P_5\}$.

Dans le cadre d'un système de médiation, si les contraintes ne sont pas traitées lors du

processus de réécriture, les contraintes de valeurs peuvent être prises en compte par filtrage des données des sources, soit au niveau du médiateur, soit au niveau des sources.

Dans le cas où les données sont filtrées au niveau du médiateur, les hypothèses suivantes doivent être vérifiées :

- les sources doivent permettre la sélection sur l'attribut $produit$. Autrement dit les sources renvoient des informations sur la parcelle culturale mais également sur les produits que la parcelle a reçu.
- le médiateur implante l'opérateur de sélection, i.e. il est capable de sélectionner effectivement les données qui vérifient la contrainte.

Dans ce cas, après avoir réécrit la requête, le médiateur récupère les données des sources contenant des parcelles qui ont reçu des produits. Ensuite, il sélectionne parmi ces données celles qui ont effectivement reçu les produits P_1 ou P_2 ou P_5 (ainsi que celles qui n'ont reçu aucun produit). Dans cette alternative, on augmente donc la charge du médiateur. De plus, les coûts de communication via le réseau augmentent car les données de toutes les sources décrites par une vue qui comporte une restriction de valeur $\forall aRecu.AlphaNum$ sont importées dans le médiateur.

La deuxième possibilité est de filtrer les données au niveau des sources. Dans ce cas, on suppose que les sources ont la capacité de sélection sur les données (et notamment sur l'attribut produit). Ceci constitue une hypothèse forte, surtout dans notre contexte applicatif où une majorité des sources est formée de simples logiciels dédiés au domaine du développement durable. Si on suppose cette hypothèse vérifiée, alors le médiateur lors du processus de réécriture identifie les vues pertinentes à la résolution de la requête, i.e. les vues qui comportent une restriction de valeur $\forall aRecu.AlphaNum$. Après exécution des vues sur les données des sources, il faut sélectionner les données qui ont effectivement reçu les produits P_1 ou P_2 ou P_5 (ainsi que celles qui n'ont reçu aucun produit). Ainsi le médiateur est déchargé de la tâche de sélection et un volume plus faible de données circule sur le réseau.

Cependant, ces deux alternatives posent un problème fondamental. Le traitement des contraintes de valeurs en dehors du processus de réécriture entraînent la perte de l'interaction entre les contraintes de cardinalité et les contraintes de valeurs. Aussi un algorithme de réécriture qui traiterait les contraintes à part n'est pas complet, i.e. il fournit moins de réponses certaines qu'un algorithme de réécriture qui prend en compte les contraintes de valeurs. Par exemple, considérons une requête Q qui interroge les parcelles culturales qui ont reçu moins de trois produits, i.e. $Q \equiv ParcelleCulturale \sqcap \leq 3aRecu$. Avec des vues V_i décrites comme des parcelles culturales qui ont reçu quelque chose de type alphanumérique, i.e. $V_i \equiv ParcelleCulturale \sqcap \forall aRecu.AlphaNum$, il n'est pas possible de déduire les parcelles culturales qui ont reçu moins de trois produits alors que cette information réside implicitement dans les contraintes de valeurs.

Après avoir présenté et discuté nos choix de modélisation des contraintes de valeurs dans le cadre des logiques de description, nous nous intéressons maintenant au problème de la réécritures de requêtes dans ce cadre.

Troisième partie

Réécriture de requêtes en présence des contraintes de valeurs

Cette partie, composée de trois chapitres, est consacrée à l'étude du problème de la réécriture de requêtes en termes de vues en présence de contraintes de valeurs. Dans le chapitre 4, nous formalisons le problème de la réécriture de requêtes en termes de vues dans le cadre des logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ et nous donnons sa complexité. Dans le chapitre 5, nous présentons le cadre de découverte des connaissances introduit par Mannila et Toivonen dans [57]. Puis nous montrons comment le problème de la réécriture de requêtes en présence de contraintes de valeurs peut être résolu dans ce cadre.

Chapitre 4

Réécriture de requêtes dans $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$

Dans ce chapitre, nous étudions le problème de réécriture de requêtes en termes de vues dans le cadre des logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$. Dans un premier temps, nous rappelons le contexte général du problème puis nous définissons formellement la notion de réponse certaine dans ce cadre. Nous proposons par la suite un processus de réécriture de requêtes pour calculer les réponses certaines. Nous montrerons que dans notre cadre d'étude, le processus de réécriture proposé permet de calculer toutes les réponses certaines d'une requête donnée. Nous caractérisons par la suite, les conditions nécessaires que doivent vérifier les réécritures dans les deux logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ et nous esquissons un algorithme qui calcule effectivement les réécritures d'une requête dans ce contexte.

4.1 Cadre général

Nous nous intéressons au problème de la réécriture de requêtes en termes de vues dans le cadre des systèmes de médiation dont la figure 4.1. rappelle l'architecture. Cette figure montre un médiateur recevant une requête Q, disposant d'un schéma $\mathcal{S}$ et d'un ensemble de vues $\mathcal{V}$. Nous nous plaçons dans le cadre d'un système de médiation suivant une approche LAV, i.e. les vues sont décrites en termes du schéma global. Nous supposons également que les vues respectent l'hypothèse du monde ouvert (OWA), i.e. les sources de données sont supposées incomplètes.

Dans cette section, nous montrons comment ce cadre de médiation et les hypothèses sous-jacentes sont décrites dans le contexte des logiques de description.

4.1.1 Description du schéma global et des vues

Pour s'abstraire d'un langage précis, nous supposerons dans la suite que $\mathcal{S}$, $\mathcal{V}$ et Q sont décrites dans un langage $\mathcal{L}$ qui sera instancié plus tard, avec la logique $\mathcal{FL}_0(\mathcal{O}_v)$ ou bien la logique $\mathcal{ALN}(\mathcal{O}_v)$. De manière plus précise, les éléments $\mathcal{S}$, $\mathcal{V}$ et Q sont définis comme suit :

– le schéma global $\mathcal{S}$ est une terminologie acyclique dans $\mathcal{L}$ contenant uniquement des définitions.

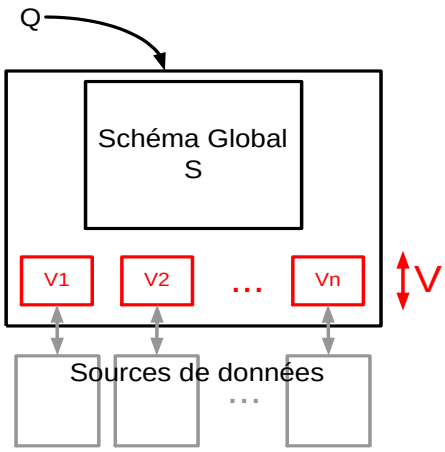

FIG. 4.1 – Système de médiation

- une requête Q est un concept dans $\mathcal{L}$, défini en utilisant les concepts et les rôles de $\mathcal{S}$. Sans perte de généralité, Q est supposée être dans sa forme normale.
- $\mathcal{V}$ est une terminologie primitive acyclique dans $\mathcal{L}$. Les spécifications primitives de $\mathcal{V}$ sont appelées *vues*. Les vues de $\mathcal{V}$ sont décrites en termes de $\mathcal{S}$. Sans perte de généralité, $\mathcal{V}$ est supposée régulière et aplatie. De plus, les vues de $\mathcal{V}$ sont supposées être dans leur forme normale.
- le langage de réécriture utilise les deux constructeurs $\sqcap$ et $\sqcup$.

L'utilisation de spécifications primitives pour modéliser les vues est motivée par le fait qu'elles permettent de capturer l'hypothèse du monde ouvert (OWA). En effet, les spécifications primitives constituent des descriptions incomplètes, dans le sens où elles décrivent uniquement les conditions nécessaires que doivent vérifier les instances des vues. Ainsi un individu, instance d'un concept D, n'est pas forcément instance d'une vue V, qui serait définie par l'axiome $V \sqsubseteq D$.

L'exemple 19 qui suit, illustre les notions introduites précédemment dans le cas où $\mathcal{L}$ est la logique $\mathcal{ALN}(\mathcal{O}_v)$.

Exemple 19
Dans la table 4.1, la terminologie $\mathcal{S}$ composée uniquement de définitions, est un exemple de schéma de médiation, alors que la terminologie $\mathcal{V}$ régulière, aplatie et normalisée liste l'ensemble des vues d'un médiateur. Chaque vue dans $\mathcal{V}$ une fois régularisée est exprimée sous forme d'une définition. Par exemple, la définition de la vue V_1 permet d'effectuer deux déductions à propos de la source de données qui lui est associée. Premièrement elle fournit des informations sur les parcelles culturales qui appartiennent à une exploitation. Deuxièmement elle ne donne qu'un sous-ensemble de l'ensemble des parcelles culturales qui appartiennent à un exploitation agricole. En effet la vue V_1 comporte également un atome $\overline{V_1}$ qui indique qu'elle est contrainte par une propriété qui lui est propre et qui la distingue de l'ensemble de toutes les parcelles culturales appartenant à une exploitation.

	Terminologie $\mathcal{S}$
$ParcelleTraitee \equiv$	$ParcelleCulturale \sqcap \geq 1\, aRecu \sqcap$
	$\forall aRecu.categoriePesticide.\{C_1, C_2, \ldots, C_{128}\} \sqcap$
	$\forall compriseDans.appartient.ExploitationAgricole$
$ParcelleBio \equiv$	$ParcelleCulturale \sqcap \geq 1\, aCertifBio \sqcap$
	$\geq 1\, numCommune \sqcap \forall numCommune.\{01000, 01001, \ldots, 98999\}$
$ParcelleEpandue \equiv$	$ParcelleCulturale \sqcap \geq 1\, dans$
	$\sqcap \forall dans.PerimetreEpandage \sqcap \forall soumise.ContratCoop$
	$\sqcap \forall soumise.souscritPar.numSiret.Alphanum$
	Terminologie $\mathcal{V}$
$V_1 \equiv$	$\overline{V_1} \sqcap ParcelleCulturale \sqcap$
	$\forall compriseDans.appartient.ExploitationAgricole$
$V_2 \equiv$	$\overline{V_2} \sqcap ParcelleCulturale \sqcap \geq 1\, dans \sqcap$
	$\forall dans.PerimetreEpandage$
	$\sqcap \forall numCommune.\{23450, 23505, 63585, 63218, 63300\}$
$V_3 \equiv$	$\overline{V_3} \sqcap ParcelleCulturale \sqcap \forall soumise.ContratCoop$
$V_4 \equiv$	$\overline{V_4} \sqcap ParcelleCulturale \sqcap \geq 1\, dans$
	$\sqcap \forall dans.PerimetreEpandage$
	$\sqcap \forall numCommune.\{23450, 23505, 63585, 63218, 63300\}$
$V_5 \equiv$	$\overline{V_5} \sqcap ParcelleCulturale \sqcap$
	$\forall compriseDans.appartient.ExploitationAgricole$
$V_6 \equiv$	$\overline{V_6} \sqcap ParcelleCulturale \sqcap \forall soumise.ContratCoop$
$V_7 \equiv$	$\overline{V_4} \sqcap ParcelleCulturale \sqcap \geq 1\, dans$
	$\sqcap \forall dans.PerimetreEpandage$
	$\sqcap \forall numCommune.\{03250, 03325, 63520, 63540, 63585\}$
	Requête Q
$Q \equiv$	$ParcelleCulturale \sqcap \forall dans.PerimetreEpandage \sqcap$
	$\forall soumise.ContratCoop$

TAB. 4.1 – Exemple de configuration d'un système de médiation

Les vues V_2 et V_4 indiquent que les sources de données qui leur sont associées, fournissent un sous-ensemble des parcelles culturales prévues à l'épandage dans les communes 23450 ou 23505 ou 63585 ou 63218 ou 63300. Les deux sous-ensembles de parcelles culturales des vues V_2 et V_4 peuvent éventuellement s'intersecter.

Le concept Q donné dans la table 4.1 décrit une requête qui demande les parcelles cadastrales prévues à l'épandage (dans un périmètre d'épandage), mises en culture et dont la culture est soumise à un contrat avec une coopérative.

○

Après avoir présenté la description d'un système de médiation dans les logiques de description, nous définissons ci-dessous la notion de réponses certaines dans ce cadre.

4.1.2 Notion de réponses certaines

Soit le système de médiation $< \mathcal{S}, \mathcal{V}, \mathcal{M} >$. Soit Q une requête en termes de $\mathcal{S}$. Nous notons par V^{ext}, l'extension de la vue V de $\mathcal{V}$ et par $\mathcal{V}^{ext}$, l'union des extensions des vues de $\mathcal{V}$.

On cherche à caractériser la notion de réponses certaines dans le contexte des logiques de description. Intuitivement, un individu t est une réponse certaine d'une requête Q, si t appartient à l'évaluation de Q sur toute base de données instanciant $\mathcal{S}$, conforme à $\mathcal{V}^{ext}$. Plus formellement, la notion de réponses certaines est définie comme suit :

Définition 4.1.1
Soient le système de médiation $< \mathcal{S}, \mathcal{V}, \mathcal{M} >$ et Q une requête en termes de $\mathcal{S}$. Soient V^{ext}, l'extension de la vue V de $\mathcal{V}$ et $\mathcal{V}^{ext}$, l'union des extensions des vues de $\mathcal{V}$.

L'ensemble $\{t \in \mathcal{V}^{ext} \mid t \in Q^{\mathcal{I}}, \forall \mathcal{I}$ modèle de $\mathcal{S}$ et de $\mathcal{V}$ tel que $V^{ext} \subseteq V^{\mathcal{I}}, \forall V \in \mathcal{V}\}$ est l'ensemble des réponses certaines de Q.

Cette définition permet de capturer la notion de réponses certaines introduite dans [1] sous l'hypothèse du monde ouvert, mais formulé dans le cadre des logiques de description.

4.1.3 Problème de réécriture de requêtes en termes de vues

Considérons un système de médiation disposant d'un schéma global $\mathcal{S}$. Soient un ensemble de vues $\mathcal{V}$ et une requête Q exprimés en termes de $\mathcal{S}$. $\mathcal{S}$, $\mathcal{V}$ et Q sont décrites dans $\mathcal{L} \in \{\mathcal{FL}_0(\mathcal{O}_v), \mathcal{ALN}(\mathcal{O}_v)\}$. Nous nous intéressons dans ce contexte au problème du calcul des réponses à une requête Q, i.e. au calcul des réponses certaines de Q, en utilisant uniquement les vues de $\mathcal{V}$. Comme discuté précédemment, ce problème peut être résolu par un processus de *réécriture de requêtes en termes de vues*. Un tel processus permet de reformuler une requête Q en une expression qui réfère uniquement les vues de $\mathcal{V}$. Pour calculer les réponses aux requêtes des utilisateurs, les réécritures sont ensuite évaluées sur les extensions des vues. De manière générale, un processus de réécriture de requêtes ne garantit pas toujours d'obtenir *toutes* les réponses certaines [18]. La proposition suivante montre dans quelles conditions, les réponses certaines peuvent être effectivement obtenues.

Proposition 4.1.1
Soit $\mathcal{S}$ un schéma dans $\mathcal{L}$. Soient une requête Q et un ensemble des vues $\mathcal{V}$ tels que Q et les vues de $\mathcal{V}$ sont exprimés dans $\mathcal{L}$, en termes de $\mathcal{S}$. Si t est une réponse certaine de Q, alors il existe une conjonction $V_1 \sqcap \ldots \sqcap V_n$ de vues de $\mathcal{V}$ telle que
- $V_1 \sqcap \ldots \sqcap V_n \sqsubseteq Q$ *et*
- $t \in V_1^{ext} \cap \ldots \cap V_n^{ext}$.

Cette proposition montre le rôle central des conjonctions de vues subsumées par Q, dans le calcul des réponses certaines.

La preuve de cette proposition est donnée en Annexe A.2, page 123. Le principe de la preuve est le suivant. Par définition, si t est une réponse certaine de Q, alors il existe un ensemble de vues, M, formé de toutes les vues qui contiennent t dans leur extension. Il existe alors une conjonction de vues C, i.e. la conjonction des vues de M, telle que $t \in C^{ext}$. Il reste alors à montrer que C est obligatoirement subsumée par la requête Q. Pour cela on suppose que C n'est pas subsumée par Q. On montre qu'il existe alors un modèle de S, noté J, tel que

$V^{ext} \subseteq V^{\mathcal{J}}\ \forall V \in \mathcal{V}$, dans lequel t n'appartient pas à $Q^{\mathcal{J}}$, et donc que t n'est pas une réponse certaine de Q, ce qui contredit l'hypothèse selon laquelle t est une réponse certaine de Q.

Par conséquent, pour obtenir les réponses certaines d'une requête Q, il est nécessaire de calculer toutes les conjonctions de vues de $\mathcal{V}$ subsumée par Q, puis d'en faire l'union. Ces expressions de vues sont appelées réécritures. Leur définition est donnée ci-dessous.

Définition 4.1.2 (Réécriture)
Soient $\mathcal{V}$ une terminologie de vues et Q un concept dans $\mathcal{L}$. Q' est une réécriture de Q en termes de $\mathcal{V}$, ou tout simplement réécriture quand il n'y a pas d'ambigüité, ssi :

1. *Q' est un concept dans le langage $\{\sqcap, \sqcup\}$ qui réfère uniquement les noms des vues de $\mathcal{V}$,*
2. *$Q' \sqsubseteq Q$.*

Une réécriture exprimée uniquement dans le langage $\{\sqcap\}$ est une conjonction de vues appelée *réécriture conjonctive*.

Généralement, il n'est pas intéressant de calculer toutes les réécritures possibles d'une requête donnée. En effet, étant données deux réécritures Q' et Q'' d'une requête Q, la réécriture Q'' n'est pas intéressante si elle renvoie toujours un sous-ensemble des réponses déjà retournées par Q'. Aussi dans le contexte des systèmes de médiation, on est intéressé par les réécritures qui renvoient le maximum de réponses. Ce type de réécriture appelée réécriture maximale, est défini ci-dessous :

Définition 4.1.3 (Réécriture maximale)
Soient $\mathcal{V}$ une terminologie de vues et Q un concept dans $\mathcal{L}$. Q' est une réécriture maximale de Q en termes de $\mathcal{V}$ ssi :

1. *Q' est une réécriture de Q,*
2. *il n'existe pas de réécriture Q_1 de Q telle que $Q' \sqsubseteq Q_1 \sqsubseteq Q$ et $Q' \not\equiv Q_1$.*

De façon informelle, une réécriture Q' de Q est maximale s'il n'existe pas d'autre réécriture Q_1 de Q dont l'ensemble des réponses fournisse un strict sur-ensemble des réponses de Q' et ce quelque soit la base de données considérée.

En pratique, on est généralement intéressé par les réécritures maximales *non redondantes* d'une requête Q. Par exemple, soit une réécriture maximale Q' avec $Q' \equiv C_1 \sqcup C_2 \sqcup C_3 \sqcup C_4$ où chaque C_i, avec $i \in \{1, ..., 4\}$, est une réécriture conjonctive de Q. On dit que Q' est redondante s'il existe deux réécritures conjonctives C_i et C_j avec $i, j \in \{1, ..., 4\}$ et $i \neq j$, tels que C_i subsume C_j. Clairement les réécritures redondantes ne sont pas intéressantes à calculer dans la perspective de répondre à une requête Q car elles sollicitent inutilement des vues. La notion de redondance est formellement définie comme suit :

Définition 4.1.4 (Redondance)
Une réécriture $Q' \equiv C_1 \sqcup ... \sqcup C_n$, où les C_i sont des conjonctions de vues, est dite redondante s'il existe $i \in \{1, ..., n\}$ et $j \in \{1, ..., n\}$ avec $i \neq j$ tels que $C_i \sqsubseteq C_j$.

Les réécritures redondantes ne sont pas intéressantes en pratique. En effet, comme illustré ci-dessous, si Q'' est une réécriture redondante, on peut toujours trouver une réécriture Q' non redondante qui lui est équivalente et dont l'évaluation est plus efficace car Q' sollicite moins de

vues.

Exemple 20
Soit $Q \equiv ParcelleCulturale \sqcap \forall recoit.\{P_1, P_2, P_{10}\}$.
Soient les vues $\{V_1, V_2, V_3, V_4, V_5\}$ de $\mathcal{V}$ telles que :
$V_1 \equiv \overline{V_1} \sqcap ParcelleCulturale$
$V_2 \equiv \overline{V_2} \sqcap ParcelleCulturale$
$V_3 \equiv \overline{V_3} \sqcap ParcelleCulturale$
$V_4 \equiv \overline{V_4} \sqcap \forall recoit.\{P_1, P_{10}\}$
$V_5 \equiv \overline{V_5} \sqcap \forall recoit.\{P_5\}$
Considérons que $Q'' \equiv (V_1 \sqcap V_4) \sqcup (V_2 \sqcap V_4) \sqcup (V_3 \sqcap V_4) \sqcup (V_3 \sqcap V_4 \sqcap V_5)$ est une réécritures maximale de Q. Cependant Q'' est redondante car la conjonction de vues $V_3 \sqcap V_4 \sqcap V_5$ est subsumée par la conjonction $V_3 \sqcap V_4$. En supprimant la conjonction $V_3 \sqcap V_4 \sqcap V_5$ de Q'' on obtient la réécriture $Q' \equiv (V_1 \sqcap V_4) \sqcup (V_2 \sqcap V_4) \sqcup (V_3 \sqcap V_4)$ qui est équivalente à Q''. Il n'est pas intéressant d'évaluer Q'' car elle sollicite plus de vues que Q' pour donner finalement le même ensemble de réponses que Q'. ∘

La notion de redondance peut également s'appliquer aux réécritures conjonctives. Dans ce cas, du fait de l'hypothèse du monde ouvert, la seule façon d'avoir redondance est d'avoir plusieurs occurrences de la même vue dans la réécriture. Par exemple, la réécriture $V_1 \sqcap V_2 \sqcap V_2 \sqcap V_2 \sqcap V_1$ est redondante par rapport à la réécriture $V_1 \sqcap V_2$. Aussi sans perte de généralité, on supposera que les réécritures conjonctives sont non redondantes.

Nous définissons maintenant de manière précise, le problème de réécriture de requêtes que nous cherchons à résoudre dans ce mémoire.

Problème 1 *Soient $\mathcal{V}$ une terminologie de vues et Q une requête, toutes les deux exprimées en termes du même schéma global dans le langage $\mathcal{L}$.*

Le problème rewrite(Q,$\mathcal{V}$,$\mathcal{L}$) consiste à calculer toutes les réécritures maximales, non redondantes de Q .

On s'intéresse maintenant à la résolution de *rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*. Nous montrons dans le reste de cette section qu'une étape importante dans la résolution de ce problème consiste dans le calcul des *réécritures conjonctives maximales* de Q. Une réécriture conjonctive maximale de Q est définie comme suit :

Définition 4.1.5 (Réécriture conjonctive maximale)
Soient $\mathcal{V}$ une terminologie de vues et Q un concept dans $\mathcal{L}$. Q' est une réécriture conjonctive maximale de Q en termes de $\mathcal{V}$ ssi :

1. *Q' est une réécriture dans $\{\sqcap\}$*
2. *il n'existe pas de réécriture Q_1 dans $\{\sqcap\}$ de Q telle que $Q' \sqsubseteq Q_1 \sqsubseteq Q$ et $Q' \not\equiv Q_1$.*

Le lemme suivant montre que dans le cadre des logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$, la résolution de *rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)* consiste à calculer toutes les réécritures conjonctives maximales de Q.

Lemme 4.1.1
Soit un problème rewrite(Q,$\mathcal{V}$,$\mathcal{L}$) avec $\mathcal{L} \in \{\mathcal{FL}_0(\mathcal{O}_v), \mathcal{ALN}(\mathcal{O}_v)\}$. Soit $\{Q_1, \ldots, Q_n\}$ l'ensemble de toutes les réécritures conjonctives maximales de Q en termes de $\mathcal{V}$.

Q' est une réécriture maximale de Q en termes de $\mathcal{V}$ si et seulement si $Q' \equiv \sqcup_{i=1}^{n} Q_i$.

Ce lemme donne deux résultats : *(i)* la solution à *rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)* avec $\mathcal{L} \in \{\mathcal{FL}_0(\mathcal{O}_v), \mathcal{ALN}(\mathcal{O}_v)\}$ est unique, et *(ii)* elle peut être facilement obtenue à partir des réécritures conjonctives maximales de Q. La preuve de ce lemme s'appuie sur deux autres lemmes donnés ci-après. Ces deux lemmes exploitent les hypothèses du monde ouvert capturées par les vues, afin de définir les conditions nécessaires et suffisantes pour déterminer quand une réécriture subsume une autre réécriture.

Le premier lemme caractérise la subsomption entre deux réécritures conjonctives.

Lemme 4.1.2
Soit $\mathcal{V}$ une terminologie dans $\mathcal{L}$. Soient Q_1 et Q_2 deux conjonctions satisfiables de vues de $\mathcal{V}$.

$Q_1 \sqsubseteq Q_2$ si et seulement si Q_2 est formé d'un sous-ensemble des vues apparaissant dans Q_1.

Ce résultat est une conséquence directe de l'hypothèse du monde ouvert.

Dans le même esprit que le lemme 4.1.2, le lemme suivant permet de caractériser la subsomption entre deux réécritures de requêtes mais qui sont exprimées cette fois dans le langage $\{\sqcap, \sqcup\}$.

Lemme 4.1.3
Soit $\mathcal{V}$ une terminologie dans $\mathcal{L}$. Soient $Q_1 \equiv \sqcup_{j=1}^{n} C_j$ et $Q_2 \equiv \sqcup_{k=1}^{p} C_k$, telles que $\forall j \in \{1, ..., n\}$ et $\forall k \in \{1, ..., p\}$, C_j et C_k sont des conjonctions de vues de $\mathcal{V}$ satisfiables.

$Q_1 \sqsubseteq Q_2$ si et seulement si pour chaque chaque C_j dans Q_1, il existe C_k dans Q_2 tel que $C_j \sqsubseteq C_k$.

Nous donnons ci-dessous le principe de la preuve du lemme 4.1.1 qui repose sur les deux lemmes 4.1.2 et 4.1.3 introduits précédemment.

1) Preuve de ($\Rightarrow$)

   Cette preuve comprend deux volets. Le premier consiste à démontrer que i) la réécriture maximale Q' de Q n'est formée que de réécritures conjonctives maximales, tandis que le deuxième consiste à montrer que ii) Q' est formée de *toutes* les réécritures conjonctives maximales de Q.

   La preuve du volet i) est une démonstration par l'absurde. On suppose que Q' est une réécriture maximale de Q et qu'une des réécritures conjonctives Q_k, où $k \in \{1, \ldots, n\}$ n'est pas maximale. On montre alors qu'il existe une réécriture qui subsume strictement Q'.

   Soit une réécriture Q'', obtenue en remplaçant dans Q', Q_k par Q'_k tel que $Q_k \sqsubseteq Q'_k$ et tel que Q'_k est maximale. Par définition on a $Q' \sqsubseteq Q''$. On veut maintenant prouver que Q'' n'est pas subsumée par Q' afin de montrer que $Q' \not\equiv Q''$. A partir du lemme 4.1.3 et de l'hypothèse de non redondance portant sur les réécritures, on déduit que Q'' n'est pas subsumée par Q'. Ainsi il existe une réécriture Q'' qui subsume strictement Q' ce qui contredit l'hypothèse de départ.

La preuve du volet ii) s'appuie sur la démonstration de la contraposée. Autrement dit Q' est supposée ne pas être formée de toutes les réécritures conjonctives maximales Q_i, $i \in \{1, \ldots, n\}$. On montre alors que Q' ne peut pas être une réécriture maximale de Q.
Il existe une réécriture Q'' formée de l'union de toutes les réécritures Q_i, pour $i \in \{1, \ldots, n\}$. Par définition, on a $Q' \sqsubseteq Q''$. On cherche ensuite à montrer que Q'' n'est pas subsumée par Q' afin de montrer que $Q' \not\equiv Q''$. Comme dans le premier volet, par le lemme 4.1.3 et grâce à l'hypothèse de non redondance, on obtient que Q'' n'est pas subsumée par Q'. Ainsi Q' ne peut pas être maximale. Ceci clôt la preuve de ($\Rightarrow$).

1) Preuve de ($\Leftarrow$)
La preuve de la justesse du lemme 4.1.1 est également une démonstration par l'absurde. Q' est l'union de toutes les réécritures conjonctives maximales de Q. On suppose que Q' n'est pas une réécriture maximale de Q. Alors il existe une réécriture Q_1 qui subsume strictement Q'. En s'appuyant sur les lemmes 4.1.2 et 4.1.3 et sur l'hypothèse de non redondance des réécritures, on montre qu'une telle réécriture Q_1 ne peut pas exister.

Le détail des preuves des lemmes 4.1.3, 4.1.2 et 4.1.1 est donné en annexe A.3, page 124.

4.1.4 Calcul des réécritures conjonctives maximales

Nous définissons à présent le problème du calcul des réécritures conjonctives maximales. Comme discuté auparavant, ces dernières jouent un rôle clé dans la résolution du problème *rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*.

Problème 2 *Soient $\mathcal{V}$ une terminologie de vues et Q une requête, toutes les deux exprimées en termes du même schéma global, dans le langage $\mathcal{L}$.*

Le problème conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$) consiste à calculer toutes les réécritures conjonctives maximales de Q .

Nous décrivons ci-dessous une approche générale pour résoudre *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*. Nous déclinons ensuite cette approche dans le contexte des logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$.

Considérons un problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*. L'espace de recherche de ce problème est l'ensemble de toutes les conjonctions de vues de taille 1 à $|\mathcal{V}|$, modulo les vues non satisfiables, ou non consistantes, avec la requête, i.e. les vues V telles que $V \sqcap Q \equiv \bot$. A titre d'exemple, la figure 4.2 ci-dessous illustre l'espace de recherche des réécritures conjonctives maximales, i.e. le treillis des parties des vues concernées, dans le cas où le nombre de vues consistantes avec la requête est égal à 8.

L'espace de recherche étant exponentiel (i.e. de taille $2^{|\mathcal{V}|}$), l'exploration exhaustive d'un tel espace n'est pas réaliste, notamment dans les contextes applicatifs comme le nôtre, où le nombre de vues peut être grand. Comme illustré en figure 4.3, page 68, une réduction de cet espace de recherche est possible en bornant la taille des réécritures conjonctives maximales recherchées, i.e. le nombre de vues qui apparaissent dans la réécriture. L'existence d'une telle borne et sa caractérisation dépend du langage utilisé pour décrire les vues et les requêtes, ainsi que du langage de réécriture. Nous donnons ci-dessous un lemme qui constitue une première étape pour caractériser une telle borne. Ce lemme sera utilisé par la suite pour spécifier une borne sur la taille des réécritures dans $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}$.

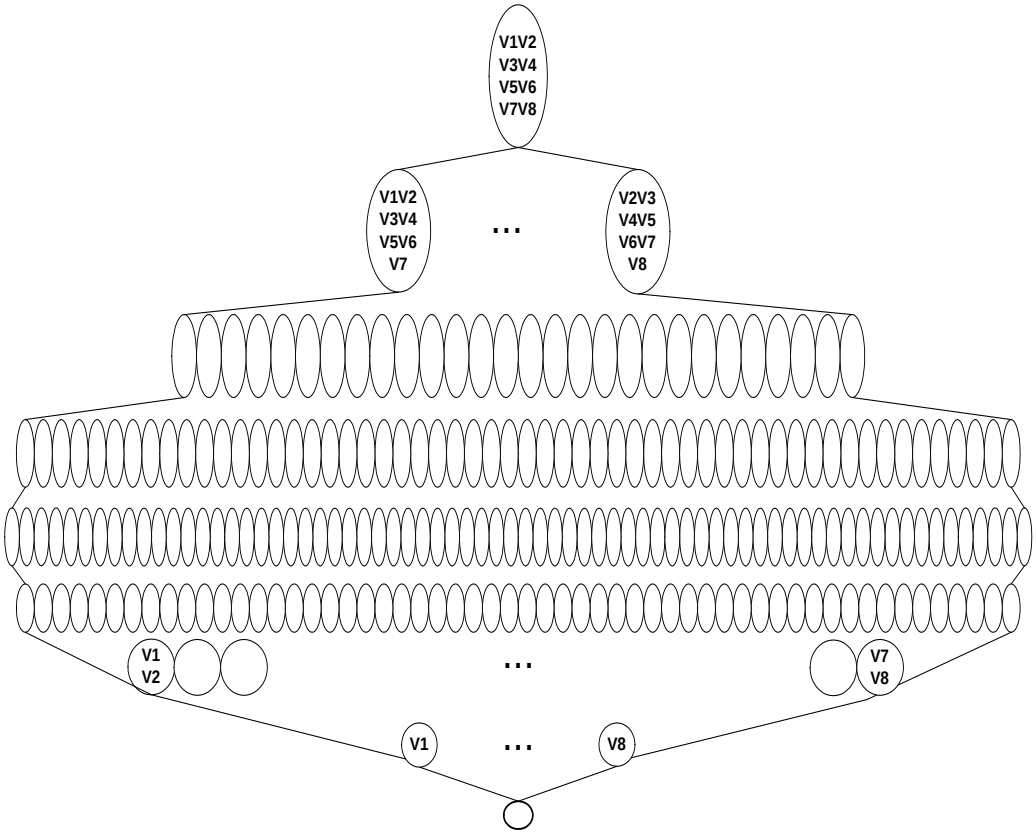

FIG. 4.2 – Espace de recherche des réécritures

Lemme 4.1.4
Soit $Q' \equiv \sqcap_{i=1}^{n} V_i$ tel que $Q' \sqsubseteq Q$.

Q' est une réécriture conjonctive maximale de Q si et seulement si pour chaque concept Q'' obtenu en supprimant dans Q' une vue V_j, avec $j \in \{1, ..., n\}$, on a $Q'' \not\sqsubseteq Q$.

La preuve de ce lemme est donnée en annexe A.4, page 127. Ce lemme stipule qu'une réécriture conjonctive maximale d'une requête Q est nécessairement formée d'un ***sous-ensemble minimal*** de noms de vues de $\mathcal{V}$ [16] dont la conjonction des éléments est subsumée par Q. Par exemple si $Q' \equiv V_1 \sqcap V_2 \sqcap V_4$ est une réécriture maximale de Q alors $V_1 \sqcap V_2$, $V_1 \sqcap V_4$ et $V_2 \sqcap V_4$ ne sont pas subsumées par Q.

En dépit de l'existence d'une borne sur la taille des réécritures, l'espace de recherche reste exponentiel. Pour éviter de parcourir de manière exhaustive l'espace de recherche du problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*, nous proposons de suivre une approche des paniers, dans l'esprit du "bucket algorithm" [56]. Informellement, cette approche fonctionne de la manière suivante. Etant donné un problème de réécriture *rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)* avec $Q \equiv \forall w_1.P_1 \sqcap \ldots \sqcap \forall w_n.P_n$, la principale idée est d'abord de considérer chaque atome $\forall w_i.P_i$ de Q isolément. A chaque atome $\forall w_i.P_i$ de la requête, est associé un panier qui contient toutes les réécritures conjonctives maximales de cet atome. Ensuite dans une deuxième étape, les réécritures candidates de Q sont calculées

[16] La minimalité est considérée ici dans le sens de l'inclusion ensembliste.

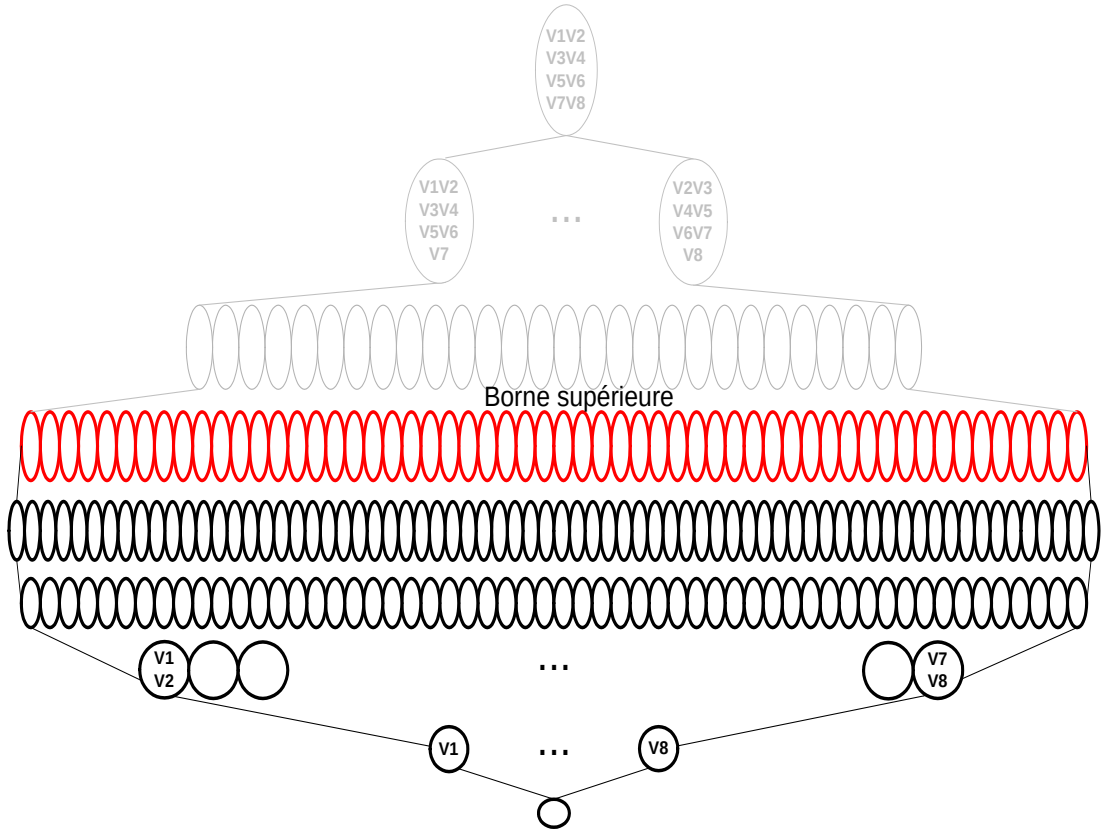

FIG. 4.3 – Borne supérieure de l'espace de recherche des réécritures

en effectuant le produit cartésien entre les paniers. Ceci permet d'obtenir un sur-ensemble de toutes les réécritures conjonctives maximales de Q. Pour obtenir effectivement les réécritures conjonctives maximales, i.e. les solutions du problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*, les réécritures inconsistantes et non maximales doivent ensuite être supprimées de ce sur-ensemble.

L'algorithme 1, page 69, appelé CalculReec décrit l'approche des paniers dans notre cadre d'étude. Cet algorithme prend en entrée une requête Q et une terminologie de vues $\mathcal{V}$ et retourne l'ensemble, noté RCM, de toutes les réécritures conjonctives maximales de Q. L'algorithme 1 se décompose en trois grandes étapes :

- la construction du panier de chaque atome d'une requête Q (lignes 1-9) qui fait appel à la fonction Construire_Panier. Construire_Panier calcule les réécritures d'un atome quelconque.
- la suppression des réécritures insatisfiables et non maximales (lignes 10-13).
- le produit cartésien entre les paniers et la suppression des réécritures insatisfiables et non maximales (lignes 14-17). Le calcul du produit cartésien permet de générer les réécritures conjonctives globales de Q tandis que la phase de suppression permet de ne renvoyer en sortie, que les réécritures conjonctives maximales de Q.

L'intérêt de l'approche des paniers est d'éviter un parcours exhaustif de l'espace de recherche en réduisant le nombre de conjonctions de vues à considérer pendant le calcul des réécritures conjonctives maximales. En effet, l'approche des paniers permet de ne considérer

Algorithme 1 CalculReec

Entrée(s) : $Q \equiv \sqcap_{i=1}^{n} \forall w_i.P_i$, $\mathcal{V} = \{V_1, ..., V_m\}$
Sortie(s) : RCM

/* Initialisation des paniers */
pour chaque $\forall w_i.P_i$ **faire**
    $B(w_i, P_i)$:=$\emptyset$
fin pour
/* Construction des paniers*/
pour chaque atome $\forall w_i.P_i$ **faire**
    /* appel de l'algorithme de construction des paniers */
    $B(w_i, P_i)$:=Construire_Panier($\forall w_i.P_i$,$\mathcal{V}$)
fin pour
/* Suppression des réécritures insatisfiables, et non maximales */
pour chaque panier $B(w_i, P_i)$ **faire**
    $B(w_i, P_i)$:=F_Supp_NonMax($B(w_i, P_i)$)
fin pour
/* Calcul du produit cartésien */
RCM :=F_Produit_Cart($B(w_i, P_i)$, $i \in \{1, ..., n\}$)
/* Elimination des réécritures insatisfiables et non maximales */
RCM :=F_Supp_NonMax(RCM)
RETURN RCM

que les vues susceptibles d'être sémantiquement pertinentes à la réécriture des atomes d'une requête.

La section qui suit, s'intéresse à la caractérisation des solutions du problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)* dans le cadre des logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$. Cette caractérisation permettra de définir l'algorithme 2, Construire_Panier, utilisé par CalculReec.

4.2 Caractérisation des réécritures dans les logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$

Cette section est décomposée en deux sous-sections. Chacune permet de caractériser les réécritures dans les logiques $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ respectivement, en donnant dans chaque cas :

- la forme des réécritures, i.e. les conditions nécessaires que doivent vérifier les réécritures conjonctives maximales,
- la taille maximale des réécritures,
- la complexité du problème du calcul de toutes les réécritures conjonctives maximales.

4.2.1 Cas du langage $\mathcal{FL}_0(\mathcal{O}_v)$

Comme discuté auparavant, l'approche des paniers permet de réduire le problème de la réécriture d'une requête Q, en la réécriture de ses atomes pris séparément. Aussi sans perte de

généralité, on s'intéresse dans cette section, au problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)* dans le cas où Q est réduite à un seul atome, i.e. $Q \equiv \forall w.P$. Autrement dit, on s'intéresse à la construction du panier $B(w, P)$ associé à l'atome $\forall w.P$. Le lemme suivant permet de caractériser les éléments d'un tel panier dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$, i.e. les ensembles minimaux de vues dont la conjonction est subsumée par $\forall w.P$.

Lemme 4.2.1
Soit le problème conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$) avec $Q \equiv \forall w.P$. Soit l la cardinalité du plus grand des ensembles de valeurs qui apparaît dans les vues ou la requête. Soit $Q' \equiv V_1 \sqcap \ldots \sqcap V_k$. Si $\{V_1, \ldots V_k\}$ est un sous-ensemble ***minimal*** *de $\mathcal{V}$ tel que $w \in V_{Q'}(P)$ alors une des conditions suivantes est vérifiée :*

a) $k = 1$, $P = A$ *et* $\forall w.P \in V_{i_1}$

b) $1 \leq k \leq l + 1$, $P = E$ *et* $\{V_1 ... V_k\}$ *est tel que : pour* $j \in \{1, \ldots, k\}$, $\forall w.E_j \in V_j$, *et* $\cap_{j=1}^{k} E_j \subseteq E$.

Notons que ce lemme peut se reformuler en remplaçant la prémisse par «si Q' est une réécriture conjonctive maximale de Q alors une des conditions (a) ou (b) est vérifiée». En effet, d'après le lemme 4.1.4, si Q' est formée d'un sous-ensemble minimal de $\mathcal{V}$ tel que $w \in V_{Q'}(P)$, alors Q' est une réécriture conjonctive maximale de Q. Le lemme 4.2.1 s'appuie sur la caractérisation de la subsomption dans $\mathcal{FL}_0(\mathcal{O}_v)$, donné dans le théorème 3.3.1. La preuve de ce lemme est donnée en annexe A.5, page 128.

Examinons maintenant les résultats de ce lemme. Ce lemme donne les conditions nécessaires pour qu'un sous-ensemble de vues forme une réécriture conjonctive maximale de Q. Un sous-ensemble minimal de $\mathcal{V}$ peut conduire à une réécriture maximale d'une requête Q si ce sous-ensemble vérifie le cas (a) ou le cas (b) du lemme. Dans le panier d'un atome d'une requête Q, on peut alors trouver des réécritures comportant une seule vue (cas (a)) ou comportant au plus $l + 1$ vues (cas (b)). Notons que le cas où un élément du panier est formé de plusieurs vues (cas (b) du lemme 4.2.1) est une conséquence de la présence du constructeur $\mathcal{O}_v$ dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$. Le calcul des réécritures de ce type constitue l'étape la plus difficile, lors de la résolution du problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)* car il est nécessaire de calculer toutes les conjonctions de vues de taille 1 à $l + 1$. Cette difficulté est spécifique à la présence des contraintes de valeurs.

L'exemple qui suit illustre le lemme 4.2.1.

Exemple 21
Soit une requête Q telle que $Q \equiv ParcelleCulturale$. Rappelons que dans ce cas Q peut être exprimée sous la forme $Q \equiv \forall \epsilon.ParcelleCulturale$ où ϵ est le mot vide.
Supposons qu'il existe deux vues V_1 et V_2 telles que $V_i \equiv \overline{V_i} \sqcap ParcelleCulturale$, pour $i \in 1, 2$. Cherchons à calculer le panier $B(\epsilon, ParcelleCulturale)$ de l'atome $\forall \epsilon.ParcelleCulturale$. Les seules vues qui contiennent dans leur description le concept $ParcelleCulturale$ sont V_1 et V_2. De plus ces vues sont de taille minimale puisqu'elles ne sont formées que d'une seule vue. Aussi le panier de $ParcelleCulturale$ est équivalent à

$$B(\epsilon, ParcelleCulturale) = \{\{V_1\}, \{V_2\}\}.$$

Considérons maintenant la requête $Q \equiv \forall numDepartement.E$, avec $E = \{03, 43, 63\}$ et calculons le panier $B(numDepartement, E)$ de l'atome $\forall numDepartement.E$. Supposons pour cela qu'il existe 4 vues $V_i, i \in \{2, ..., 6\}$ telles que $V_i \equiv \overline{V_i} \sqcap \forall numDepartement.E_i$ où

$E_3 = \{03, 15, 18, 23, 43, 80\}$
$E_4 = \{03, 63\}$
$E_5 = \{01, 07, 11, 43, 63\}$
$E_6 = \{26, 63\}$
Les plus petits sous-ensembles de vues dont la conjonction est subsumée par cet atome sont :
– $\{V_4\}$,
– $\{V_3, V_5\}$,
– $\{V_5, V_6\}$ et
– $\{V_3, V_6\}$
Ainsi $B(numDepartement, E) = \{\{V_4\}, \{V_3 \sqcap V_5\}, \{V_5 \sqcap V_6\}, \{V_3 \sqcap V_6\}\}$.
Dans cet exemple, les paniers ne contiennent que des réécritures conjonctives maximales. Aussi si on considère le cas de la requête $Q \equiv ParcelleCulturale \sqcap \forall numDepartement.E$, avec $E = \{03, 43, 63\}$ à réécrire à l'aide des vues $V_1, V_2, V_3, V_4, V_5, V_6$,
alors à partir des paniers $B(\epsilon, ParcelleCulturale)$ et $B(numDepartement, E)$ et du lemme 4.1.4, on peut déduire que le concept $Q' \equiv V_1 \sqcap V_3 \sqcap V_5$ est un exemple de réécriture conjonctive maximale de Q. ◦

Comme le montre l'exemple 21 ci-dessus, le nombre de vues apparaissant dans une réécriture d'un atome donné (e.g. $\forall numDepartement.E$) est borné par la cardinalité maximale des contraintes de valeurs (ici 6). Le pire des cas apparaît quand les vues possèdent des contraintes de valeurs qui comparées deux par deux, ne se distinguent que par une seule valeur et quand ces contraintes n'ont aucune valeur commune avec E. Dans ce cas, il est nécessaire de générer l'ensemble vide à partir de la conjonction de ces vues. Ce cas est illustré ci-dessous.

Exemple 22
Soit $Q \equiv \forall numDepartement.E$, avec $E = \{63, 43, 23\}$. Soient 5 vues V_i telles que $V_i \equiv \overline{V_i} \sqcap \forall numDepartement.E_i$ où
$E_1 = \{01, 02, 03, 04\}$
$E_2 = \{01, 02, 03, 05\}$
$E_3 = \{01, 02, 04, 05\}$
$E_4 = \{01, 03, 04, 05\}$
$E_5 = \{02, 03, 04, 05\}$.
On constate que $Q' \equiv V_1 \sqcap V_2 \sqcap V_3 \sqcap V_4 \sqcap V_5$ est subsumé par $\forall numDepartement.\emptyset$. D'où, dans cet exemple, Q' est une réécriture conjonctive maximale de Q. Cette réécriture comporte $l + 1$ vues, où l, la cardinalité maximale des contraintes de valeurs, est égale à 4.
Notons que ce type de réécriture doit être considéré dans le processus de réécriture de Q car elle donne les individus qui ne sont pas contraints par un numéro de commune et ainsi ces individus forment un sous-ensemble de la classe d'individus décrite par Q.

◦

Ainsi la taille maximale des réécritures dans le pire des cas est dérivée du nombre nécessaire de vues à intersecter pour obtenir l'ensemble vide comme restriction de valeurs d'un rôle.

Comme conséquence du lemme 4.2.1, on peut maintenant caractériser précisément la taille maximale des réécritures conjonctives maximales dans $\mathcal{FL}_0(\mathcal{O}_v)$. Cette taille maximale est fixée par le théorème suivant.

Théorème 4.2.1
Soit le problème conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$) où $Q \equiv \forall w.P$. Soient n le nombre d'atomes dans Q et l la cardinalité du plus grand des ensembles de valeurs apparaissant dans les vues ou la requête.

Les réécritures conjonctives maximales de Q en termes des vues $\mathcal{V}$, dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$, contiennent au plus $n * (l + 1)$ vues de $\mathcal{V}$.

La démonstration de ce théorème est donnée en annexe A.5, page 128. La taille maximale des réécritures conjonctives maximales d'une requête est calculée à partir du nombre maximal de vues (c.f. cas (b) du lemme 4.2.1), nécessaires pour réécrire *un* atome de cette requête. Comme discuté auparavant et illustré par la figure 4.3, page 68, cette borne supérieure sur la taille des réécritures permet de réduire l'espace de recherche des problèmes *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)* et *rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)*.

Le théorème 4.2.1 permet de déduire un algorithme exponentiel en temps pour calculer les solutions des problèmes *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)* et *rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)*. La complexité de cet algorithme résulte du fait que pour traiter *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)*, au pire des cas, il faut considérer tous les sous-ensembles de vues, formés d'au plus $n * (l + 1)$ vues, prises parmi les m vues de $\mathcal{V}$. La complexité est donnée par le théorème suivant :

Théorème 4.2.2
Soient n le nombre d'atomes dans Q, l la cardinalité du plus grand des ensembles de valeurs apparaissant dans les vues ou la requête et m le nombre de vues dans $\mathcal{V}$.

Il existe un algorithme pour résoudre les problèmes conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$) et rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$), dont la complexité en temps est en $O(m^{n*(l+1)})$.

Plus précisément, cette complexité est une borne supérieure de la complexité réelle donnée par la formule suivante :

$$2^m - \sum_{i=n*(l+1)+1}^{m} C_m^i.$$

On ne considère dans l'ensemble des parties de $\mathcal{V}$, que les conjonctions de vues d'au plus $n * (l + 1)$ vues. Autrement dit, l'espace de recherche est 2^m auquel on retire tous les sous-ensembles de taille supérieure à $n * (l + 1)$.

4.2.2 Cas du langage $\mathcal{ALN}(\mathcal{O}_v)$

Comme effectué pour le langage $\mathcal{FL}_0(\mathcal{O}_v)$, nous caractérisons dans cette section, le problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)*. La forme des réécritures dans ce cas est donnée par le lemme suivant.

Lemme 4.2.2
Soit le problème conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$) où $Q \equiv \forall w.P$. Soient l la cardinalité du plus grand des ensembles de valeurs qui apparaît dans les vues ou la requête, et p la profondeur maximale[17] *des atomes apparaissant dans les requêtes et les vues. Si $Q' \equiv V_{i_1} \sqcap \ldots \sqcap V_{i_k}$ où $\{V_{i_1}, \ldots V_{i_k}\}$ est un sous-ensemble **minimal** de $\mathcal{V}$ tel que $w \in V_{Q'}(P) \cup E(Q')$ alors une des conditions suivantes est vérifiée :*

[17] la profondeur d'un atome $\forall w.P$ est égal à la longueur du mot w, i.e. le nombre de rôles dans w.

a) $k = 1$,
  a.1) $P \in \{A, \neg A\}$ *et* $\forall w.P \in V_{i_1}$ *ou,*
  a.2) $P = (\geq nR)$ *et* $\forall w.(\geq pR) \in V_{i_1}$ *avec* $p \geq n$ *ou,*
  a.3) $P = (\leq nR)$ *et* $\forall w.(\leq pR) \in V_{i_1}$ *avec* $p \leq n$
b) $1 \leq k \leq l + 1$,
  b.1) $P = E$ *et* $\{V_{i_1}...V_{i_k}\}$ *est tel que : pour* $j \in [1, k]$, $\forall w.E_{i_j} \in V_{i_j}$, *et* $\cap_{j=i_1}^{i_k} E_j \subseteq E$.
  b.2) $P = (\leq n\, R_v)$, *avec* $R_v \in \mathcal{R}_v$, $\{V_{i_1}...V_{i_k}\}$ *est tel que : pour* $j \in [1, k]$, $\forall w.E_{i_j} \in V_{i_j}$, *et* $|\cap_{j=i_1}^{i_k} E_j| \leq n$.
c) $1 \leq k \leq l + p$ *et il existe un préfixe* $w'v$ *de* w *tel que* $\forall w'.(\leq 0v) \in \sqcap_{j=i_1}^{i_k} V_j$.

La preuve de ce lemme est donnée en annexe A.6, page 129. Notons que ce lemme, comme le lemme 4.2.1, peut se reformuler en remplaçant la prémisse par «si Q' est une réécriture conjonctive maximale de Q alors une des conditions (a),(b) ou (c) est vérifiée». En effet, d'après le lemme 4.1.4, si Q' est formée d'un sous-ensemble minimal de $\mathcal{V}$ tel que $w \in V_{Q'}(P) \cup E(Q')$, alors Q' est une réécriture conjonctive maximale de Q. De plus, ce lemme comme le lemme 4.2.1, donne uniquement les conditions nécessaires que doit vérifier une réécriture conjonctive maximale d'une requête, comme illustré par l'exemple 23 qui suit.

Exemple 23
Soit $Q \equiv \forall v.u.\{1, 6, 7\}$.
Considérons les vues V_1, V_2 et V_3 suivantes :
– $V_1 \equiv \overline{V_1} \sqcap \forall v.A \sqcap \forall v.u.\{1, 2, 3\}$,
– $V_2 \equiv \overline{V_2} \sqcap \forall v.\neg A \sqcap \forall v.u.\{1, 2, 4\}$,
– $V_3 \equiv \overline{V_3} \sqcap \forall v.u.\{1, 3, 4\}$.
Alors d'après le cas (b.1) du lemme 4.2.2, $V_1 \sqcap V_2 \sqcap V_3$ appartient au panier $B(v.u, \{1, 6, 7\})$ car $V_1 \sqcap V_2 \sqcap V_3 \sqsubseteq \forall v.u.\{1\}$. De plus, d'après le cas (c) du lemme 4.2.2, $V_1 \sqcap V_2$ appartient au panier $B(v.u, \{1, 6, 7\})$ car $V_1 \sqcap V_2 \sqsubseteq \forall v.\bot$.
Ainsi $V_1 \sqcap V_2 \sqcap V_3$ n'est pas une réécriture conjonctive maximale de Q. La seule réécriture conjonctive maximale de Q est alors $V_1 \sqcap V_2$. ◦

Analysons maintenant les résultats du lemme 4.2.2. Dans $\mathcal{ALN}(\mathcal{O}_v)$, les réécritures d'un panier d'un atome d'une requête donnée peuvent être formées d'une seule vue (cas (a)), ou d'au plus $l + 1$ vues (cas (b)), ou d'au plus $l + p$ vues (cas (c)). Evidemment, les cas des éléments constitués de plusieurs vues (cas (b) et (c)) sont plus complexes que dans $\mathcal{FL}_0(\mathcal{O}_v)$. Le cas (b.1) du à la présence du constructeur $\mathcal{O}_v$ est similaire au cas (b) du lemme 4.2.1 caractérisant les réécritures dans $\mathcal{FL}_0(\mathcal{O}_v)$. Le cas (b.2) est du à l'intéraction entre le constructeur $\mathcal{O}_v$ et le constructeur de cardinalité ($\leq nr$). Nous illustrons ce dernier cas dans l'exemple ci-dessous.

Exemple 24
Soit une requête Q telle que $Q \equiv \forall arecu. \leq 3typeProduit.$
Supposons qu'il existe 3 vues $V_i, i \in \{5, 6, 7\}$ telles que $V_i \equiv \overline{V_i} \sqcap \forall arecu.typeProduit.E_i$ avec
$E_5 = \{P_1, P_{10}, P_{15}, P_{20}, P_{27}\}$
$E_6 = \{P_1, P_{10}, P_{15}, P_{20}, P_{26}\}$
$E_7 = \{P_1, P_{10}, P_{15}, P_{26}, P_{27}\}$.

Soit $Q' \equiv V_5 \sqcap V_6 \sqcap V_7$. On peut constater que $Q' \sqsubseteq \forall arecu.typeProduit.\{P_1, P_{10}, P_{15}\}$ et ainsi Q' est subsumé par $\forall arecu. \leq 3typeProduit$. D'où, dans cet exemple, Q' est une réécriture conjonctive maximale de Q. ◦

Dans l'exemple précédent, le nombre de vues apparaissant dans Q' est borné par la cardinalité maximale des ensembles de valeurs, soit 4. Le pire des cas se présente lorsque l'atome à réécrire est de la forme $\leq 0R$ et comme pour le cas (b) du lemme 4.2.1, on cherche le nombre de vues à intersecter, nécessaires pour obtenir l'ensemble vide comme restriction d'un rôle.

Enfin le cas (c) du lemme 4.2.2 est lié à la notion d'*inconsistance implicite* dans $\mathcal{ALN}(\mathcal{O}_v)$. La notion d'inconsistance implicite dénote l'apparition du concept $\bot$ dans la restriction d'un rôle donné (e.g. $\forall w.\bot$, avec $w \neq \epsilon$). Elle induit des cas de subsomption de la forme suivante :

$$\forall v.\bot \sqsubseteq \forall vu.P \ (i.e. \ \leq 0v \sqsubseteq \forall vu.P).$$

Ce cas correspond au cas (2) du théorème 3.4.1 de la subsomption dans $\mathcal{ALN}(\mathcal{O}_v)$. Etant donnée une requête $Q \equiv \forall w.P$, les conjonctions de vues subsumées par $\forall v.\bot$ avec v préfixe de w, sont des réécritures de Q. Ainsi le cas (c) est déterminé par le nombre de vues nécessaires dans une conjonction, pour obtenir un concept de la forme $\forall w'.v.\bot$ (ou $\forall w'. \leq 0v$), i.e. pour obtenir une inconsistance implicite. Dans le cas du langage $\mathcal{ALN}$, une forme générale d'inconsistance implicite est donnée dans [53]. Nous donnons ci-desssous une illustration de cette forme en considérant les atomes suivants :

1) $\forall v_1.v_2.v_3.v_4.A$
2) $\forall v_1.v_2.v_3.v_4.\neg A$
3) $\forall v_1.v_2.v_3.v_4. \leq nR$
4) $\forall v_1.v_2.v_3.v_4. \geq mR$
5) $\forall v_1.v_2.v_3. \geq 1v_4$
6) $\forall v_1.v_2. \geq 1v_3$
7) $\forall v_1. \geq 1v_2$

La conjonction des atomes (1),(2),(5),(6) et (7) (resp. (3),(4),(5),(6) et (7)) est équivalente à $\forall v_1.\bot$. En effet, il faut 2 atomes pour générer l'inconsistance à une profondeur p donnée. La conjonction des atomes (1) et(2) (resp. (3) et (4)) génère l'inconsistance à la profondeur $p = 4$. Autrement dit elle permet d'obtenir le concept $\forall v_1.v_2.v_3.v_4.\bot$. Ensuite pour obtenir une inconsistance implicite à la profondeur $r < p$, il est nécessaire d'ajouter au moins $(p-r)$ atomes. L'inconsistance implicite $\forall v_1.\bot$ est obtenue en ajoutant les $(p - r = 3)$ atomes (5),(6),(7) à la conjonction des atomes (1) et (2) (resp. (3) et (4)).

Ainsi dans $\mathcal{ALN}$, le nombre maximal d'atomes nécessaires pour générer une description de la forme $\forall R.\bot$ dépend de la profondeur maximale des atomes utilisés. Soit p cette profondeur, alors le nombre maximum d'atomes nécessaires est $(p - 1) + 2 = p + 1$. Cependant dans le langage $\mathcal{ALN}(\mathcal{O}_v)$, cette borne est augmentée compte tenu de la présence du constructeur $\mathcal{O}_v$. En effet alors que dans $\mathcal{ALN}$, la seule possibilité pour générer le concept $\bot$ est la conjonction d'au plus deux atomes (e.g. $A \sqcap \neg A$ ou $(\leq nR) \sqcap (\geq mR)$, avec $n < m$), dans $\mathcal{ALN}(\mathcal{O}_v)$, le concept $\bot$, i.e. $\emptyset$, peut être induit par une conjonction d'au plus $l + 1$ atomes (comme discuté dans le cas (b) du lemme 4.2.1). De cette façon, le nombre maximal d'atomes (de vues) nécessaires pour générer des inconsistances implicites dans $\mathcal{ALN}(\mathcal{O}_v)$, est obtenu en aditionnant les deux bornes $l+1$ et $p-1$, soit $l+p$. Ce cas est illustré dans l'exemple suivant.

Exemple 25
Soit $Q \equiv \forall v_1.P$.
Soient les atomes suivants :
1) $\forall v_1.v_2.v_3.v_4.\{1,2,3\}$
2) $\forall v_1.v_2.v_3.v_4.\{1,2,4\}$
3) $\forall v_1.v_2.v_3.v_4.\{1,3,4\}$
4) $\forall v_1.v_2.v_3.v_4.\{2,3,4\}$
5) $\forall v_1.v_2.v_3. \geq 1 v_4$
6) $\forall v_1.v_2. \geq 1 v_3$
7) $\forall v_1. \geq 1 v_2$
La cardinalité maximale des contraintes de valeurs est $l = 3$. La profondeur maximale des atomes est $p = 4$, car le mot le plus long est $v_1.v_2.v_3.v_4$ et le nombre de rôles dans ce mot est 4. La conjonction des $(l + 1 = 4)$ atomes (1), (2), (3), (4) conduit à l'inconsistance implicite $\forall v_1.v_2.v_3.v_4.\bot$. La conjonction de ces $(l + 1)$ atomes avec les $(p - 1 = 3)$ atomes (5), (6), (7) conduit à l'inconsistance implicite $\forall v_1.\bot$. Ainsi la conjonction de ces atomes est subsumée par $Q \equiv \forall v_1.P$. Au pire des cas $(p + l)$ atomes sont nécessaires pour générer l'inconsistance implicite permettant de réécrire un atome. ◦

Un autre résultat intéressant du lemme 4.2.2 est qu'il permet d'identifier précisément les formes des réécritures dans lesquelles le constructeur $\mathcal{O}_v$ intervient. En effet les ensembles de valeurs interviennent dans deux types de calculs

- le calcul des réécritures du cas (b.1)
- le calcul des réécritures du cas (b.2), et du cas (c).

Pour conclure cette section, nous donnons la taille maximale des réécritures conjonctives maximales et la complexité des problèmes *rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)* et *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)*.

Théorème 4.2.3
Soit le problème conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$). Soient l la cardinalité du plus grand des ensembles de valeurs apparaissant dans les vues ou la requête, n le nombre d'atomes dans Q et p la profondeur maximale des atomes des vues ou de la requête.

Les réécritures conjonctives maximales de Q en termes des vues $\mathcal{V}$, dans $\mathcal{ALN}(\mathcal{O}_v)$, contiennent au plus $n * (l + p)$ vues de $\mathcal{V}$.

Cette borne supérieure de la taille des réécritures conjonctives maximales d'une requête Q est dérivée de la taille maximale des réécritures conjonctives maximales d'un atome (cas (c) du lemme 4.2.2). On déduit à partir du théorème 4.2.3, qu'il existe un algorithme pour calculer les solutions des problèmes *rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)* et *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)*, dont la complexité est fixée par le théorème suivant.

Théorème 4.2.4
Soient m le nombre de vues dans $\mathcal{V}$, n le nombre d'atomes dans Q, l la cardinalité du plus grand des ensembles de valeurs apparaissant dans les vues ou la requête, et p la profondeur maximale des atomes des vues ou de la requête.

Il existe un algorithme pour résoudre les problèmes conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$) et rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$), dont la complexité en temps est en $O(m^{n*(l+p)})$.

Ce résultat découle du fait que pour traiter un problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)*, d'après le théorème 4.2.3, nous avons uniquement besoin de considérer des sous-ensembles de vues, formés d'au plus $n * (l + p)$ vues, prises parmi les m vues de $\mathcal{V}$. Comme pour le théorème 4.2.2, notons que la complexité réelle est donnée par la formule suivante :

$$2^m - \sum_{i=n*(l+p)+1}^{m} C_m^i.$$

Cette section a permis de caractériser les formes des réécritures dans $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$ respectivement. En s'appuyant sur ces résultats, la section qui suit donne les principales étapes pour calculer le panier d'un atome d'une requête quelconque.

4.3 Calcul des paniers d'une requête

Cette section donne le principe de l'algorithme 2, appelé Construire_Panier, qui calcule les réécritures conjonctives d'un atome exprimé dans la logique $\mathcal{ALN}(\mathcal{O}_v)$.

L'algorithme Construire_Panier, repose sur le lemme 4.2.2. Il renvoit pour un atome quelconque d'une requête Q, ses réécritures conjonctives, telles que définies par le lemme 4.2.2. Les lignes 3-16 de Construire_Panier sont consacrées au traitement du cas (a) du lemme 4.2.2, i.e. au calcul des réécriture de taille 1. Les lignes 17-25 permettent de calculer les réécritures du cas (b.1). Plus précisément, cette partie fait d'abord appel à la fonction F_Vues_E qui isole un sous-ensemble $Vues$ de vues de $\mathcal{V}$ qui vérifient les propriétés du cas (b.1), i.e. qui sont contraintes par un ensemble de valeurs via le mot w_i. Cette partie fait ensuite appel à la fonction F_Min_Vues_subsume_E qui permet de calculer les sous-ensembles minimaux $SolVues$ de $Vues$ dont la conjonction des éléments est subsumée par un atome $\forall w_i.E$. Les sous-ensembles $SolVues$ sont ensuite ajoutés au panier de l'atome en cours de traitement. Les lignes 26-33 permettent de calculer les réécritures du cas (b.2). Cette partie fait appel à la fonction F_Vues_E pour obtenir l'ensemble de vues $Vues$, i.e. les vues de $\mathcal{V}$ qui sont contraintes par un ensemble de valeurs via le mot $w_i.R$. Cette partie fait ensuite appel à la fonction F_Min_Vues_subsume_N qui permet de calculer les sous-ensembles minimaux $SolVues$ de $Vues$ dont la conjonction des éléments est subsumée par un atome $\forall w_i.R. \leq nR$. Les sous-ensembles $SolVues$ sont ajoutés au panier de l'atome $\forall w_i.R. \leq nR$. Enfin les lignes 34-37 permettent de calculer les réécritures du cas (c), i.e. les vues menant à des inconsistances implicites de la forme $\forall v_i.\perp$ où v_i est un préfixe de w_i, grâce à la fonction F_Inconsis_Implic. Cet algorithme se réduit trivialement à un algorithme de construction des paniers dans $\mathcal{FL}_0(\mathcal{O}_v)$ en considérant uniquement les cas (a) et (b) et plus précisément les sous-parties correspondant aux lignes 1-8 et 1-24.

Dans le cadre de l'algorithme général CalculReec, présenté page 69, il est nécessaire après la phase de calcul des paniers, de supprimer des paniers, les réécritures insatisfiables et non maximales. En effet la conjonction de deux vues réécrivant un atome, peut mener à une inconsistance. De plus, l'algorithme Construire_Panier s'appuie sur le lemme 4.2.2 et comme discuté auparavant, ce lemme ne donne que les conditions nécessaires que doivent vérifier les réécritures conjonctives maximales. Aussi un traitement des paniers, a posteriori, est nécessaire pour supprimer les réécritures non maximales. Enfin après le produit cartésien, une nouvelle phase de suppression des réécritures insatisfiables et non maximales, permet de n'obtenir en sortie que les réécritures maximales de Q.

Algorithme 2 Construire_Panier

Entrée(s) : $\forall w_i.P_i$, $\mathcal{V}$
Sortie(s) : $B(w_i, P_i)$

/* Initialisation du panier */
$B(w_i, P_i)$:=$\emptyset$
/* Cas (a) du lemme 4.2.2 */
pour chaque $V \in \mathcal{V}$ **faire**
  si $\forall w_i.P \in V$ **alors**
    si $P = P_i$ **alors**
      $B(w_i, P_i) := B(w_i, P_i) \cup \{V\}$
    fin si
    si $(P =\leq nR)$ et $P_i = (\leq mR)$ et $n \leq m$ **alors**
      $B(w_i, P_i) := B(w_i, P_i) \cup \{V\}$
    fin si
    si $P = (\geq nR)$ et $P_i = (\geq mR)$ et $n \geq m$ **alors**
      $B(w_i, P_i) := B(w_i, P_i) \cup \{V\}$
    fin si
  fin si
fin pour
/* Cas (b) du lemme 4.2.2 */
/* Cas (b.1) */
si $P_i = E$ **alors**
  /* Recherche des vues V telles que $\forall w_i.E_i \in V$ */
  $Vues$:=F_Vues_E(w_i,$\mathcal{V}$)
  /* Recherche des sous-ensembles minimaux de $\mathcal{V}$ dont la conjonction est subsumée par $\forall w_i.E$ */
  $SolVues$:=F_Min_Vues_subsume_E($Vues$,w_i,E,$\mathcal{V}$)
  $B(w_i, P_i) := B(w_i, P_i) \cup SolVues$
fin si
/* Cas (b.2) */
si $P_i = (\leq nR)$ et $R \in \mathcal{R}_V$ **alors**
  /* Recherche des vues V telles que $\forall w_i.R.E_i \in V$ */
  $Vues$:=F_Vues_E($w_i.R$,$\mathcal{V}$)
  /* Recherche des sous-ensembles minimaux de $\mathcal{V}$ dont la conjonction est subsumée par $\forall w_i. \leq nR$ */
  $SolVues$:=F_Min_Vues_subsume_N($Vues$,$w_i.R$,n,$\mathcal{V}$)
  $B(w_i, P_i) := B(w_i, P_i) \cup SolVues$
fin si
/* Cas (c) du lemme 4.2.2 */
/* Recherche des conjonctions de vues V menant à des inconsistances implicites sur des préfixes de w_i */
$SolVues$:=F_Inconsis_Implic(w_i,$\mathcal{V}$)
$B(w_i, P_i) := B(w_i, P_i) \cup SolVues$
RETURN $B(w_i, P_i)$

Après avoir introduit l'algorithme Construire_Panier qui constitue l'étape principale de l'algorithme CalculReec, nous nous intéressons maintenant à la correction de l'algorithme général CalculReec.

Propriété 4.3.1
L'algorithme CalculReec de calcul des réécritures conjonctives maximales de Q est correct.

Pour démontrer la correction de cet algorithme, il est nécessaire de montrer qu'il est complet, juste et qu'il se termine. Informellement, la démonstration de la complétude de l'algorithme CalculReec est la suivante. Il est trivial que tous les éléments calculés sont des réécritures conjonctives de Q. La démonstration de la complétude consiste alors à prouver que les réécritures calculées contiennent toutes les réécritures conjonctives maximales de Q. Pour cela, on montre que Q' peut se décomposer en sous-ensembles minimaux de vues dont la conjonction de chacun d'entre-eux est subsumée par un atome de Q. Ainsi ces sous-ensembles minimaux de vues sont calculés dans la phase de construction des paniers et Q' est générée via la phase du produit cartésien. On est donc certain que toutes les réécritures conjonctives maximales de Q sont dans la sortie du produit cartésien de CalculReec. Les réécritures non maximales sont ensuite supprimées du produit cartésien. Aussi seules les réécritures conjonctives maximales sont retournées par CalculReec. Ainsi l'algorithme CalculReec est juste. La terminaison de l'algorithme CalculReec est relativement simple à démontrer. En effet, les boucles qui apparaissent dans les algorithmes CalculReec et Construire_Panier, font au pire des parcours exhaustifs d'ensembles de taille exponentiel mais toujours finis. Le détail de la preuve du théorème 4.3.1 est donné en annexe A.7, page 131. La preuve de la justesse de l'algorithme n'est pas détaillée puisque triviale.

L'étape clé de l'algorithme *CalculReec* repose sur celle de la construction des paniers. Lors de la construction des paniers, on calcule les cas classiques de réécriture dans le langage $\mathcal{ALN}$ et les cas de réécritures liés à la présence des contraintes de valeurs.

Dans le chapitre qui suit, nous nous concentrons sur les réécritures de requêtes engendrées par la présence des contraintes de valeurs car ces problèmes posent de nouvelles difficultés en termes de réécriture de requête. Puis nous montrons comment les techniques de fouilles de données participent à la résolution du problème de la réécriture de requêtes en termes de vues, en présence de ces contraintes.

Chapitre 5

Techniques de fouille de données pour la réécriture

Ce chapitre se concentre sur les cas de réécritures engendrés par la présence des contraintes de valeurs car ils posent de nouveaux problèmes dans le cadre de la réécriture. Nous montrons comment le problème de calculer ces réécritures peut se formuler dans un cadre ensembliste. Nous exploitons cette formulation ensembliste pour ramener ce problème au cadre de découverte des connaissances proposé dans [57]. Dans un premier temps, nous motivons le choix de l'utilisation de ce cadre. Il permet notamment d'adapter des algorithmes existants, développés dans le cadre de la découverte des connaissances, pour calculer effectivement les réécritures liées à la présence de $\mathcal{O}_v$. Ensuite nous présentons ce cadre théorique de découverte des connaissances, puis nous montrons comment le calcul des réécritures engendrées par la présence des contraintes de valeurs, se formalise dans ce cadre.

5.1 Vers une formulation ensembliste de la réécriture

Dans cette section, nous focalisons sur les problèmes du calcul des nouveaux cas de réécritures engendrés par le constructeur $\mathcal{O}_v$ (soient les cas (b.1) et (b.2) de réécritures spécifiés par le lemme 4.2.2 du chapitre précédent).

Soit un problème *rewrite(Q,$\mathcal{V}$,L)* avec $\mathcal{L} \in \{\mathcal{FL}_0(\mathcal{O}_v), \mathcal{ALN}(\mathcal{O}_v)\}$ et $Q \equiv \forall w.P$. Considérons maintenant le problème de la création du panier $B(w, P)$. Nous nous intéressons aux problèmes du calcul des réécritures des cas (b.1) et (b.2) du lemme 4.2.2. Pour définir plus précisément ces problèmes, nous introduisons l'ensemble $\mathcal{V}_w = \{V_1, ..., V_p\}$ qui désigne le sous-ensemble de vues de $\mathcal{V}$ telles que $\forall i \in \{1, ..., p\}, \exists E_i \mid (\forall w.E_i) \in V_i$.

Le problème *E_conj_rewrite(E,w)* correspond au calcul des réécritures de type (b.1). Il est défini comme suit :

Problème 3 *Soit un problème rewrite(Q,$\mathcal{V}$,L) avec $\mathcal{L} \in \{\mathcal{FL}_0(\mathcal{O}_v), \mathcal{ALN}(\mathcal{O}_v)\}$ et $Q \equiv \forall w.E$.*

Soit $\mathcal{V}_w \subseteq \mathcal{V}$.

Le problème E_conj_rewrite(E,w) consiste à calculer les plus petites conjonctions de vues de $\mathcal{V}_w$ subsumées par $\forall w.E$.

Le problème *N_conj_rewrite(n,wR)* correspond au calcul des réécritures de type (b.2). Il est defini comme suit :

Problème 4 *Soit un problème rewrite(Q,V,L) avec* $\mathcal{L} = \mathcal{ALN}(\mathcal{O}_v)$ *et* $Q \equiv \forall w. \leq n\ R$.
Soit $\mathcal{V}_{wR} \subseteq \mathcal{V}$.
Le problème N_conj_rewrite(n,wR) consiste à calculer les plus petites conjonctions de vues de $\mathcal{V}_{wR}$ *subsumées par* $\forall w. \leq n\ R$.

Pour un mot w, un ensemble E, un entier n donnés, nous cherchons maintenant à donner une formulation ensembliste des problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,w)*, pour les ramener par la suite, à un cadre de découverte des connaissances.

Pour reformuler plus précisément *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,w)*, nous introduisons quelques notations :

- $\mathtt{F}_w = \{E_1, ..., E_p\}$: l'ensemble des E_i associés aux vues V_i de $\mathcal{V}_w$.
- $S_1(w, E)$: l'ensemble de tous les sous-ensembles minimaux MS de $\mathtt{F}_w$ tel que l'intersection des éléments de MS est contenue dans E.
- $S_2(w, n)$: l'ensemble de tous les sous-ensembles minimaux MS de $\mathtt{F}_w$ tel que la cardinalité de l'intersection des éléments de MS est inférieure ou égale à n.

Nous allons à présent, montrer que les solutions de *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,w)* se caractérisent par l'intermédiaire des ensembles $S_1(w, E)$ et $S_2(w, n)$. De manière plus générale, les ensembles $S_1(w, E)$ et $S_2(w, n)$ sont formellement définis comme suit.

Définition 5.1.1
Soient w *un mot,* E *un ensemble et* n *un entier. Soit* $\mathtt{F}_w$ *un ensemble.* $\mathcal{P}(\mathtt{F}_w)$ *est l'ensemble des parties de* $\mathtt{F}_w$.
$S_1(w, E) = \{X \in \mathcal{P}(\mathtt{F}_w) |\ \cap\{x \in X\} \subseteq E$ *et* $\forall Y \subset X, \cap\{y \in Y\} \not\subseteq E\}$
$S_2(w, n) = \{X \in \mathcal{P}(\mathtt{F}_w) |\ |\cap\{x \in X\}| \leq n$ *et* $\forall Y \subset X,\ |\cap\{x \in X\}| > n\}$

$S_1(w, E)$ caractérise les plus petits sous-ensembles de $\mathtt{F}_w$ dont l'intersection des éléments est contenue dans E, tandis que $S_2(w, n)$ caractérise les plus petits sous-ensembles de $\mathtt{F}_w$ dont la cardinalité de l'intersection des éléments est inférieure à n. L'exemple suivant illustre les différentes définitions et notions introduites ci-dessus.

Exemple 26
Supposons qu'il existe 4 vues $V_i, i \in \{1, ..., 4\}$ telles que $V_i \sqsubseteq \forall numDepartement.E_i$ où
$E_1 = \{23, 15, 18, 80, 43, 03\}$
$E_2 = \{03, 63\}$
$E_3 = \{01, 07, 11, 43, 63\}$
$E_4 = \{26, 63\}$,
et 3 vues $V_i, i \in \{5, 6, 7\}$ telles que $V_i \sqsubseteq \forall arecu.typeProduit.E_i$ pour $i \in \{5, 6, 7\}$ où
$E_5 = \{P_1, P_{10}, P_{15}, P_{20}, P_{27}\}$
$E_6 = \{P_1, P_{10}, P_{15}, P_{20}, P_{26}\}$
$E_7 = \{P_1, P_{10}, P_{15}, P_{26}, P_{27}\}$.
On a alors $\mathcal{V}_{numDepartement} = \{V_1, V_2, V_3, V_4\}$, $F_{numDepartement} = \{E_1, E_2, E_3, E_4\}$,
$\mathcal{V}_{arecu.typeProduit} = \{V_5, V_6, V_7\}$ et $F_{arecu.typeProduit} = \{E_5, E_6, E_7\}$.

On constate que les plus petits sous-ensembles de $F_{numDepartement}$ dont l'intersection des éléments est contenue dans E, sont les suivants :
– $E_2 \subseteq E$,
– $E_1 \cap E_3 \subseteq E$,
– $E_3 \cap E_4 \subseteq E$ et
– $E_1 \cap E_4 \subseteq E$.
D'où $S_1(numDepartement, E) = \{\{E_2\}, \{E_1, E_3\}, \{E_1, E_4\}, \{E_3, E_4\}\}$.
On constate que le seul sous-ensemble de $F_{arecu.typeProduit}$ dont la cardinalité de l'intersection des éléments est inférieure à 3 est l'ensemble $\{E_5, E_6, E_7\}$. En effet, l'intersection des éléments de $\{E_5, E_6, E_7\}$ donne $\{P_1, P_{10}, P_{15}\}$ dont la cardinalité est égale à 3.
D'où $S_2(arecu.typeProduit, 3) = \{\{E_5, E_6, E_7\}\}$. ◦

Le lemme suivant caractérise les solutions de *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,wR)* avec les ensembles $S_1(w, E)$ et $S_2(wR, n)$.

Lemme 5.1.1
Soient n un entier, w et wR des mots et E un ensemble.
Soient deux problèmes E_conj_rewrite(E,w) et N_conj_rewrite(n,wR).
Soit $b = \sqcap_{j=i_1}^{i_k} V_j$ avec $k \geq 1$.
1. *b est une solution de E_conj_rewrite(E,w)*
   ssi $\{V_{i_1}, ..., V_{i_k}\} \subseteq \mathcal{V}_w$ et $\{E_{i_1}, ..., E_{i_k}\} \in S_1(w, E)$
2. *b est une solution de N_conj_rewrite(n,wR)*
   ssi $\{V_{i_1}, ..., V_{i_k}\} \subseteq \mathcal{V}_{w.R}$ et $\{E_{i_1}, ..., E_{i_k}\} \in S_2(w.R, n)$

La démonstration de ce lemme est donnée en annexe A.8, page 131.

L'exemple qui suit illustre le processus de formulation ensembliste des problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,wR)*.

Exemple 27
Soit une requête Q telle que $Q \equiv \forall numDepartement.E \sqcap \forall arecu. \leq 3typeProduit$, avec $E = \{63, 43, 03\}$.
Cherchons les réécritures *E_conj_rewrite(E,numDepartement)* de l'atome $\forall numDepartement.E$ de Q.
Pour cela, on considère l'ensemble de vues $\mathcal{V}_{numDepartement} = \{V_1, V_2, V_3, V_4\}$ et l'ensemble $F_{numDepartement} = \{E_1, E_2, E_3, E_4\}$, définis dans l'exemple 26.
Pour décider si les vues $V_i, i \in \{1, ..., 4\}$, ou leur conjonction, appartiennent à *E_conj_rewrite(E,numDepartement)*, il suffit de calculer $S_1(numDepartement, E)$, i.e. de considérer les plus petits sous-ensembles de $F_{numDepartement}$ dont l'intersection des éléments est contenue dans E.
Dans l'exemple 26, on a vu que $S_1(numDepartement, E) = \{\{E_2\}, \{E_1, E_3\}, \{E_1, E_4\}, \{E_3, E_4\}\}$.
On en déduit alors que les conjonctions de vues suivantes :
– V_2,
– $V_1 \sqcap V_3$,
– $V_3 \sqcap V_4$ et
– $V_1 \sqcap V_4$

sont solutions de *E_conj_rewrite(E,numDepartement)* et qu'elles appartiennent au panier $B(numDepartement, E)$ de $\forall numDepartement.E$
Cherchons maintenant les réécritures *N_conj_rewrite(3,arecu.typeProduit)* de $\forall arecu.(\leq\ 3\, typeProduit)$.
Pour cela, on considère l'ensemble $\mathcal{V}_{arecu.typeProduit} = \{V_5, V_6, V_7\}$ et l'ensemble $F_{arecu.typeProduit} = \{E_5, E_6, E_7\}$, définis dans l'exemple 26.
Dans l'exemple 26, on a vu que $S_2(arecu.typeProduit, 3) = \{\{E_5, E_6, E_7\}\}$.
On en déduit alors que la conjonction de vues $V_5 \sqcap V_6 \sqcap V_7$ est une solution de *N_conj_rewrite(3,arecu.typeProduit)* et qu'elle appartient au panier $B(arecu, (\leq 3typeProduit))$ de $\forall arecu. \leq 3typeProduit$.

○

Après avoir montré comment le problème du calcul des réécritures dues à la présence des contraintes de valeurs peut se ramener à une formulation ensembliste, nous allons par la suite, montrer comment les problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,wR)* et plus précisément la caractérisation des ensembles $S_1(w, E)$ et $S_2(wR, n)$ peuvent s'énoncer dans un cadre de découverte des connaissances.

Dans la section qui suit, nous présentons tout d'abord, le domaine de la découverte de connaissances dans les bases de données.

5.2 La découverte de connaissances dans les bases de données

De nombreux domaines, comme par exemple celui de l'astronomie, du domaine médical ou du domaine commercial, ont collecté et accumulé des données au fil des années et disposent aujourd'hui de grands volumes de données difficilement exploitables à l'échelle humaine [26]. Par exemple, dans le domaine de l'astronomie, le nombre d'objets stockés dans les bases de données peut atteindre 10^9. Aussi il émerge de ces domaines, de part les enjeux économiques et médicaux qu'ils engendrent, une forte demande en théories et en outils, capables d'assister les humains dans la tâche d'extraction de connaissances utiles pour le domaine.

L'objectif du domaine de la découverte de connaissances dans les bases de données est de proposer des théories et des outils pour extraire et mieux comprendre la sémantique des bases de données. La découverte de connaissances est un processus qui peut se décomposer en trois grandes étapes :

- le pré-traitement des données qui vise à nettoyer, intégrer et filtrer les données venant de différentes bases,
- la fouille de données (data mining) qui consiste à extraire des motifs intéressants de la base de données,
- le post-traitement des résultats qui permet de sélectionner parmi les connaissances extraites, celles qui semblent pertinentes aux yeux des experts.

Dans ce mémoire, nous nous intéressons à la deuxième étape de ce processus. Cette étape de fouille de données sera exploitée par la suite dans le but de rattacher le problème du calcul des réécritures au problème de trouver les motifs potentiellement intéressants. Dans le domaine de la fouille de données, des techniques et des algorithmes, capables de traiter de grands volumes

de données et d'extraire de celles-ci des connaissances utiles, ont été proposés et éprouvés. De nombreux travaux dans ce domaine se sont notamment consacrés à la découverte des règles d'association [4], des dépendances fonctionnelles [49] ou des dépendances d'inclusion [58] d'une base de données.

Présentation d'un cadre de découverte de connaissances

Pour rattacher les problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,w)* à un cadre de découverte de connaissances, nous nous appuyons sur le cadre théorique introduit par Mannila et Toivonen dans [57]. Ce cadre peut être énoncé comme suit :

- Soit r une base de données.
- Soit $\mathcal{L}$ un langage pour exprimer des motifs.
- Soit P un prédicat pour exprimer si un motif X de $\mathcal{L}$ est un motif "intéressant" de r, i.e., si l'évaluation du motif X dans r est vraie, notée $P(X, r)$.

L'objectif est de trouver la théorie de r selon $\mathcal{L}$ et P, i.e., l'ensemble $\mathcal{T}h(r, \mathcal{L}, P)$=$\{\ \varphi \in \mathcal{L}\ |P(r, \varphi)$ est vrai$\}$ qui correspond à tous les motifs intéressants de r.

L'exemple suivant permet d'illustrer les notions de ce cadre de découverte de connaissances.

Exemple 28

Soit r une base de données (Figure 5.1).
La valeur $t(X) = 1$ (resp. $t(X) = 0$) pour un attribut X d'un tuple t de r indique la présence (resp. l'absence) de la propriété X pour t.
Le langage $\mathcal{L}$ est l'ensemble des parties de $\{A, B, C, D\}$.
Dans cet exemple, on considère le prédicat P "être fréquent" défini comme suit :
Soit $X \in \mathcal{L}$.
$P(r, X)$ est vrai si la proportion de lignes de r contenant X est supérieure à un seuil fixé par l'utilisateur. Considérons ce seuil égal à $1/2$.
On cherche les motifs fréquents de r, i.e. sous-ensembles X de $\mathcal{L}$ qui vérifient P.

r :

	A	B	C	D
1	1	1	1	1
2	0	1	1	0
3	1	1	1	0

TAB. 5.1 – La base de données r

Soit $X = \{A, B\}$. Alors $P(r, \{A, B\})$ est vrai car le couple $\{A, B\}$ apparaît dans 2 tuples sur 3. Ainsi la proportion de transactions contenant $\{A, B\}$ est égale à $2/3$, supérieure au seuil fixé à $1/2$.

○

Si de plus, une relation de spécialisation/généralisation entre les motifs de $\mathcal{L}$ existe et que le prédicat vérifie la propriété d'anti-monotonie par rapport à cette relation, alors on peut obtenir une représentation plus condensée de la théorie. Plus formellement, on obtient :

Soit $\preceq$ une relation de spécialisation/généralisation, i.e. un ordre partiel sur les motifs de $\mathcal{L}$. On dit que X généralise Y et que Y spécialise X quand $X \preceq Y$. Soit $\mathcal{S}$ un ensemble de motifs de $\mathcal{L}$ tel que si $\varphi \in \mathcal{S}$ et $\gamma \preceq \varphi$ alors $\gamma \in \mathcal{S}$. On peut alors décrire $\mathcal{S}$ en termes de bordures de $\mathcal{S}$. La bordure positive de $\mathcal{S}$, notée $\mathcal{B}d^+(\mathcal{S})$, est l'ensemble de tous les motifs de $\mathcal{S}$ tels que leurs généralisations sont dans $\mathcal{S}$ mais pas leurs spécialisations. La bordure négative de $\mathcal{S}$, notée $\mathcal{B}d^-(\mathcal{S})$, désigne les motifs de $\mathcal{L}$ qui ne sont pas dans S, mais dont la généralisation est dans $\mathcal{S}$. Les notions de bordures sont définies formellement comme suit :

$\mathcal{B}d^+(\mathcal{S})$ = $\{\varphi \in \mathcal{S} \mid$ pour tout $\theta \in \mathcal{L}$ avec $\varphi \preceq \theta$, on a $\theta \notin \mathcal{S}\}$

$\mathcal{B}d^-(\mathcal{S})$ = $\{\varphi \in \mathcal{L} \setminus \mathcal{S} \mid$ pour tout $\gamma \in \mathcal{L}$ avec $\gamma \preceq \varphi$, on a $\gamma \in \mathcal{S}\}$.

Un prédicat $Pred$ est dit anti-monotone selon $\preceq$, si $\forall X, Y \in \mathcal{L}$ tels que $X \preceq Y$, si $Pred(r, Y)$ est vrai alors $Pred(X, r)$ est vrai.

Le prédicat P de $\mathcal{T}h(r, \mathcal{L}, P)$ doit être anti-monotone afin d'assurer à la théorie $\mathcal{T}h(r, \mathcal{L}, P)$ les mêmes propriétés que l'ensemble $\mathcal{S}$. Ceci permet alors de définir les bordures $\mathcal{B}d^+(\mathcal{T}h(r, \mathcal{L}, P))$ et $\mathcal{B}d^-(\mathcal{T}h(r, \mathcal{L}, P))$ de la théorie $\mathcal{T}h(r, \mathcal{L}, P)$. La bordure positive $\mathcal{B}d^+(\mathcal{T}h(r, \mathcal{L}, P))$ de la théorie spécifie les motifs les plus spécifiques de la théorie qui vérifient le prédicat P tandis que la bordure négative $\mathcal{B}d^-(\mathcal{T}h(r, \mathcal{L}, P))$ dénote les motifs les plus généraux du langage qui ne vérifient pas le prédicat P. Ainsi, la théorie peut être décrite en termes de ses bordures.

L'exemple suivant permet d'illustrer les notions de théorie et de bordures.

Exemple 29

Poursuivons l'exemple 28.

Fixons l'ordre partiel $\preceq$ à l'inclusion ensembliste $\subseteq$.

Soient $X, Y \in \mathcal{L}$. Prenons $X = \{A, B, C\}$ et $Y = \{A, C\}$. On a $Y \preceq X$ si et seulement si $Y \subseteq X$. Or $\{A, C\} \subseteq \{A, B, C\}$ et par conséquent $Y \preceq X$. Notons que $\{A, C\}$ généralise $\{A, B, C\}$ tandis que $\{A, B, C\}$ spécialise $\{A, C\}$.

Pour notre exemple, la théorie $\mathcal{T}h(r, \mathcal{L}, P)$ est égale à l'ensemble $\{\emptyset, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$. On peut vérifier que les éléments de $\mathcal{T}h(r, \mathcal{L}, P)$ vérifient le prédicat P.

Illustrons maintenant la propriété d'anti-monotonie du prédicat P.

Soit $X = \{A, B, C\} \in \mathcal{T}h(r, \mathcal{L}, P)$. Pour tout sous-ensemble Y de $X = \{A, B, C\}$, on peut vérifier que $P(r, Y)$ est vrai. En effet, soit $Y = \{A, C\}$, on a bien $P(r, \{A, C\})$ est vrai puisque $\{A, C\}$ apparaît dans 2 transactions sur 3.

Le prédicat P étant anti-monotone, la théorie $\mathcal{T}h(r, \mathcal{L}, P)$ peut être caractérisée en termes de bordures.

Dans cet exemple, la bordure positive de $\mathcal{T}h(r, \mathcal{L}, P)$ est composée d'un seul élément $\{A, B, C\}$. Le motif $\{A, B, C\}$ est le motif le plus spécifique de $\mathcal{T}h(r, \mathcal{L}, P)$ qui vérifie le prédicat P. La bordure négative de $\mathcal{T}h(r, \mathcal{L}, P)$ est également composée d'un seul élément $\{D\}$. Le motif $\{D\}$ est le motif le plus général du langage qui ne vérifie pas le prédicat.

La figure 5.1 représente les éléments de $\mathcal{L}$ sous forme d'un treillis en marquant les éléments de la théorie et les deux bordures de la théorie. ◦

La section qui suit s'attache à montrer comment les problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,w)* exhibés en section 5.1 peuvent être formalisés dans le cadre précédemment introduit.

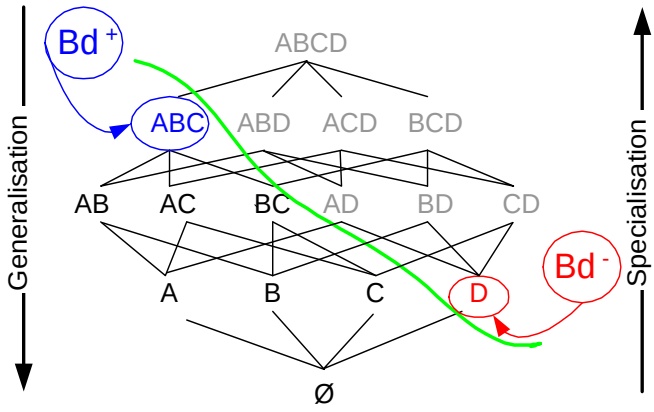

FIG. 5.1 – Théorie et bordures

5.3 Caractérisation du calcul des réécritures dans un cadre de découverte de connaissances

Soit $\mathcal{V}_w = \{V_1, ..., V_p\}$ un ensemble de vues de $\mathcal{V}$ qui contiennent dans leur description une restriction de valeur sur le mot w vers un ensemble de valeurs E_i.

Nous cherchons à résoudre les problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,w)* qui consistent à identifier les sous-ensembles minimaux de $\mathcal{V}_w$ dont la conjonction des éléments réécrit les atomes $\forall w.E$ et $\forall v.(\leq\ n\ R)$ respectivement, avec $w = vR$. D'après la section 5.1, étant donné $\mathtt{F}_w = \{E_1, ..., E_p\}$, l'ensemble des ensembles de valeurs E_i des vues de $\mathcal{V}_w$, leur résolution revient à calculer les deux ensembles $S_1(w, E)$ et $S_2(w, n)$.

Dans cette section, nous montrons comment le calcul de $S_1(w, E)$ et $S_2(w, n)$ peut être rattaché au cadre théorique de la section 5.2.

5.3.1 Identification de $S_1(w, E)$

Dans ce contexte, le premier ensemble $S_1(w, E)$ peut se ramener au cadre théorique précédent comme suit :

- la relation r est vide.
- le langage $\mathcal{L}_w$ est l'ensemble des parties de $\mathtt{F}_w$, i.e. $\mathcal{P}(\mathtt{F}_w)$.
- le prédicat, noté P_1, est défini de la façon suivante :
  Soient $X \in \mathcal{L}_w$, $X = \{E_1, ..., E_k\}$ et E un ensemble de valeurs
  $P_1(E, X)$ est vrai ssi $\cap_{i=1}^{k}\{E_i\} \not\subseteq E$.
- la relation d'ordre est la relation d'inclusion $\subseteq$.

La théorie $\mathcal{T}h(\emptyset, \mathcal{L}_w, P_1)$ est alors l'ensemble des sous-ensembles de $\mathtt{F}_w$ qui vérifient le prédicat P_1.

L'exemple qui suit illustre la formulation de $S_1(w, E)$ dans le cadre introduit ci-dessus.

Exemple 30

Poursuivons l'exemple 26 donné en chapitre 4.
Soit le mot $numDepartement$.

On a $\mathcal{V}_{numDepartement} = \{V_1, V_2, V_3, V_4\}$ et $F_{numDepartement} = \{E_1, E_2, E_3, E_4\}$.
La base de données est vide.
Le langage $\mathcal{L}_{numDepartement}$ est l'ensemble des parties de $F_{numDepartement}$.
D'où $\mathcal{L}_{numDepartement} = \{\emptyset, \{E_1\}, \{E_2\}, \{E_3\}, \{E_4\}, \{E_1, E_2\}, \{E_1, E_3\}, \{E_1, E_4\}, \{E_2, E_3\}, \{E_2, E_4\}, \{E_3, E_4\}, \{E_1, E_2, E_3, E_4\}\}$.
L'ensemble de valeurs E, paramètre du prédicat P_1, est fixé à $E = \{63, 43, 03\}$.
On remarque que :
– $P_1(E, \{E_1\})$ est vrai car $E_1 = \{23, 15, 18, 80, 43, 03\} \not\subseteq \{63, 43, 03\}$ et donc $E_1 \not\subseteq E$.
– $P_1(E, \{E_2\})$ est faux car $E_2 = \{03, 63\} \subseteq \{63, 43, 03\}$ et donc $E_2 \subseteq E$.
– $P_1(E, \{E_1, E_3\})$ est faux car $E_1 \cap E_3 = \{43\} \subseteq \{63, 43, 03\}$ et donc $E_1 \cap E_3 \subseteq E$.
La théorie $\mathcal{T}h(\emptyset, \mathcal{L}_{numDepartement}, P_1)$ est dans ce contexte, égale à $\{\{E_1\}, \{E_3\}, \{E_4\}\}$.
○

Afin de calculer la théorie $\mathcal{T}h(\emptyset, \mathcal{L}_{numDepartement}, P_1)$, il est nécessaire de s'assurer que le prédicat P_1 est anti-monotone. Ce résultat est établi par le lemme suivant.

Lemme 5.3.1
P_1 est anti-monotone par rapport à $\subseteq$.

La preuve du lemme 5.3.1 est donnée en annexe A.9, page 132.

Afin de simplifier les notations, on note par IEE (pour Intersection d'Ensembles non inclus dans E), la théorie $Th(\emptyset, \mathcal{L}_w, P_1)$. Le théorème suivant donne le principal résultat de cette section. Il caractérise $S_1(w, E)$ comme la bordure négative $\mathcal{B}d^-(IEE)$ de $Th(\emptyset, \mathcal{L}_w, P_1)$.

Théorème 5.3.1
Soit le problème E_conj_rewrite(E,w).
$S_1(w, E) = \mathcal{B}d^-(IEE)$

La preuve du théorème 5.3.1 est donnée en annexe A.9, page 132.

Exemple 31
Dans la suite de l'exemple 30, on a :
$\mathcal{B}d^+(IEE) = \{\{E_1\}, \{E_3\}, \{E_4\}\}$ et $\mathcal{B}d^-(IEE) = \{\{E_2\}, \{E_1, E_3\}, \{E_1, E_4\}, \{E_3, E_4\}\}$.
On en déduit que $S_1(numDepartement, E) = \{\{E_2\}, \{E_1, E_3\}, \{E_1, E_4\}, \{E_3, E_4\}\}$.
On retrouve bien le résultat de l'exemple 26. ○

5.3.2 Identification de $S_2(w, n)$

Pour un mot donné w, on cherche maintenant à caractériser $S_2(w, n)$ dans le cadre théorique précédent.

Dans ce contexte, la relation est l'ensemble vide, $\mathcal{L}_w$ consiste en l'ensemble des parties de $\mathtt{F}_w = \{E_1, ..., E_p\}$, la relation d'ordre reste l'inclusion et le prédicat $P_2(n, X)$ est défini comme suit :
Soit $X = \{E_1, ..., E_k\} \in \mathcal{L}_w$, $P_2(n, X)$ est vrai ssi $| \cap_{i=1}^{k} E_i | > n$.

Lemme 5.3.2
P_2 est anti-monotone selon $\subseteq$.

La preuve du lemme 5.3.2 est donnée en annexe A.9, page 132.

Soit IEN une notation pour $Th(\emptyset, \mathcal{L}_w, P_2)$ (IEN pour Intersection d'Ensembles dont la cardinalité est strictement supérieure à n). Le théorème suivant donne une caractérisation de $S_2(w, n)$ en termes de la bordure négative $\mathcal{B}d^-(IEN)$ de $Th(\emptyset, \mathcal{L}_w, P_2)$.

Théorème 5.3.2
Soit le problème N_conj_rewrite(n,w).
$S_2(w, n) = \mathcal{B}d^-(IEN)$

La preuve du théorème 5.3.2 est donnée en annexe A.9, page 132.

Exemple 32
Dans la suite de l'exemple 26, pour le mot $arecu.typeProduit$ on a :
$\mathcal{V}_{arecu.typeProduit} = \{V_5, V_6, V_7\}$ et $F_{arecu.typeProduit} = \{E_5, E_6, E_7\}$.
D'où $\mathcal{L}_{arecu.typeProduit} = \{\emptyset, \{E_5\}, \{E_6\}, \{E_7\}, \{E_5, E_6\}, \{E_5, E_7\}, \{E_6, E_7\}, \{E_5, E_6, E_7\}\}$.
Le paramètre n du prédicat P_2 est égal à 3.
On a :
– $P_2(3, \{E_5\})$ est vrai, car $\mid E_5 \mid > 3$.
– $P_2(3, \{E_6, E_7\})$ est vrai, car $\mid E_6 \cap E_7 \mid > 3$.
– $P_2(3, \{E_5, E_6, E_7\})$ est faux, car $\mid E_5 \cap E_6 \cap E_7 \mid \leq 3$.
La théorie $\mathcal{T}h(\emptyset, \mathcal{L}_{arecu.typeProduit}, P_2)$ est égale à $\{\{E_5\}, \{E_6\}, \{E_7\}, \{E_5, E_6\}, \{E_5, E_7\}, \{E_6, E_7\}\}$.
Les bordures de $\mathcal{T}h(\emptyset, \mathcal{L}_{arecu.typeProduit}, P_2)$ sont :
$\mathcal{B}d^+(IEN) = \{\{E_5, E_6\}, \{E_5, E_7\}, \{E_6, E_7\}\}$ et $\mathcal{B}d^-(IEN) = \{\{E_5, E_6, E_7\}\}$.
On en déduit que $S_2(arecu.typeProduit, 3) = \{\{E_5, E_6, E_7\}\}$. On retrouve le résultat de l'exemple 26.

○

Les ensembles $S_1(E, w) = \mathcal{B}d^-(IEE)$ et $S_2(n, w) = \mathcal{B}d^-(IEN)$ ainsi obtenus permettent de déduire les solutions des problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,w)*. Ainsi à partir des exemples 30, 31 et 32 où $S_1(numDepartement, E) = \{\{E_2\}, \{E_1, E_3\}, \{E_1, E_4\}, \{E_3, E_4\}\}$ et $S_2(arecu.typeProduit, 3) = \{\{E_5, E_6, E_7\}\}$, on sait que V_2, $V_1 \sqcap V_3$, $V_1 \sqcap V_4$ et $V_3 \sqcap V_4$ appartiennent à *E_conj_rewrite(E,numDepartement)* et donc au panier $B(numDepartement, E$ et que $V_5 \sqcap V_6 \sqcap V_7$ appartient à *N_conj_rewrite(3,arecu.typeProduit)* et donc au panier $B(arecu, (\leq 3typeProduit))$.

Nous notons finalement que des *algorithmes par niveau* ont été proposés pour calculer la bordure positive mais aussi négative [57, 41], et peuvent être réutilisés pour le calcul de $\mathcal{B}d^-(IEE)$ et de $\mathcal{B}d^-(IEN)$. Les algorithmes par niveau listent les motifs de $\mathcal{L}_w$ des plus généraux aux plus spécialisés, comme le montre la figure 5.2. Un niveau k est constitué des motifs formés de k éléments. A chaque niveau k, les motifs sont testés. S'ils vérifient le prédicat, les motifs sont sélectionnés pour peupler la théorie. S'ils ne vérifient pas le prédicat, ils sont éliminés et sélectionnés pour appartenir à la bordure négative.

Avec ce type d'algorithme, notons que comme illustré par la figure 5.2, dans l'exemple 31, l'espace de recherche du problème *E_conj_rewrite(E,w)* n'est pas parcouru entièrement.

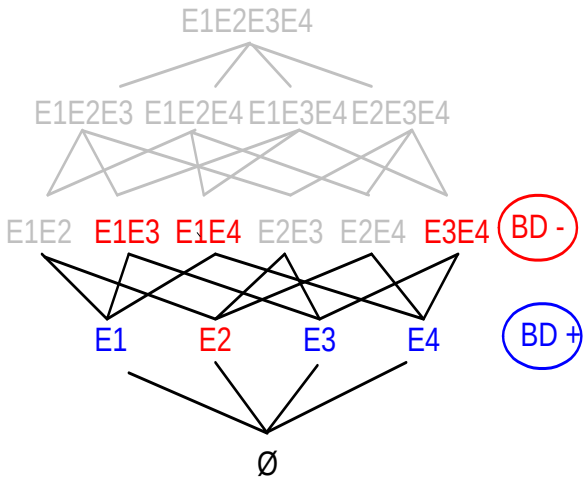

FIG. 5.2 – Théorie et bordures de l'exemple 31

En effet seuls les sous-ensembles $\{E_1\}$, $\{E_2\}$, $\{E_3\}$, $\{E_4\}$, $\{E_1, E_3\}$, $\{E_1, E_4\}$, $\{E_3, E_4\}$ de $\mathcal{L}_{numDepartement}$ sont testés pour calculer les bordures. Pour conclure, le problème *E_conj_rewrite(E,w)* est un problème difficile à résoudre. Cependant cette caractérisation du problème en termes de bordures permet d'envisager des solutions efficaces pour traiter le problème de la réécriture de requêtes.

Dans la partie suivante, nous tirons profit des ces algorithmes afin de mettre en oeuvre un prototype de réécriture dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$.

Quatrième partie

Mise en Oeuvre

Dans ce mémoire, le problème de la réécriture a été étudié pour deux logiques : $\mathcal{FL}_0(\mathcal{O}_v)$ et $\mathcal{ALN}(\mathcal{O}_v)$. Pour la mise en oeuvre, seuls les algorithmes de réécriture dans $\mathcal{FL}_0(\mathcal{O}_v)$ ont fait l'objet d'une implémentation. Cette partie est dédiée à la présentation d'un prototype de réécriture dans $\mathcal{FL}_0(\mathcal{O}_v)$ et de quelques expérimentations. Dans un premier temps, nous présentons l'architecture du moteur de réécriture dans $\mathcal{FL}_0(\mathcal{O}_v)$. Puis, nous focalisons sur la partie implémentant le calcul des réécritures engendrées par la présence des contraintes de valeurs, appelée *ComputeEConj*. Nous proposons alors deux versions de *ComputeEConj* : *ComputeEConj_1* et *ComputeEConj_2*, dont nous donnons quelques résultats expérimentaux.

Chapitre 6

Implémentation et expérimentations

6.1 Architecture du moteur de réécriture

Cette section présente les choix effectués pour l'implémentation du moteur de réécriture. Le moteur prend en entrée une requête et des vues sous forme de documents XML. Il donne en sortie des réécritures conjonctives maximales de Q d'une requête donnée.

L'architecture globale du moteur de réécriture est décrite en figure 6.1. Le moteur s'appuie

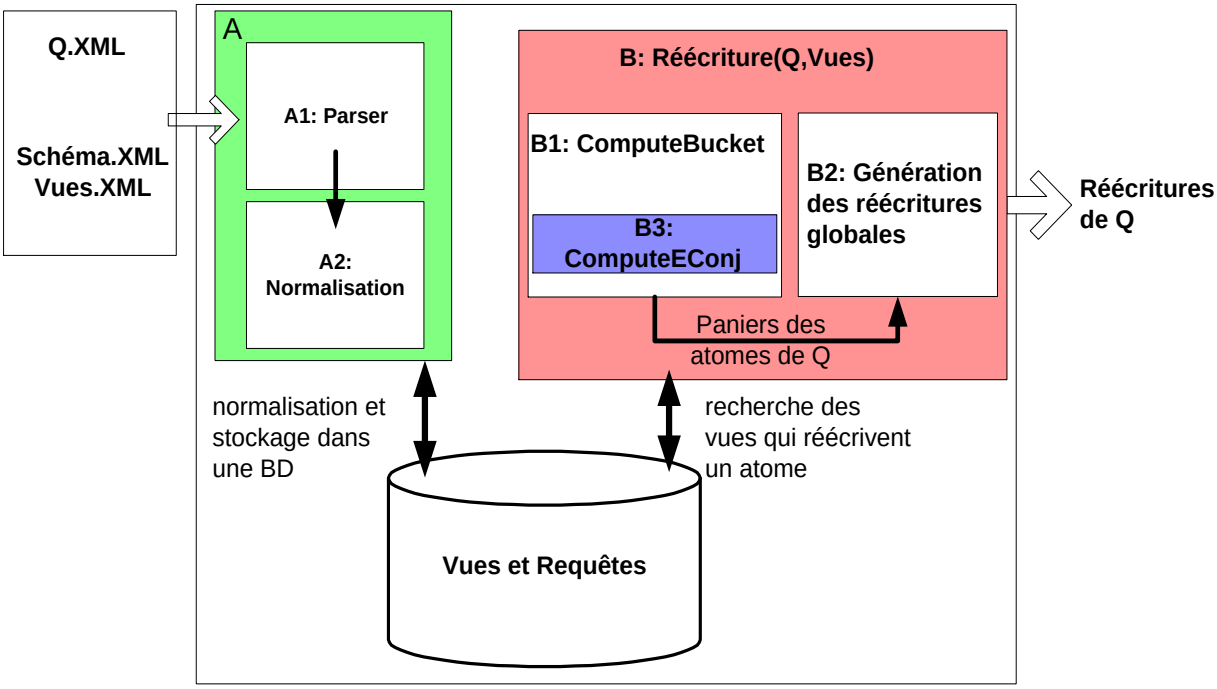

FIG. 6.1 – Description générale du moteur de réécriture

sur les composants suivants :

– une base de données pour stocker les requêtes et les vues. L'utilisation d'un système de gestion de bases de données (SGBD) est motivée par la capacité de ce type de système à

gérer efficacement de grands volumes de données et à traiter un grand nombre de transactions simultanées. Ainsi les SGBD se prêtent bien à la tâche d'un médiateur. Le SGBD choisi est Oracle.

– un analyseur de requêtes et de vues. L'analyseur permet d'analyser les requêtes et les vues présentées sous forme de documents XML, puis de les charger au sein du SGBD dans des sructures de données appropriées. Ensuite ces structures sont utilisées par un module de normalisation afin de transformer les vues et les requêtes dans leur forme normale. L'analyseur est un programme JAVA qui par le biais de la couche JDBC, va charger les vues et/ou les requêtes au sein du SGBD. Le module de normalisation est un programme PL/SQL.
– un module CalculReec pour calculer les réécritures des requêtes à partir des vues stockées dans le SGDB. Ce module renvoie les réécritures conjonctives maximales d'une requête donnée. Ce module est un programme PL/SQL.

La figure 6.2 donne le schéma conceptuel de la base de données sur laquelle s'appuie le moteur de réécriture, en suivant les conventions graphiques des diagrammes de classes UML. Plus précisément, il décrit les vues et les requêtes de la base de données comme suit. Les

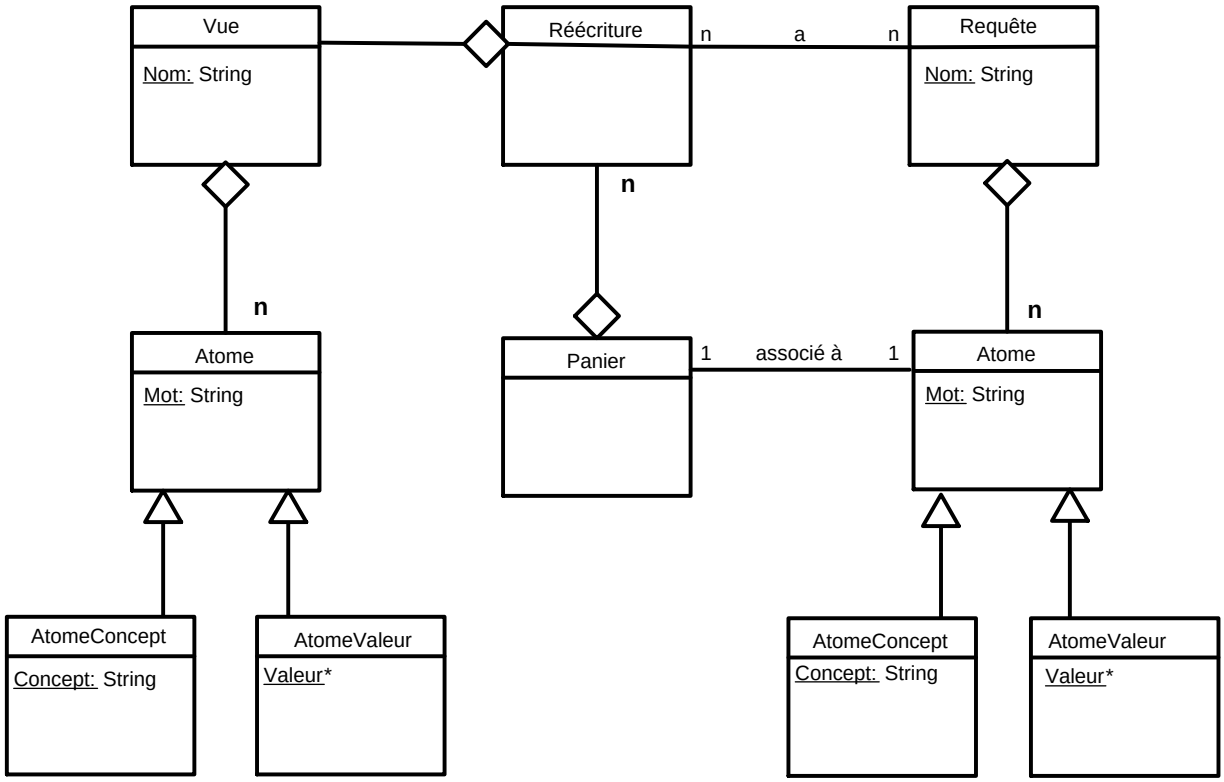

FIG. 6.2 – Schéma conceptuel UML de la base de données

vues et les requêtes sont composées d'atomes où chaque atome est un couple (mot,concept) ou (mot,valeur). Le couple (mot,concept)/(mot,valeur) capture alors la notion d'atome de concept normalisé telle que définie au chapitre 3. Une requête a plusieurs réécritures. Une réécriture est

composée de vues. Un atome d'une requête est associé à un seul panier qui est lui-même formé de réécritures. La base de données est détaillée en annexe B, section B.1.

Nous nous intéressons maintenant au module de calcul des réécritures.

6.2 Le module de calcul des réécritures

Le module CalculReec est composé de deux sous-modules :
- un sous-module pour calculer les paniers des atomes d'une requête Q. Ce sous-module est appelé *ComputeBucket*.
- un sous-module pour générer les réécritures globales de Q, à partir de ses paniers.

Nous nous concentrons dans cette partie sur le sous-module *ComputeBucket* et plus précisément sur le programme *ComputeEConj* utilisé par *ComputeBucket* lorsque l'atome à réécrire est de la forme $\forall mot.E$. Une version plus détaillée de *ComputeBucket* est donnée en annexe B, section B.2. *ComputeEConj* permet de calculer les solutions de *E_conj_rewrite(mot,E)*, i.e., les plus petites conjonctions de vues subsumées par $\forall mot.E$.

Le programme *ComputeEConj* est une adaptation d'un algorithme existant de découverte de connaissances. Plus précisément, il s'agit d'une adaptation d'un algorithme classique de génération des motifs fréquents d'une base de données, appelé APriori. Notre choix s'est porté sur cet algorithme car la bordure négative est également calculée sans traitement supplémentaire. L'implémentation d'APriori permet ainsi d'obtenir les solutions de *E_conj_rewrite(w,E)* qui ont été précédemment caractérisées comme étant la bordure négative d'une théorie donnée.

Le programme *ComputeEConj* prend en entrée les vues candidates, i.e., les vues qui contiennent dans leur description un atome de la forme $\forall mot.E_i$. Plus précisément, *ComputeEConj* prend en entrée les ensembles de valeurs associées aux vues candidates, i.e. F_{mot}. Il donne en sortie les plus petites conjonctions de vues subsumées par $\forall mot.E$.

La section qui suit présente les principales étapes d'APriori ainsi qu'une structure de données habituellement utilisée pour implémenter APriori.

6.2.1 APriori

Les principales étapes de l'algorithme APriori sont :
- la génération des motifs candidats à un niveau donné,
- l'élagage des motifs candidats pour lesquels il existe un motif sous-ensemble éliminé dans les niveaux précédents,
- le test du prédicat sur les candidats générés.

Une structure habituelle pour stocker efficacement les motifs fréquents est la structure de *trie* appelé également arbre préfixé. Cette structure a l'avantage de permettre un stockage optimisé des motifs. De plus elle facilite l'étape de génération des candidats. Un exemple de cette structure est donnée dans la figure 6.3 où le premier chemin depuis la racine, passant par A et allant vers B permet le stockage des motifs $\{A\}$, $\{A, B\}$, $\{A, B, C\}$ et $\{A, B, C, D\}$. A un niveau k, on trouve les motifs de taille k. Par exemple, au niveau 3, il y a 4 motifs de taille 3 : $\{\{A, B, C\}, \{A, B, D\}, \{A, C, D\}, \{B, C, D\}\}$. La phase de génération des candidats consiste à générer les motifs candidats de taille $k + 1$, à partir des motifs de taille k. La phase d'élagage consiste à parcourir le trie pour vérifier si tous les sous-ensembles des motifs générés au

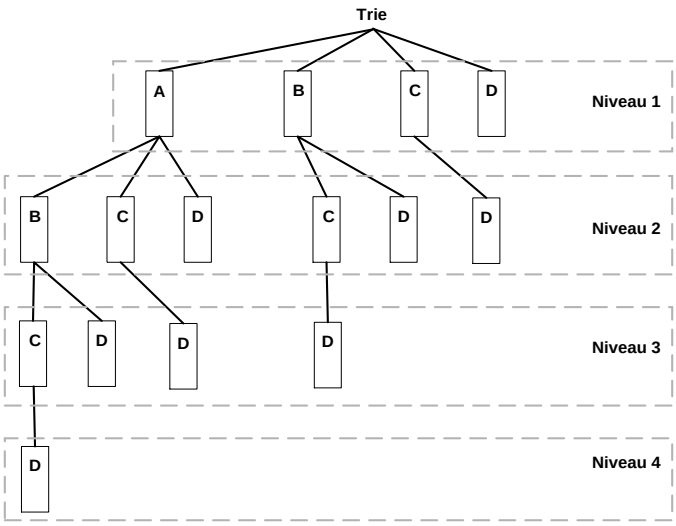

FIG. 6.3 – Exemple de TRIE

niveau k sont effectivement fréquents (i.e. s'ils sont dans le trie). Le test du prédicat consiste à vérifier si les candidats générés au niveau k vérifient le prédicat. Le principe de fonctionnement d'APriori est décrit en annexe B, section B.3.

Nous nous intéressons maintenant à l'implémentation de *ComputeEConj* qui repose sur l'algorithme APriori.

6.2.2 Alternatives pour l'implémentation d'APriori

Pour le sous-module *ComputeEConj*, nous disposons de deux implémentations : *ComputeEConj_1* et *ComputeEConj_2*. Chacune d'elles adapte APriori dans le sens où elles utilisent un prédicat de non-inclusion à la place du prédicat usuel. Dans ce contexte, un motif correspond à un sous-ensemble de F_{mot}. Le prédicat de non-inclusion implémente le prédicat P_1 introduit en section 5.3. Le prédicat P_1 est vrai pour un motif donné, si l'intersection des ensembles de valeurs identifiés par le motif n'est pas inclus dans un ensemble donné E. Les implémentations *ComputeEConj_1* et *ComputeEConj_2* sont basées respectivement sur :

- une *approche faiblement couplée* [72] dans le sens où *ComputeEConj_1* est uniquement implanté à l'aide de procédures stockées. Les données restent dans le SGBD et toutes les opérations nécessaires à la réalisation de l'algorithme sont des requêtes SQL.
- une *approche externe* [72] dans le sens où *ComputeBucket* fait appel à un programme extérieur *ComputeEConj_2* pour le calcul des solutions de *E_conj_rewrite(mot,E)*. Dans cette configuration, les données sont exportées vers un programme externe en C++. Les opérations sont alors réalisées en mémoire principale, puis les résultats sont transférés dans le SGBD.

Dans l'approche faiblement couplée, les trois principales étapes d'APriori sont implémentées sous forme de procédures stockées en PLSQL. Ces procédures s'appuient sur une table $Trie$ qui implémente la structure de trie en relationnel. Cette structure est inspirée de celle utilisée pour l'implémentation d'APriori sous le système ATLaS [78]. *ComputeEConj_1* prend en

entrée les candidats sous forme d'une table en relationnel et renvoie les plus petites conjonctions de vues qui sont subsumées par l'atome $\forall mot.E$. Les phases de génération et d'élagage reposent sur des requêtes hiérarchiques sur $Trie$. L'implémentation de *ComputeEConj_1* est détaillée en annexe B, section B.3.1.

ComputeEConj_2 est plus précisément une adaptation d'une implémentation C++ d'APriori réalisée par C.Borgelt [14]. Elle est reconnue pour être la plus efficace actuellement au regard des expérimentations réalisées pour les ateliers FIMI (Frequent Itemset Mining Implementation) [35, 34]. Cette implémentation a été réalisée en collaboration avec F.Flouvat dans le cadre de ses travaux de thèse, voir par exemple [28] dans un contexte différent. D'un point de vue pratique, *ComputeEConj_2* est un exécutable appelé à la place de *ComputeEConj_1* par *ComputeBucket*. *ComputeEConj_2* prend en entrée les candidats sous forme d'un fichier texte et retourne un fichier texte qui contient les plus petites conjonctions de vues qui sont subsumées par l'atome $\forall mot.E$.

Nous nous intéressons maintenant aux résultats expérimentaux de chaque version de *ComputeEConj*.

6.3 Expérimentations

6.3.1 Présentation des jeux d'essais

En dépit de l'existence d'un contexte applicatif, il n'a pas été possible d'expérimenter *ComputeEConj* sur des jeux de données réels. En effet, le projet d'intégration des sources de données agricoles ne fait que débuter. Aussi pour le moment, nous ne disposons pas d'une vision globale sur l'ensemble des données qu'il est possible de récolter. Pour récupérer ces informations, il est nécessaire de passer par une phase d'enquêtes auprès des acteurs du monde agricole. De telles enquêtes ont été amorcées mais ne sont pas encore achevées. Aussi les expérimentations ont été réalisées sur des données synthétiques.

Les jeux d'essais ont été créés à l'aide d'un générateur aléatoire implémenté sous Oracle. Nous avons élaboré 2 types de jeux d'essais :

- des jeux dans lesquels la cardinalité des contraintes est fixe, i.e. tous les ensembles de valeurs des vues ont la même cardinalité,
- des jeux dans lesquels la cardinalité des contraintes varie de 1 à n.

Ces jeux permettent de mesurer l'impact des propriétés portant sur la taille des contraintes, sur la configuration de l'espace de recherche et des solutions. Pour chacun de ces jeux, nous avons fait varier la cardinalité des contraintes et le nombre de vues afin d'estimer le nombre de vues qui peuvent être traitées par les différentes implémentations. On a mesuré les temps d'exécution de *ComputeEConj_1* et *ComputeEConj_2* sur chaque type de jeux d'essais. Pour des jeux d'essais de petites tailles, *ComputeEConj_1* est rapidement mis en difficulté. En revanche, *ComputeEConj_2* est efficace dans la mesure où l'espace des solutions se prête bien à une résolution par des algorithmes de type APriori. Les expérimentations ont été réalisées sur un PC pentium 4 pro 2.6 GHz avec 3 Go de mémoire.

6.3.2 Expérimentations de *ComputeEConj_1*

Dans cette section, nous mettons en évidence les étapes les plus coûteuses de *ComputeEConj_1* ainsi que l'influence du nombre de vues/valeurs sur les temps d'exécution. Ensuite nous montrons que cette implémentation ne passe pas à l'échelle.

Dans la première série de tests, les valeurs dans les contraintes de valeurs sont tirées au sort parmi 100 valeurs distinctes. Les résultats de cette série sont résumés dans la table 6.1.

Entrée	Génération	Elagage	Prédicat	TempsTotal
20 vues,10 valeurs	0,06 s	0,992 s	0,370 s	1,372 s
30 vues,10 valeurs	0,16 s	11,766 s	0,732 s	12,738 s
45 vues,10 valeurs	0,43 s	165,567 s	1,914 s	188,101 s
60 vues,10 valeurs	4,631 s	1266,333s	3,916 s	1272,74 s

TAB. 6.1 – Temps d'exécution de *ComputeEConj_1* : valeurs tirées au hasard parmi 100

Pour traiter 20 vues dont la taille des contraintes de valeurs est de 10 éléments au maximum, *ComputeEConj_1* a nécessité 1.372 seconde. Pour traiter 60 vues dont la taille des contraintes de valeurs est de 10 éléments au maximum, *ComputeEConj_1* a nécessité 21 min 09 secondes. Pour le cas du traitement de 60 vues, la figure 6.4 montre que l'étape la plus coûteuse est celle de l'élagage. Ceci résulte du fait que chaque opération de *ComputeEConj_1* est équivalente à une requête sur le SGBD. De plus les phases d'élagage s'appuient sur des requêtes hiérarchiques qui sont coûteuses en termes de temps d'exécution.

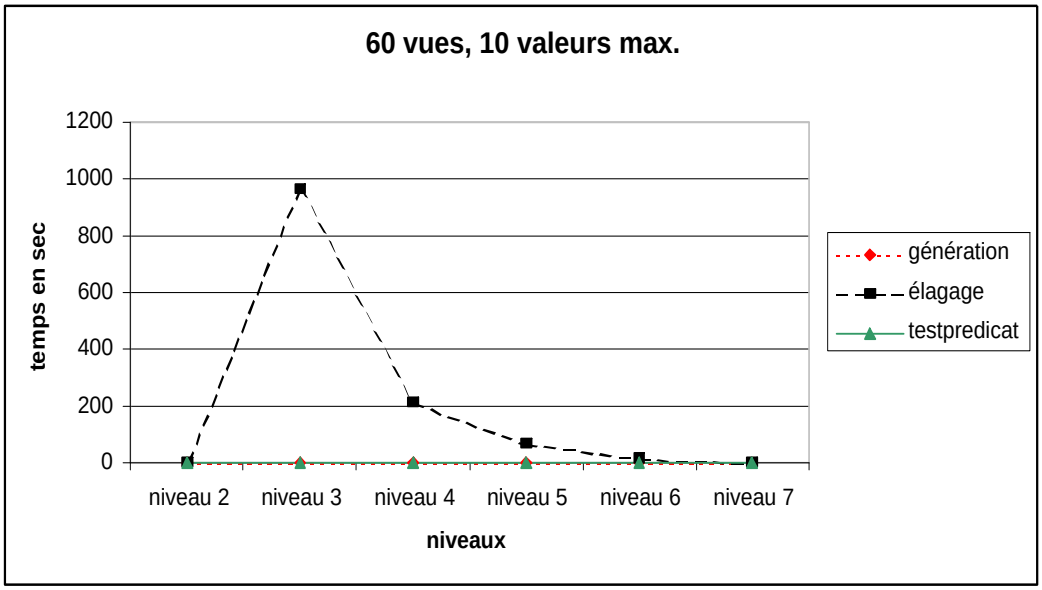

FIG. 6.4 – Temps d'exécution de *ComputeEConj_1*

Une deuxième série de test de *ComputeEConj_1* a été réalisée en tirant les valeurs des contraintes au hasard parmi 500. Elle met en évidence que plus le nombre de valeurs augmente,

plus le temps d'exécution de *ComputeEConj_1* est affecté. Cette série a donné les résultats recensés par la table 6.2. Dans ce cas de figure, les contraintes de valeurs ont peu de chance

Entrée	Temps Total	# éléments dans B	# éléments dans F
60 vues,10 valeurs	1'13	1687	161
60 vues,20 valeurs	17'33	1790	610
60 vues,30 valeurs	56'58	2645	1521
80 vues,10 valeurs	8'29	3017	269

TAB. 6.2 – Temps d'exécution de *ComputeEConj_1* avec élagage : valeurs tirées au hasard parmi 500

de s'intersecter. Aussi l'intersection des motifs conduit rapidement à l'ensemble vide, i.e. des motifs solutions sont de petite taille. Ainsi il est possible de traiter plus de vues. Cependant dès que la taille des contraintes augmente, les temps d'exécution deviennent très longs. Par exemple, le traitement de 60 vues ayant des contraintes de valeurs de taille 30 au maximum, nécessite près d'une heure alors que la taille de la bordure négative n'est que 2645 éléments.

En supprimant la phase d'élagage, on remarque que la phase du test du prédicat est également coûteuse comme le montre la figure 6.5. Pour traiter 60 vues dont la taille des contraintes de valeurs est de 25 éléments au maximum, *ComputeEConj_1* nécessite 14min 21secondes. Notons que la génération des candidats de niveaux 6 et 7 est de l'ordre de la minute (figure 6.6).

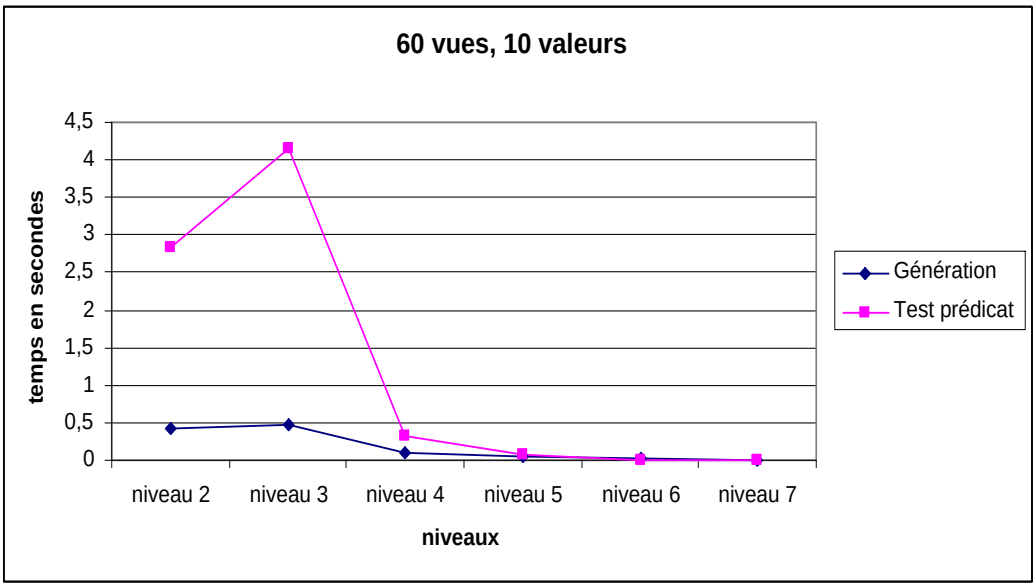

FIG. 6.5 – Temps d'exécution de *ComputeEConj_1*sans élagage

Ces différents jeux d'essais montrent que l'exécution de *ComputeEConj_1* est très coûteuse, surtout pour les phases d'élagage et de test du prédicat. Ce résultat est du au fait que chaque opération de *ComputeEConj_1* est équivalente à une requête sur le SGBD. A titre d'exemple,

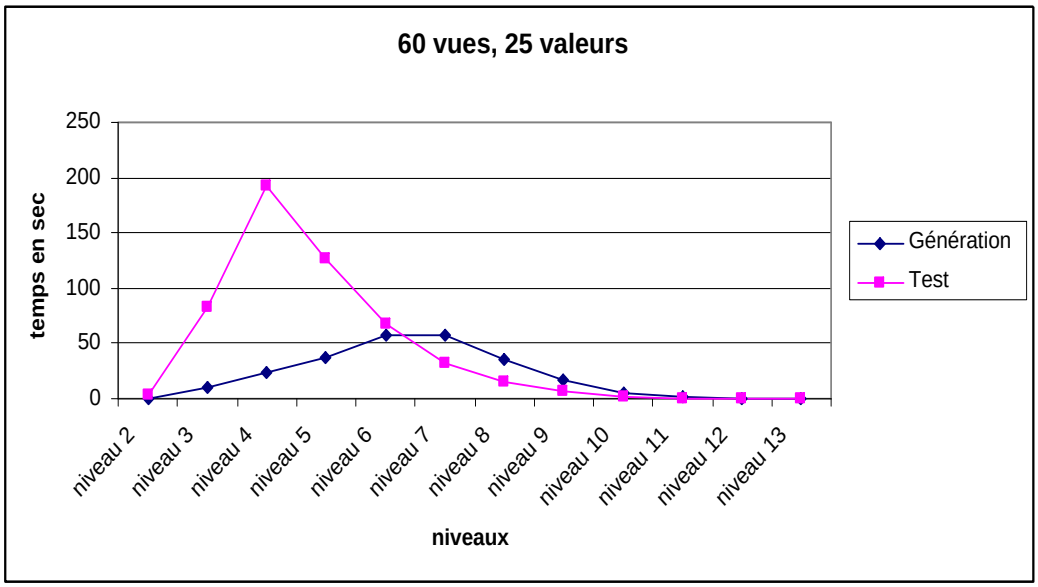

FIG. 6.6 – Temps d'exécution de *ComputeEConj_1* sans élagage : 25 valeurs au maximum dans les contraintes de valeurs

s'il y a C_n^k motifs à un niveau donné où n est le nombre de candidats à l'entrée de *ComputeEConj_1*, on doit effectuer $C_n^k \times |B|$ requêtes pour la phase d'élagage et C_n^k requêtes pour le test du prédicat.

Ces résultats nous conduisent à nous orienter pour le calcul des paniers vers des implémentations plus efficaces d'APriori, suivant des approches externes. La section qui suit, s'attache à présenter les résultats d'exprérimentations de l'implémentation externe de *ComputeEConj*.

6.3.3 Expérimentations de *ComputeEConj_2*

Reprenons quelques jeux coûteux pour *ComputeEConj_1*. Comme le montre la table 6.3, l'application *ComputeEConj_2* traite ces cas instantanément.

	ComputeEConj_1	*ComputeEConj_2*
60 vues, 10 valeurs parmi 100	$21'09s$	$0s$
60 vues, 10 valeurs parmi 500	$1'13s$	$0s$
60 vues, 20 valeurs parmi 500	$17'33s$	$0s$
60 vues, 30 valeurs parmi 500	$56'58s$	$0s$
80 vues, 10 valeurs parmi 500	$8'29s$	$0s$

TAB. 6.3 – Comparaison des temps d'exécution de *ComputeEConj_1* et *ComputeEConj_2* pour des contraintes de taille variable

Nous cherchons maintenant à obtenir une estimation du nombre de vues pouvant être traitées par *ComputeEConj_2* en un temps acceptable pour différentes tailles de contraintes. Notons

que cette borne correspond au nombre de vues que *ComputeBucket* a identifiées comme étant pertinentes à la réécriture d'un atome de la forme $\forall mot.valeurs$. Ainsi cette borne conditionne uniquement la taille de l'entrée de *ComputeEConj_2* et ne fait pas figure de limite sur le nombre de vues que *ComputeBucket* peut traiter.

	15000 vues	10000 vues	5000 vues	3000 vues
10	$\times$	$38s$	$6s$	$2s$
20	$\times$	$\times$	$17s$	$3s$
30	$\times$	$\times$	$\times$	$8s$
40	$\times$	$\times$	$\times$	$19s$

TAB. 6.4 – Temps d'exécution de *ComputeEConj_2* pour des contraintes de taille fixe

	15000 vues	10000 vues	5000 vues	3000 vues
10	$45s$	$19s$	$5s$	$1s$
20	$\times$	$38s$	$6s$	$2s$
30	$\times$	$\times$	$10s$	$3s$
40	$\times$	$\times$	$17s$	$3s$

TAB. 6.5 – Temps d'exécution de *ComputeEConj_2* pour des contraintes de taille variable

Comme le montrent les tableaux 6.4 et 6.5, lorsque la taille des contraintes de valeurs est petite, il est possible de prendre un grand nombre de vues en entrée (e.g. 10000 vues pour des contraintes de taille 10 et 15000 vues pour des contraintes de taille inférieure à 10).

Dès que la taille des contraintes augmente, le nombre de vues que peut traiter l'algorithme, diminue. Par exemple, pour des contraintes de taille 20 exactement (tableau 6.4), il est possible de traiter 5000 vues alors qu'il n'est pas possible pour ce même nombre de vues, de traiter des contraintes de taille 30.

Notons que pour des contraintes dont la taille est comprise entre 1 et 40 (tableau 6.5), *ComputeEConj_2* traite sans problème 5000 vues. Ceci s'explique par le fait que plus la taille des contraintes est grande, plus les contraintes risquent de s'intersecter et que cette intersection ne soit pas incluse dans E. Par conséquent, le nombre de motifs intéressants est susceptible d'être important et le programmme dans ce cas, nécessite plus d'espace mémoire que disponible.

Comme le montre le graphique de la figure 6.7, la bordure négative est formée d'un grand nombre de motifs (12378154) de petite taille (≤ 3). Pour les trouver, *ComputeEConj_2* a généré et testé 12915503 motifs. Dans nos jeux de données, la bordure négative se caractérise en général, par un grand nombre de motifs de petites tailles.

D'une manière générale, APriori est mis en difficulté lorsqu'il existe de grandes solutions. En effet, lorsqu'APriori génère un motif candidat de taille k, APriori a alors testé ses 2^k sous-ensembles. Aussi nous avons généré des jeux d'essais qui s'approchent du cas critique, afin d'obtenir une configuration de jeux dans laquelle la taille des solutions est grande.

Dans le tableau 6.6, on trouve deux jeux d'essais. Le premier jeu est un cas critique, i.e. tout l'espace de recherche doit être parcouru. Dans ce cas de figure, 2^{30} candidats sont testés

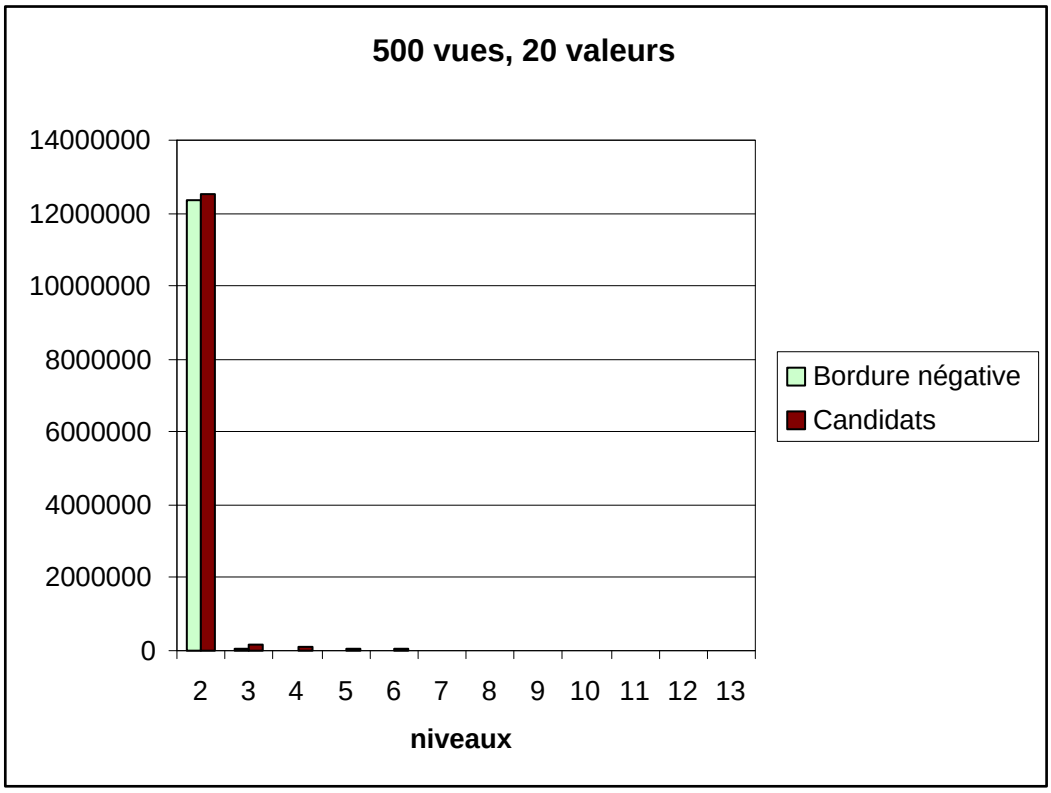

FIG. 6.7 – Distribution de la bordure négative pour 500 vues

30 vues, pire cas	30 vues, 30 valeurs exact./100
$\times$	$15s$

TAB. 6.6 – Temps d'exécution de *ComputeEConj_2* dans des configurations critiques

et la bordure négative est formée d'un seul élément de taille 30. Le nombre de motifs devient rapidement trop grand pour tenir en mémoire.

Le deuxième cas est dérivé du cas critique. Le jeu est composé de 30 vues en entrée dont les contraintes de valeurs sont de cardinalité 30 et où les valeurs ont été tirées au hasard parmi 100. Dans ce cas, les contraintes sont susceptibles d'avoir beaucoup de valeurs en commun, et donc de s'intersecter fortement. Ainsi les motifs de la bordure négative sont susceptibles d'être grands. En effet comme le montre la figure 6.8, les motifs sont composés d'au plus 6 éléments. Ce jeu est cependant traité rapidement car le nombre de candidats testés est 243308, même si leur taille est grande (≤ 16).

Discussion des résultats

Comme l'ont montré les différentes expérimentations dans le domaine de la découverte des connaissances [35, 34] et celles que nous avons réalisées avec le nouveau prédicat de non-

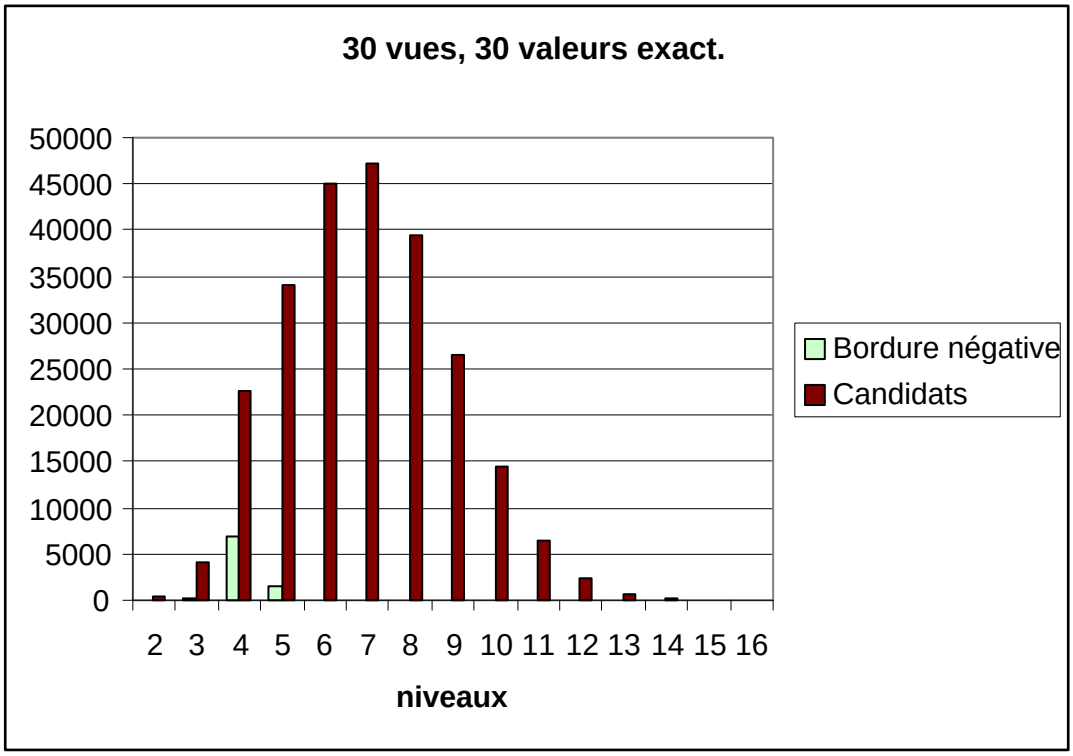

FIG. 6.8 – Distribution de la bordure négative pour 30 vues

inclusion, ce type d'implémentation est plus particulièrement adapté au calcul de motifs de petite taille. L'exécution d'APriori devient problématique quand une grande partie de l'espace de recherche doit être parcourue, i.e. quand il existe de grands motifs intéressants. Dans de telles configurations, pour le problème de la découverte des fréquents, des algorithmes ont été proposés afin de trouver plus efficacement les motifs de grande taille [40, 59, 27]. Il pourrait être intéressant d'adapter ces algorithmes à notre problème. Néanmoins, de tels algorithmes ne trouvent pas obligatoirement la bordure négative et leur stratégie ne s'applique peut-être pas aussi efficacement à notre contexte de réécriture.

Les jeux générés ont également montré que la bordure négative est généralement formée d'un grand nombre de motifs de petite taille. Dans le contexte de la réécriture, un grand nombre d'éléments dans la bordure négative, i.e. dans les paniers, est problématique lors de la phase du produit cartésien entre les paniers. Les cas critiques ont montré qu'il est possible d'avoir des motifs de très grande taille dans les paniers.

D'une manière générale, les expérimentations ont montré qu'il est possible de traiter jusqu'à 15000 vues pour réécrire un atome. Dans [68], l'implémentation de l'algorithme Minicon est considérée comme permettant le passage à l'échelle, car elle permet de traiter en totalité pour réécrire une requête jusqu'à 12235 vues. Aussi par rapport aux résultats de cette expérimentation, on peut conclure que *ComputeBucket*, basée sur *ComputeEConj_2*, est un algorithme de réécriture qui passe à l'échelle.

Pour conclure, nous avons présenté les expérimentations de deux implémentations de *ComputeEConj*. *ComputeEConj_1* est totalement intégrée dans le SGBD. L'avantage de cette so-

lution réside dans le fait que les données restent dans le SGBD et les opérations se font directement dans le SGBD. Cependant chaque opération étant implémentée comme une requête, l'implémentation *ComputeEConj_1* d'APriori devient rapidement coûteuse. *ComputeEConj_2* est une implémentation externe au SGBD. Cette implémentation est beaucoup plus efficace que *ComputeEConj_1*. En effet, dans *ComputeEConj_1*, chaque opération est une requête alors que dans *ComputeEConj_2*, les opérations sont optimisées d'un point de vue algorithmique. De plus, *ComputeEConj_2* utilise des structures de données adaptées pour compresser la taille occupée par les données en mémoire.

Conclusion

Dans cette thèse, nous nous sommes intéressés au problème de la réécriture de requêtes en termes de vues. Ce problème joue un rôle important dans le domaine de l'intégration de données. Il contribue en effet à la mise en oeuvre du module de traitement de requêtes dans les médiateurs suivant une approche LAV. Le cadre applicatif de cette thèse est le développement d'un système d'intégration de données dans le domaine agricole. Dans ce contexte, nous avons plus particulièrement étudié le problème de la réécriture de requêtes en termes de vues en présence de contraintes de valeurs. Les contraintes de valeurs peuvent être vues comme un type de données énuméré. Elles permettent de spécifier les valeurs autorisées pour un attribut donné. L'exploitation des contraintes de valeurs dans notre contexte a un double intérêt : *(i)* elles permettent une description plus fine des requêtes et des vues et *(ii)* elles permettent d'optimiser le processus d'évaluation des requêtes en réduisant le nombre de réécritures candidates, et par conséquent le nombre de sources de données à solliciter.

Pour étudier le problème de la réécriture de requête en présence de contraintes de valeurs, nous nous sommes appuyés sur deux cadres : *(i)* les logiques de description pour formaliser le problème et étudier ses propriétés théoriques, et *(ii)* un cadre de découverte des connaissances pour développer des solutions algorithmiques permettant de le résoudre. Plus précisément, nous avons formalisé le problème de réécriture dans le cas où les requêtes et les vues sont décrites dans les langages $\mathcal{FL}_0(\mathcal{O}_v)$ ou $\mathcal{ALN}(\mathcal{O}_v)$. Le problème est évidemment plus difficile dans $\mathcal{ALN}(\mathcal{O}_v)$ que dans $\mathcal{FL}_0(\mathcal{O}_v)$. Cette difficulté est due à l'interaction entre les contraintes de valeurs et les restrictions de cardinalités sur les rôles. Nous avons montré que dans notre cadre d'étude pour calculer toutes les réponses certaines il suffit de considérer comme langage de réécriture le langage $\mathcal{L} = \{\sqcap, \sqcup\}$.

Etant donnés les langages utilisés dans notre contexte, i.e., $\mathcal{FL}_0(\mathcal{O}_v)$ ou $\mathcal{ALN}(\mathcal{O}_v)$ pour décrire les vues et les requêtes et $\mathcal{L} = \{\sqcap, \sqcup\}$ comme langage de réécriture, l'espace de recherche des réécritures est de taille exponentielle mais fini. De plus, en donnant une borne concernant la taille des réécritures, il est alors facile d'exhiber un algorithme en temps exponentiel pour calculer les réécritures d'une requête donnée.

Un aspect original de ce travail est d'avoir établi un lien entre le problème de la réécriture de requêtes dans les systèmes de médiation et un cadre de découverte de connaissance dans les bases de données. En effet, les problèmes de calcul des réécritures qui sont spécifiques à la présence des contraintes de valeurs peuvent se ramener à un calcul de bordure négative d'une théorie donnée dans le cadre introduit dans [57]. Ainsi nous pouvons bénéficier d'algorithmes de découverte des connaissances existants comme, par exemple, l'algorithme APriori pour optimiser le calcul des rééécritures de requêtes.

Un prototype implémentant notre approche a été développé. Les expérimentations réalisées avec ce prototype sur des données synthétiques démontrent sa capacité à passer à l'échelle. Néanmoins, il est difficile de comparer les performances de notre prototype avec d'autres applications de réécriture de requêtes dans la mesure où dans ce domaine les résultats théoriques ont toujours primé sur les résultats expérimentaux. Aujourd'hui, compte tenu du nombre important de sources de données disponibles, le passage à l'échelle des systèmes de médiation devient primordial. A notre connaissance, [68] est une des rares références qui décrit l'évaluation des performances d'un algorithme de réécriture.

Le prototype actuel implémente principalement le calcul des paniers d'une requête donnée dans le cadre de la logique $\mathcal{FL}_0(\mathcal{O}_v)$. Il serait cependant intéressant d'implémenter les phases manquantes du processus de réécritures et notamment celle concernant la combinaison

des éléments des paniers pour former les réécritures candidates. Il reste également à étendre le prototype à la logique $\mathcal{ALN}(\mathcal{O}_v)$.

D'un point de vue théorique, diverses perspectives peuvent être envisagées dans la continuité de ce travail. Nous nous sommes intéressés au problème de réécriture de requêtes en présence du constructeur $\mathcal{O}_v$. Les résultats présentés dans cette thèse peuvent cependant être aisément étendus au cas du constructeur $\mathcal{O}$, considéré dans sa sémantique non standard telle que déclinée dans CLASSIC [15]. En effet, la caractérisation structurelle de la subsomption proposé dans CLASSIC peut être exploitée pour étendre nos résultats à la logique $\mathcal{ALN}(\mathcal{O})$.

Le cadre formel des logiques de description nous a permis de spécifier le problème de la réécriture en présence de contraintes de valeurs. Cependant si les logiques de description se prêtent bien à la description du schéma global d'un système de médiation, elles ne sont pas très adaptées en tant que langage de requête. En effet, les logiques de description ne permettent pas la représentation des prédicats n-aires. Une solution pour pallier les limites des logiques de description serait d'utiliser le cadre des langages hybrides comme, par exemple, le langage CARIN [32]. Ainsi il devient possible de décrire le schéma global avec une logique de description et d'exprimer les requêtes et les vues avec des requêtes conjonctives qui référencent dans leur corps des descriptions de concepts et de rôles. Une perspective intéressante du travail présenté dans cette thèse serait, par exemple, d'étudier le problème de réécriture de requête dans CARIN-$\mathcal{ALN}(\mathcal{O}_v)$.

Enfin, l'utilisation d'un cadre de découverte des connaissances dans les bases de données offre des perspectives intéressantes pour l'implémentation d'algorithmes de réécriture de requêtes capable de traiter un grand nombre de vues. Dans notre approche, nous avons utilisé un tel cadre pour améliorer le coût en temps de la construction des paniers d'une requête donnée. Cependant, le processus de calcul des réécritures comporte une autre phase critique qui est celle de la reconstitution des réécritures conjonctives globales à partir des réécritures de chaque panier. Ce type d'étape intervient également dans d'autres algorithmes de réécritures comme, par exemple, le Minicon [68]. Il serait intéressant d'étudier comment cette étape peut se rattacher à un cadre de découverte des connaissances.

Bibliographie

[1] S. Abiteboul and O. M. Duschka. Complexity of answering queries using materialized views. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 254–263. ACM Press, 1998.

[2] F. N. Afrati, M. Gergatsoulis, and T. Kavalieros. Answering queries using materialized views with disjunctions. In *Proc. of the 7th Int. Conf. on Database Theory, ICDT'99*, volume 1540 of Lecture Notes in Computer Science, pages 435–452, 1999.

[3] F. N. Afrati, C. Li, and P. Mitra. Answering queries using views with arithmetic comparisons. In *Symposium on Principles Of Database Systems*, pages 209–220, 2002.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedingsnumber20Very Large Databases Santiago de Chile, Chile*, pages 487–499, 1994.

[5] F. Baader, M. Buchheit, and B. Hollunder. Cardinality restrictions on concepts. In *Proceedings of the German AI Conference, KI'94*, volume 861 of *Lecture Notes in Computer Science*, pages 51–62, Saarbrücken (Germany), 1994. Springer–Verlag.

[6] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[7] F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies – revisited. Report 00-04, LTCS, RWTH Aachen, Germany, 2000. See http ://www-lti.informatik.rwth-aachen.de/Forschung/Reports.html.

[8] F. Baader, R. Küsters, and R. Molitor. Rewriting concepts using terminologies - revisited. Technical Report LTCS-Report-99-12, Aachen University of Technology Research Group for Theoretical Computer Science, 2000.

[9] F. Baader and Ralf Küsters. Least common subsumer computation w.r.t. cyclic aln-terminologies. In *Description Logics*, 1998.

[10] C. Baru, A. Gupta, B. Ludascher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and C. Chu. Xml-based information mediation with mix. In *In Proc. ACM SIGMOD Conf on Management of Data*, pages 597–599, 1999.

[11] C. Beeri, A. Halevy, and M.C. Rousset. Rewriting Queries Using Views in Description Logics. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *16th ACM Symposium on PODS, New York, NY, May 12–14, Tucson, Arizona*, pages 99–108. ACM Press, 1997.

[12] Z. Bellahsene and M. Roantree. Querying distributed data in a super-peer based architecture. In *DEXA*, pages 296–305, 2004.

[13] B. Benatallah, M-S. Hacid, C. Rey, and F. Toumani. Request Rewriting-Based Web Service Discovery. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *International Semantic Web Conference (ISWC 2003), Sanibel Island, FL, USA*, volume 2870 of *LNCS*, pages 242–257. Springer–Verlag, October 2003.

[14] C. Borgelt. Efficient implementations of apriori and eclat. In *FIMI'03 Workshop on Frequent Itemset Mining Implementations*, November 2003.

[15] A. T. Borgida and P. F. Patel-Schneider. A semantics and complete algorithm for subsumption in the classic description logic. *Journal of Artificial Intelligence Research*, 1 :277–308, 1994.

[16] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Answering regular path queries using views. In *Proc. of the 16th IEEE Int. Conf. on Data Engineering (ICDE 2000)*, pages 389–398, 2000.

[17] D. Calvanese, G. De Giacomo, M. Lenzerini, and M.Y. Vardi. Query processing using views for regular path queries with inverse. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS 2000)*, pages 58–66, 2000.

[18] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views in description logics. In *KRDB*, pages 6–10, 1999.

[19] Projet Caravel. Le select, publishing data an services made easy, http ://www-caravel.inria.fr/leselect/, 2005.

[20] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1) :65–74, 1997.

[21] J. Van den Bussche. Rewriting queries using views over monadic database schemas. *Inf. Process. Lett.*, 79(3) :111–114, 2001.

[22] F.M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In *Principles of Knowledge Representation and Reasoning*, pages 193–238. CLSI Publications, 1996.

[23] O.M. Duschka and M.R. Genesereth. Answering recursive queries using views. In *In Proc. of the Sixteenth ACM SIGMOD Symposium on Principles of Database Systems*, pages 109–116, 1997.

[24] O.M. Duschka and A.Y. Levy. Recursive plans for information gathering. In *15th International Joint Conference on Artificial Intelligence*, pages 778–784, Nagoya, Japan, 1997.

[25] S. Abiteboul et al. The lowell database research self-assessment. Technical report, 2003.

[26] U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery : An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–34. 1996.

[27] F. Flouvat, F. De Marchi, and J.-M. Petit. ABS : Adaptive borders search of frequent itemsets. In *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*.

[28] F. Flouvat, F. De Marchi, and J.-M. Petit. A thorough experimental study of datasets for frequent itemsets. In *ICDM 2005, To appear*, 2005.

[29] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The tsimmis approach to mediation : Data models and languages. *Journal of Intelligent Information Systems (JIIS)*, 8(2) :117–132, 1997.

[30] F. Goasdoué. *Réécriture de requêtes en termes de vues dans CARIN et Intégration d'information*. Thèse de doctorat, Paris XI Orsay, 2001.

[31] F. Goasdoué, V. Lattes, and M.-C. Rousset. The use of CARIN language and algorithms for information integration : The PICSEL system. *International Journal of Cooperative Information Systems*, 9(4) :383–401, 2000.

[32] F. Goasdoué and M.-C. Rousset. Answering queries using views : A KRDB perspective for the semantic web. *ACM Trans. Inter. Tech.*, 4(3) :255–288, 2004.

[33] F. Goasdoué and M.-C. Rousset V. Lattès. The Use of CARIN Language and Algorithms for Information Integration : The PICSEL System. *IJICIS*, 9(4) :383–401, 2000.

[34] B. Goethals. Frequent itemset mining implementations repository, http ://fimi.cs.helsinki.fi/, 2004.

[35] B. Goethals and M.J. Zaki. Advances in frequent itemset mining implementations : Introduction to FIMI03. Technical report, http ://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-90/intro.pdf, 2003.

[36] G. Grahne and A.O. Mendelzon. Tableau techniques for querying information sources through global schemas. In *ICDT '99 : Proceeding of the 7th International Conference on Database Theory*, pages 332–347, London, UK, 1999. Springer-Verlag.

[37] R. Greenwald and D.C. Kreines. A desktop quick reference. In O'Reilly, editor, *Oracle in a Nutshell*, page 926, 2002.

[38] Aberdeen Group. Enterprise information integration : The new way to leverage e-information (second edition). Technical report, 2003.

[39] Gartner Group. Content integration will become a top priority by 2006. Technical report, 2005.

[40] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma. Discovering all most specific sentences. *ACM Transaction on Database System*, 28(2) :140–174, 2003.

[41] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R.S. Sharma. Discovering all most specific sentences. *ACM Transactions on Database Systems*, 28(2) :140–174, 2003.

[42] V. Haarslev and R. Moller. Racer : A core inference engine for the semantic web. pages 27–36, 2003.

[43] A. Halevy. Answering queries using views : A survey. *VLDB Journal*, 10(4) :270–294, 2001.

[44] A. Halevy. Data integration : A status report. In *German Database Conference BTW-03*, Leipzig, Germany, 2003. Invited Talk.

[45] A. Halevy, Z.G. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *International Conference on Data Engineering (ICDE'2003)*, Bangalore, India, 2003.

[46] I. Horrocks. The FaCT system. pages 307–312, 1998.

[47] I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In B. Nebel, editor, *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, 2001.

[48] I. Horrocks and S. Tobies. Optimisation of terminological reasoning. In *Description Logics*, pages 183–192, 2000.

[49] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. Tane : An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(3) :100–111, 1999.

[50] H. Jaudoin, J.-M. Petit, C. Rey, M. Schneider, and F. Toumani. Query rewriting using views in presence of value constraints. In Frank Wolter Ian Horrocks, Ulrike Sattler, editor, *Description Logics*, pages 112–119, 2005.

[51] H. Jaudoin, F. Toumani, J.-M. Petit, and M. Schneider. Utilisation d'un cadre de découverte des connaissances pour la réécriture de requêtes en présence de contraintes de valeurs. In Jacques Le Maitre, editor, *BDA 2004*, pages 407–426, 2004.

[52] R. Küsters. Characterizing the Semantics of Terminological Cycles in $\mathcal{ALN}$ using Finite Automata. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 499–510. Morgan Kaufmann, 1998.

[53] R. Küsters. *Non-Standard Inferences in Description Logics*, volume 2100 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2001. Ph.D. thesis.

[54] R. Küsters and A. Borgida. What's in an Attribute ? Consequences for the Least Common Subsumer. *Journal of Artificial Intelligence Research*, 14 :167–203, 2001.

[55] M. Lenzerini. Data integration : A theoretical perspective. In *PODS*, Madison, Wisconsin, 2002.

[56] A.Y. Levy, A. Rajaraman, and J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *ProceedingsVery Large DatabasesMumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, September 1996.

[57] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3) :241–258, 1997.

[58] F. De Marchi, S. Lopes, and J.-M. Petit. Efficient algorithms for mining inclusion dependencies. In *EDBT'02*, volume 2287 of *Lecture Notes in Computer Science*, pages 464–476, Prague, Czech Republic, 2002.

[59] F. De Marchi and J.-M. Petit. Zigzag : a new algorithm for mining large inclusion dependencies in databases. In *ICDM 2003*, pages 27–34, Melbourne, Florida, USA, 2003.

[60] C. Martin. Inventaire des documents et outils existants en matière d'enregistrement des pratiques agricoles et de valorisation des données enregistrées (cemagref, maapar, ademe). Technical report, 2002.

[61] C. Martin and C. Pagès. Conditions et moyens d'une gestion informatisée des données pour la mise en oeuvre de l'agriculture raisonnée : Exploration stratégique par la méthode d'audit patrimonial. *Ingénieries - Eau, Agriculture, Territoires*, page 12, 2002.

[62] C. Martin and F. Vigier. Setting up a shared geographic information system for agricultural quality and environmental management at production level. *EFITA*, page 8, 2001.

[63] E. Mena, V. Kashyap, A.P. Sheth, and A. Illarramendi. OBSERVER : An approach for query processing in global information systems based on interoperation across pre-existing ontologies. In *First IFCIS International Conference on Coopera tive Information Systems (CoopIS '96)*, pages 14–25, Brussels, Belgium, June 1996, 1996.

[64] Microsoft. Frequent itemset mining implementations repository, see http ://www.microsoft.com/sql/default.mspx, 2005.

[65] C. Mullins. In IBM Press, editor, *DB2 Developer's Guide*, page 1487. Que, 2004.

[66] B. Nebel. Terminological reasoning is inherently intractable (research note). *Artif. Intell.*, 43(2) :235–249, 1990.

[67] N. F. Noy. Semantic integration : a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4) :65–70, 2004.

[68] R. Pottinger and A. Halevy. Minicon : A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2-3) :182–198, 2001.

[69] C. Rey. *Découverte des meilleures couvertures d'un concept en utilisant une terminologie*. Thèse de doctorat, Blaise Pascal, 2004.

[70] M.-C. Rousset. Backward reasoning in aboxes for query answering. In *KRDB*, pages 50–54, 1999.

[71] M.-C. Rousset and C. Reynaud. Knowledge representation for information integration. *Inf. Syst.*, 29(1) :3–22, 2004.

[72] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems : alternatives and implications. In *SIGMOD '98 : Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 343–354, New York, NY, USA, 1998. ACM Press.

[73] A. Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, (13(2)) :141–176, 1994.

[74] A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3) :183–236, 1990.

[75] A. Silberschatz, M. Stonebraker, and J.D. Ullman. Database research : Achievements and opportunities into the 21st century. Technical report, Stanford, CA, USA, 1996.

[76] V. Soulignac, F. Gibold, F. Pinet, and F. Vigier. Note sur sigemo : Système informatisé de gestion des epandages de matières organiques. *CASSINI*, page 10, 2004.

[77] J.D. Ullman. Information integration using logical views. *Theor. Comput. Sci.*, 239(2) :189–210, 2000.

[78] H. Wang and C. Zaniolo. Atlas : A native extension of sql for data mining. In *SDM*, 2003.

Annexes

Annexe A

Démonstrations

A.1 Caractérisation de la subsomption

L'énoncé du **théorème 3.3.1** est :

Soient C, D deux descriptions de concepts dans leur forme normale.

$C \sqsubseteq D$ si et seulement si une des conditions suivantes est vérifiées :

1) $D \equiv \top$ ou bien

2) $w \in V_C(P)$ pour tout $\forall w.P \in D$.

Ce théorème étant un cas particulier du théorème 3.3.1, sa démonstration est donnée dans celle du théorème 3.4.1.

La preuve du théorème 3.4.1 est dérivée de la caractérisation de la subsomption structurelle donnée dans CLASSIC [54]. Cette caractérisation est formalisée via la notion de graphes de description dont nous rappelons la définition ci-dessous, après avoir au préalable, sélectionné le sous-ensemble de CLASSIC qui correspond à notre langage, et après avoir supprimé la notion d'attribut qui n'est pas traitée dans ce mémoire :

Définition A.1.1
Graphe de description

Un graphe de description est un tuple (N, E, r, l), formé d'un ensemble fini N de noeuds, d'un ensemble fini E de arcs, d'un noeud racine $r \in N$, et d'une fonction l de N dans l'ensemble des labels des noeuds.

Le label d'un noeud est $\bot$ ou bien un tuple de la forme (C, D, H), formé d'un ensemble fini C de noms de concepts (les atomes du noeud), d'un ensemble fini D de valeurs ou $\top_V$ (le dom du noeud) et un ensemble fini H de tuples (les arcs du noeud). Les noms de concept de C sont des noms de concepts atomiques ou $\top_C$ ou $\top_H$.

Un arc est un tuple de la forme (R, m, M, G') où R est un nom de rôle ; m, un min, est un entier positif ; M, un max, est un entier positif ou l'infini ∞ ; G' est un graphe de description imbriquée. G' est appelé graphe de restriction du noeud pour le rôle R. Il est supposé que les noeuds de G' sont distincts de tous les noeuds de G et de tout autre graphe de restriction dans G.

L'article [54] définit également un ensemble de règles non rappelées ici, qui permettent de donner une forme normalisée aux graphes de description. Cette forme normalisée d'un graphe

de description, appelée *graphe canonique*, permet par la suite, de définir l'algorithme de subsomption. On note par $\hat{G}_C$, le graphe canonique d'une description de concept. Un ensemble de règles permettent de transcrire un concept en graphe de description et inversement.

Avant de démontrer la complétude et la justesse du théorème 3.4.1, nous définissons le lemme suivant :

Lemme A.1.1
Soient C une description de concept, $\hat{C}$ sa forme normale, $\hat{G}_C$ le graphe canonique de C, et $T(\hat{G}_C)$ la transcription de $\hat{G}_C$ en description de concept. Soit $T(\hat{G}_C)_{11}$ le concept obtenu après application de la règle 11) de la forme normale au concept $T(\hat{G}_C)$ jusqu'à ce qu'elle ne s'applique plus.

Alors $\hat{C}$ a la même syntaxe que $T(\hat{G}_C)_{11}$.

Ce lemme souligne que la forme normale définie dans ce mémoire et le graphe canonique de CLASSIC explicitent la même connaissance.

La preuve de ce lemme s'appuie sur les règles de normalisation des graphes de description ainsi que sur les règles de fusion des graphes, de leurs arcs et de leurs noeuds données dans [54]. Cependant une lecture de cet article permet une meilleure lisibilité de la démonstration qui suit :

Démonstration

Soient $\hat{C}$ et $T(\hat{G}_C)_{11}$ deux concepts issus de la normalisation de C, le premier via la forme normale présentée dans ce mémoire, le deuxième via le graphe canonique de CLASSIC présenté dans [54].
Alors $T(\hat{G}_C)_{11}$ est une conjonction d'atomes : $\forall w_i.P_i$.

– Si $T(\hat{G}_C)_{11} \equiv \bot$ alors $\hat{G}_C$ est marqué incohérent.
  Selon les règles (1,4,5,7,13) permettant d'obtenir un graphe canonique, C contenait dans sa description une des descriptions suivantes :
  – le concept $\bot$ d'après 1),
  – $\geq nR$ et $\leq mR$ avec $n < m$, selon [5)], ce qui peut aussi signifier que A (modélisé via $\geq 1R_A$) et $\neg A$ (modélisé via $\leq 0R_A$) était dans C
  – la conjonction d'individus classiques et de valeurs d'après [7)]
  – par combinaison des règles 13) et 5), les atomes $\geq 1r_1$, $\forall r_1. \geq 1r_2$, $\forall r_1.r_2.(\leq 2r_3)$ et $\forall r_1.r_2.(\geq 3r_3)$ qui mènent au concept $\bot$.
    La règle 13) spécifie qu'un arc r d'un noeud, dont le graphe de restriction est incohérent (le concept imbriqué est $\forall r.\bot$) doit avoir son max à 0, i.e. il doit rendre explicite le concept $\leq 0r$.
  – un ensemble de valeurs vide selon 4).
  Dans la forme normale présentée ici, la règle 4) n'est pas prise en compte à ce niveau parce qu'il est impossible d'avoir un ensemble de valeurs à la racine r de G, étant donné la syntaxe de $\mathcal{ALN}(\mathcal{O}_v)$.
  La forme normale ne fournit pas non plus la règle 7) car une description ne peut pas contenir dans sa description, une conjonction d'un ensemble de valeurs avec des individus classiques puisque les ensembles de noms de rôles $\mathcal{R}_C$ et $\mathcal{R}_V$ sont disjoints.
  Ansi ces règles exceptée la règle 7) sont couvertes dans l'ordre par les règles (7,3,4) et la combinaison des règles (3,4,10) de notre forme normale. Ainsi notre forme normale est assu-

rée d'inférer le concept bottom à partir des mêmes intéractions de concepts mis en évidence dans CLASSIC.

– Si $(\geq nR)$ appartient à la description de $T(\hat{G}_C)_{11}$, alors par la règle 4) de transformation d'une description concept en un graphe de description, la racine de $\hat{G}_C$ contient un arc R de min égal à n.
  Ceci signifie que C contenait une conjonction de k concepts : $(\geq min_k R)$. Ensuite par application des règles de fusion des arcs, la conjonction de ces k concepts devient $(\geq MAX(min_k)R)$. Ces règles de fusion correspondent à la règle 5) de notre forme normale et assure que $\hat{C}$ contient aussi $(\geq minR)$.
– Si $(\leq nR)$ appartient à la description de $T(\hat{G}_C)_{11}$, alors par la règle 5) de transformation d'une description concept en un graphe de description, la racine de $\hat{G}_C$ contient un arc R de max égal à n.
  Selon les règles de fusion des arcs et la règle 16) de normalisation d'un graphe de description, $(\leq maxR)$ appartient à $T(\hat{G}_C)_{11}$ si ce concept appartenait "explicitement" à $T(\hat{G}_C)_{11}$, ou si $\forall R.E$ appartenait à $T(\hat{G}_C)_{11}$ où la cardinalité de E est égale à max.
  – Dans le premier cas, C puis $G_C$contenait une conjonction de k concepts : $(\leq max_k R)$. Par application des règles de fusion d'arc, $\hat{G}_C)$ contient un arc R de max égal à $MIN(max_k)$ où $MIN(max_k) = n$ et $T(\hat{G}_C)_{11}$ contient $(\leq nR)$. Cette règle correspond avec la règle 6) de notre forme normale et assure que C dans sa forme normale contiendra aussi $(\leq maxR)$.
  – Dans le second cas, c'est la règle 16) de normalisation d'un graphe de description qui rend explicite le concept $(\leq nR)$ dans $\hat{G}_C)$. Elle indique que si la racine de G_C contient un arc R, et, si le graphe de restriction de R a E comme domaine et si la cardinalité n de E est inférieure au max de R alors $\hat{G}_C)$ contient un arc R de max égal n.
    La règle 9) de notre forme normale permet aussi de rendre explicite le concept $(\leq nR)$ si $\forall R.E \in C$ et si la cardinalité de E est égale à n.
    Notons que si la racine de $\hat{G}_C)$ contient un arc R, dont le graphe de restriction de R a E comme domaine et dont le max de R est égal à la cardinalité n de E, ceci peut également du à la règle de fusion d'arcs de même lable puis de leurs noeuds dont les domaines de valeurs ont été intersectés.
    La règle 8) de notre forme normale permet aussi d'expliciter cette connaissance.

  Notons que la combinaison des différentes règles de fusion de graphes de description, des arcs et des noeuds, et la règle 16) de normalisation des graphes peut conduire à ce que $T(\hat{G}_C)_{11} \equiv (\leq nR)$. Ceci est également garanti par la combinaison des règles 6),8),9) de la forme normale.
– Si A $\neg A$ appartiennent à la description de $T(\hat{G}_C)_{11}$, c'est que A ou $\neg A$ appartenaient à C et dans ce cas ils sont aussi dans la description de $\hat{C}$
– Si E appartient à la description de $T(\hat{G}_C)_{11}$: ce cas est impossible car on ne peut avoir un ensemble de valeurs à la racine r de G, d'après la syntaxe de $\mathcal{ALN}(\mathcal{O}_v)$.
– Si $\forall w_i.P_i$ appartient à la description de $T(\hat{G}_C)_{11}$ où $w_i = v_1...v_n$. Dans ce cas, la racine de $\hat{G}_C$ contient un arc v_1 dont le noeud du graphe de restriction contient lui-même un arc v_2 et ainsi de suite jusqu'à l'arc v_n dont le graphe de restriction G_n (de P_i) à une des caractéristiques suivantes :
  – G_n est incohérent et dans ce cas $\forall v_1...v_{n-1}.(\leq 0 v_n)$ appartient à $T(\hat{G}_C)_{11}$

Comme vu précédemment, c'est qu'une des règles ou combinaison des règles (1,4,5,7,13) des graphes avait été appliquée. C'est la règle 13) qui explicite $(\leq 0v_n)$. Ces règles sont couvertes par les règles (7,3,4) et la combinaison des règles (3,4,10), où la règle 10) explicite $(\leq 0v_n)$.
A ce niveau, on peut également considérer la règle de normalisation (4) combinée avec la règle 13) des graphes de descriptions. Cette règle indique que le graphe de restriction G_n est incohérent si le domaine E de G_n est l'ensemble vide. Elle ne s'applique ici, d'après la syntaxe de $\mathcal{ALN}(\mathcal{O}_v)$ que dans le cas où v_n appartient à $\mathcal{R}_\mathcal{V}$, l'ensemble des noms de rôles dont l'image est un ensemble de valeurs. Les règles (4,13 sont couvertes par la règle de normalisation (9). En effet si la cardinalité d'un ensemble de valeurs E d'un concept $\forall v_n.E$, est 0 alors la connaissance $(\leq 0v_n)$ est explicitée.
Ainsi, $\forall w_i.\bot$ sera rendu explicite dans $\hat{C}$ selon les mêmes règles imposées par la normalisation des graphes de description.

– le concept P_i de G_n est tel que $P_i \in \{(\geq nR), (\leq nR), A, \neg A\}$: on retombe dans les cas précédemment évoqués, et $\forall w_i.P_i$ sera également rendu explicite dans $\hat{C}$
– le concept P_i de G_n est tel que $P_i \equiv E$. Alors d'après les règles de fusion d'arcs et de noeuds, le graphe de restriction de l'arc v_{n-1} contenait $k \geq 1$ arcs v_n dont les graphes de restrictions respectifs contenaient des ensembles de valeurs (dom). Lors de la fusion des arcs, les noeuds des k arcs v_n ont été fusionnés et les ensembles de valeurs ont été intersectés pour donner l'ensemble de valeurs E. La règle (8)) de notre forme normale permet également d'expliciter cette connaissance cachée, et garantit que nous trouvons aussi le concept $\forall w_i.E$ dans la forme normale de C.

□

Avant de présenter la preuve du théorème 3.4.1, nous rappelons finalement, l'algorithme de subsomption donné dans [54] réduit au sous-langage $\mathcal{ALN}(\mathcal{O}_v)$ de CLASSIC.

Théorème A.1.1
Algorithme de subsomption dans CLASSIC

Soit une description D et un graphe de description $G = (N, E, r, l)$, $subsumes(D, G)$ renvoie vrai si et seulement si l'une des conditions suivantes est vraie :

1. *G est marqué incohérent, i.e. sa transcription en concept est $\bot$*
2. *D est le concept $\top$*
3. *D est un nom de concept, $\top_C$ ou $\top_H$ est un élement des atomes de r*
4. *D est $(\geq nR)$ et i) un arc de r a R comme rôle et min est supérieur ou égal à n ; ou ii) $n = 0$ et r a $\top_C$ dans ces atomes*
5. *D est $(\leq nR)$ et i) un arc de r a R comme rôle et max est inférieur ou égal à n*
8. *D est $\{I_1, ..., I_n\}$ et le domaine (dom) de r est un sous-ensemble$\{I_1, ..., I_n\}$*
10. *D est $\forall R.H$ et i) un arc de r a R comme rôle et G' comme graphe de restriction et, $subsumes?(H, G')$ est vrai ; ou ii) $subsumes?(H, G(\top))$ est vrai et r a $\top_C$ dans ses atomes*
12. *D est $H \sqcap I$ et $subsumes?(H, G)$ et $subsumes?(I, G)$ sont tous les deux vrais.*

Nous rappelons ci-dessous l'énoncé du **théorème 3.4.1** :
Soient C, D deux concepts, exprimés dans leur forme normale.
$C \sqsubseteq D$ si et seulement si l'une des conditions suivantes est vérifiée :

(1) $C \equiv \bot$ ou $D \equiv \top_c$, ou
(2) $w \in V_C(P) \cup E(C)$ pour chaque $\forall w.P$ dans D.

Démonstration

$\Rightarrow$ Soient C et D deux description de concept tels que $C \sqsubseteq D$. Supposons que D et C sont dans leur forme normale , i.e. $D \equiv \forall w_1.P_1 \sqcap ... \sqcap \forall w_n.P_n$.
Supposons que G est la description de graphe canonique du concept C, comme $C \sqsubseteq D$ alors $subsumes(D, G)$ est vrai.
Nous devons montrer que (1) soit $C \equiv \bot$ ou $D \equiv \top_C$ ou
(2) $w \in V_P(C) \cup E(C)$ pour chaque $\forall w.P$ in D.
Si G est marqué incohérent alors $C \equiv \bot$ et la condition (1) du théorème 3.3.1 est vérifiée.
Si $D \equiv \top_C$ alors la condition (1) du théorème 3.3.1 est également vérifiée.
Sinon, selon la règle 12 de l'algorithme de CLASSIC, nous avons pour tout $(\forall w_i.P_i)$ dans D, $i \in \{1..n\}$, $subsumes(\forall w_i.P_i, G)$ est vrai.

- Si $w_i = \epsilon$ alors $\forall w_i.P_i \equiv P_i$ et un des cas suivants est vérifié :
  - P_i est un nom de concept (A or $\neg A$) et selon la règle 3 de l'algorithme de CLASSIC, P_i appartient à la racine de G. De plus par la règle 3) de transformation d'une description concept en un graphe de description, on a $C \sqsubseteq P_i$. Ainsi $\forall \epsilon.P_i$ appartient à C et ϵ appartient à $V_{P_i}(C)$ par définition de $V_{P_i}(C)$. La condition (2) du théorème 3.3.1 est vérifiée.
  - $P_i \equiv (\geq nR)$ et selon la règle 4 de l'algorithme de CLASSIC
    - (i) ou bien il existe un arc depuis la racine de G avec R comme rôle et dont le min est supérieur ou égal à n. De cette façon par la règle 4) de transformation d'une description concept en un graphe de description : $(\geq minR) \in C$ où $min \geq n$. Ainsi $\forall \epsilon.(\geq minR)$ appartient à C et par définition de $V_{(\geq nR)}(C)$ $(min \geq n)$, $\epsilon \in V_{(\geq nR)}(C)$. La condition (2) du théorème 3.3.1 est donc vérifiée.
    - (ii) ou $n = 0$ ce qui implique que $P_i \equiv \top_C$ et dans ce cas la condition (1) du théorème 3.3.1 est vérifiée.
  - $P_i \equiv (\leq nR)$ et selon la règle 5 de l'algorithme de CLASSIC, il existe un arc depuis la racine de G avec R comme rôle et dont le max est inférieur ou égal à n. Ainsi par la règle 5) de transformation d'une description concept en un graphe de description : $(\leq maxR) \in C$ où $max \leq n$. Ainsi nous avons $\forall \epsilon.(\leq maxR)$ qui appartient à la forme normale de C. Ainsi comme $max \leq n$, ϵ appartient à $V_{(\leq nR)}(C)$ par définition de $V_{(\geq nR)}(C)$. La condition (2) du théorème 3.3.1 est donc vérifiée.

  Nous ne testons pas le cas où $P_i \equiv E$ car l'ensemble de valeurs (selon notre définition de $\mathcal{ALNO}_v$) est uniquement utilisé pour restreindre les rôles $r_v \in \mathcal{R}_V$. Ainsi les ensembles de valeurs ne peuvent pas apparaître dans le domaine (dom) de la racine r de G.
- Si $w_i \neq \epsilon$.
  Supposons que $w_i = r_1.r_2...r_n$. Nous avons $subsumes(\forall w_i.P_i, G)$ est vrai, i.e. $subsumes(\forall r_1.(r_2...r_n.P_i), G)$ est vrai ou de manière équivalente $subsumes(\forall r_1.P_{r_1}, G)$ est vrai avec $P_{r_1} \equiv \forall r_2...r_n.P_i$, est vrai. Ainsi selon la règle 10) de l'algorithme, nous avons :
  - soit ii) $subsumes(P_{r_1}, G(\top_c))$ est vrai et $\forall w_i.P_i \equiv \top_c$. Comme D est donné dans sa forme normale, pour tout n, on a $\forall w_i.P_i \equiv \top_c$ si $subsumes(P_i, G(\top_c))$ est vrai (car

$\top \sqsubseteq P_i$).
Si D est seulement formé de $\forall w_i.P_i$ alors $\forall w_i.P_i \equiv \top_c$ et la condition (1) du théorème 3.3.1 est vérifiée. Sinon $\top_c$ n'apparaîtra pas dans la forme normale de D et ne sera pas traité. Dans la suite de la preuve, on ne détaillera plus ce cas de subsomption.

– soit i) un arc depuis la racine de G a r_1 comme rôle et G_1 comme graphe de restriction, avec $subsumes(P_{r_1}, G_1)$ est vrai. Deux cas peuvent se présenter :
I) ou bien G_1 est marqué incohérent selon la règle 1) de l'algorithme CLASSIC. Par application des règles de transcription d'un graphe de description en une description de concept, $\forall r_1.\bot$ appartient à C. Ainsi r_1 n'importe quel suffixe de r_1, comme $r_1.r_2...r_n = w_i$ appartient à $E(C)$ par définition de $E(C)$. Ainsi la condition (2) du théorème 3.3.1 est vérifiée.
II) ou bien il existe un arc r_2 depuis la racine de G_{r_1} dont le graphe de restriction G_{r_2} est subsumé par P_{r_2}, avec $P_{r_2} \equiv \forall r_3...r_n.P_i$.
Et ainsi de suite jusqu'à ce que,
A) il y ait $i \leq n$ tel que i) il existe dans G un chemin $r_1.r_2...r_i$ formé de la succession d'arcs $\{r_1, r_2, ..., r_i\}$, depuis la racine de G, et ii) le graphe de restriction G_{r_i} du dernier arc r_i est marqué incohérent. Alors selon la règle 3) de transformation d'un graphe de description en une description de concept, on a $\forall r_1.r_2...r_i.\bot \in C$. Ainsi $r_1.r_2...r_i$ et n'importe quel suffixe de celui-ci, comme $r_1.r_2...r_i...r_n = w_i$, appartient à $E(C)$ par définition de $E(C)$ et la condition (2) du théorème 3.3.1 est vérifiée.
OU
B) $i = n$ et i) il existe dans G un chemin $r_1.r_2...r_n$ formé de la succession des arcs $\{r_1, r_2, ..., r_n\}$, depuis la racine de G, et ii) le graphe de restriction G_{r_n} du dernier arc r_n est subsumé par P_i.
B.1) si P_i est un concept atomique (A ou $\neg A$) alors selon la règle 3 de l'algorithme CLASSIC, P_i appartient à la racine de G_{r_n}. Ainsi par la règle 3) de transcription d'un graphe de description en une description de concept, $C \sqsubseteq \forall r_1.r_2...r_i...r_n.P_i$, et $\forall w_i.P_i$ appartient à C. Ainsi $w_i \in V_{P_i}(C)$ par définition de $V_{P_i}(C)$. La condition (2) du théorème 3.3.1 est donc vérifiée.
B.2) si $P_i \equiv (\geq nR)$ alors selon la règle 4 de l'algorithme CLASSIC, il existe un arc depuis la racine de G_{r_n} avec R comme rôle et dont le min est supérieur ou égal à n. Ainsi par transformation d'un graphe de description en une description de concept (règle 3)) : $\forall r_1.r_2...r_i...r_n.(\geq minR) \in C$ où $min \geq n$. Ainsi $w_i \in V_{(\geq nR)}(C)$ par définition de $V_{(\geq nR)}(C)$ ($min \geq n$). La condition (2) du théorème 3.3.1 est alors vérifiée.
B.3) si $P_i \equiv (\leq nR)$ alors selon la règle 5 de l'algorithme CLASSIC, il existe un arc depuis la racine de G_{r_n} avec R comme rôle et dont le max est inférieur ou égal à n. Ainsi par transformation d'un graphe de description en une description de concept (règle 3)) : $\forall r_1.r_2...r_i...r_n.(\leq maxR) \in C$ où $max \leq n$. Ainsi $\forall w_i.(\leq maxR)$ appartient à C et $w_i \in V_{(\leq nR)}(C)$ par définition de $V_{(\geq nR)}(C)$ ($max \leq n$). La condition (2) du théorème 3.3.1 est alors vérifiée.
B.4) si $P_i \equiv E$ alors selon la règle 8 de l'algorithme CLASSIC, le domaine (dom) de la racine de G_{r_n} est un sous-ensemble de E. Soit E' le domaine (dom) de la racine de G_{r_n}. Ainsi $r_1.r_2...r_i...r_n \in V_E(C)$ par définition de $V_E(C)$ ($E' \subseteq E$) et la condition (2) du théorème 3.3.1 est vérifiée.

$\Leftarrow$ Soient C et D deux descriptions de concept dans leur forme normale telle que $D \equiv$

$\sqcap_{i=1}^{n} \forall w_i.P_i$. Nous avons 1) soit $C \equiv \bot$ ou $D \equiv \top_C$ ou 2) $w_i \in V_{P_i}(C) \cup E(C)$ pour chaque $\forall w_i.P_i$ dans D. Nous voulons montrer que cela implique $C \sqsubseteq D$.

1) si $C \equiv \bot$ alors l'interprétation de C est vide et toute description D le subsume. si $D \equiv \top_C$ alors son interprétation est l'ensemble de tous les individus *classiques*, i.e. δ_C. Comme tout concept dans $\mathcal{ALN}(\mathcal{O}_v)$ est un sous-ensemble de δ_C, tout concept C est subsumé par $\top_C$.
2) $w_i \in V_{P_i}(C) \cup E(C)$ pour tout $\forall w_i.P_i$ dans D
   - si $w_i \in E(C)$ alors $\forall v_i.\bot$ où $w_i = v_i u$, est dans la description de C. Ainsi $C \sqsubseteq \forall v_i.\bot \sqsubseteq \forall v_i u.P_i$ et $C \sqsubseteq \forall w_i.P_i$.
   - si $w_i \in V_{P_i}(C)$ alors
     - si $P_i = A$ ou $P_i = \neg A$ alors $\forall w_i.P_i$ est dans la description de C et $C \sqsubseteq \forall w_i.P_i$.
     - si $P_i = (\leq nR)$ alors $\forall w_i.(\leq kR)$ où $k \leq n$ est dans la description de C et $C \sqsubseteq \forall w_i.(\leq kR) \sqsubseteq \forall w_i.P_i$, car $k \leq n$.
     - si $P_i = (\geq nR)$ alors $\forall w_i.(\geq kR)$ où $k \geq n$ est dans la description de C et $C \sqsubseteq \forall w_i.(\geq kR) \sqsubseteq \forall w_i.P_i$, car $k \geq n$.
     - si $P_i = E$ alors $\forall w_i.E'$ où $E' \subseteq E$ est dans la description de C et $C \sqsubseteq \forall w_i.E' \sqsubseteq \forall w_i.P_i$, car $E' \subseteq E$.

     Ainsi pour tout $\forall w_i.P_i$ dans D nous avons $C \sqsubseteq \forall w_i.P_i$ et $C \sqsubseteq \sqcap_{i=1}^{n} \forall w_i.P_i$ et $C \sqsubseteq D$.

□

A.2 Réponses certaines

L'énoncé de la proposition 4.1.1 est rappelé ci-dessous :

Soient $\mathcal{S}$ un schéma.

Soient Q une requête et $\mathcal{V}$, l'ensemble des vues tels que Q et $\mathcal{V}$ sont exprimés dans $\mathcal{L}$, en termes de $\mathcal{S}$.

Soit C^{ext}, l'extension d'une conjonction C de vues de $\mathcal{V}$.

Si t est une réponse certaine de Q, alors il existe une conjonction de vues de $\mathcal{V}$, C, subsumée par Q, telle que $t \in C^{ext}$.

Démonstration
Soit t une réponse certaine.
Soit $Q \equiv A_1 \sqcap \ldots \sqcap A_n$ une requête.
Soit $\mathcal{I}$ un modèle de $\mathcal{S}$, tel que $V^{ext} \subseteq V^{\mathcal{I}}, \forall\, V \in \mathcal{V}$.
On veut montrer que si t est une réponse certaine de Q, i.e. $t \in Q^{\mathcal{I}}$, alors il existe une conjonction C de vues de $\mathcal{V}$ telle que $C \sqsubseteq Q$ et $t \in C^{ext}$.

Selon la définition 4.1.1 des réponses certaines, si t est une réponse certaine de Q, alors $t \in \mathcal{V}^{ext}$. Il existe donc un sous-ensemble de $\mathcal{V}$ dont les vues contiennent t dans leur extension.
Soit M, l'ensemble de toutes les vues qui contiennent t dans leur extension. Soit C_M, la conjonction des vues de M. Alors $t \in C_M^{ext}$.

Il reste maintenant à montrer que C_M est obligatoirement subsumée par la requête Q. Pour cela on suppose que C_M n'est pas subsumée par Q, et on montre que t n'est pas une réponse certaine de Q.

Si $C_M \not\sqsubseteq Q$ alors comme $\mathcal{L} \in \{\mathcal{FL}_0(\mathcal{O}_v), \mathcal{ALN}(\mathcal{O}_v)\}$, il existe $d \in \{1, \ldots, n\}$ tel que $C_M \not\sqsubseteq A_d$.
Ainsi il existe une interprétation J pour laquelle $C_M^{\mathcal{J}} \not\subseteq A_d^{\mathcal{J}}$.
Supposons J définie comme suit :
$\forall i \in \{1, \ldots, n\} \backslash \{d\}$, $A_i^J = A_i^I$ et $A_d^J = A_d^I \backslash \{t\}$.
Vérifions que J est conforme aux extensions des vues, i.e. que $V^{ext} \subseteq V^{\mathcal{J}}$, $\forall V \in \mathcal{V}$.
Pour toutes les vues de $\mathcal{V} \backslash M$, comme leurs extensions ne contiennent pas t, la suppression de t dans A_i^I n'a pas d'incidence sur l'interprétation $V^{\mathcal{J}}$ des vues de $\mathcal{V} \backslash M$. Ainsi $\forall V \in \mathcal{V} \backslash M$, $V^{\mathcal{J}} = V^{\mathcal{I}}$ et par conséquent $V^{ext} \subseteq V^{\mathcal{J}}$.
Pour les vues de M, comme $C_M \not\sqsubseteq A_d$, alors A_d n'apparaît pas dans aucune description de vue de M. Ainsi l'interprétation $\mathcal{J}$ de A_d n'a aucune incidence sur l'interprétation des vues de M.
Ainsi $\forall V \in M$, $V^{\mathcal{J}} = V^{\mathcal{I}}$ et $V^{ext} \subseteq V^{\mathcal{J}}$.
Il existe donc un modèle $\mathcal{J}$ de S, tel que $V^{ext} \subseteq V^{\mathcal{J}}$, $\forall V \in \mathcal{V}$.
Cependant, t n'appartient pas à $Q^{\mathcal{J}}$ car $t \notin A_i^J$, et donc t n'est pas une réponse certaine de Q.
□

A.3 Caractérisation de *rewrite(Q,$\mathcal{V}$,L)* en termes de *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*

L'énoncé du **lemme 4.1.2** est :

Soit $\mathcal{V}$ une terminologie dans $\mathcal{L}$. Soient Q_{i_1} et Q_{i_2} deux conjonctions de vues satisfiables, en termes de $\mathcal{V}$.

$Q_{i_1} \sqsubseteq Q_{i_2}$ si et seulement si Q_{i_2} réfère un sous-ensemble de vues apparaissant dans la description de Q_{i_1}.

Démonstration

(⇐) Soit Q_{i_1} et Q_{i_2} deux conjonctions de vues de $\mathcal{V}$. Nous montrons que si Q_{i_2} réfère un sous-ensemble de vues de Q_{i_1} alors $Q_{i_1} \sqsubseteq Q_{i_2}$.
Supposons que $Q_{i_1} \equiv V_1 \sqcap \ldots \sqcap V_n$.
Q_{i_2} réfère uniquement un sous-ensemble de $\{V_1, \ldots, V_n\}$.
Supposons Q_{i_2} équivalente à l'expression suivante : $Q_{i_2} \equiv V_1 \sqcap \ldots \sqcap V_{j-1} \sqcap V_{j+1} \sqcap \ldots \sqcap V_n$.
Ainsi, comme la logique de description des vues est $\mathcal{ALN}(\mathcal{O}_v)$, pour toute interprétation $\mathcal{I}$, nous avons $Q_{i_1}^{\mathcal{I}} \subseteq Q_{i_2}^{\mathcal{I}}$.

(⇒) Soient Q_{i_1} et Q_{i_2} deux conjonctions de vues satisfiables de $\mathcal{V}$. Nous montrons que si $Q_{i_1} \sqsubseteq Q_{i_2}$ alors Q_{i_2} réfère un sous-ensemble de vues apparaissant dans la description de Q_{i_1}.
Comme $\mathcal{V}$ est une terminologie primitive qui a été régularisée, déployée et normalisée, chaque vue V_{i_j} est une conjonction entre une description D_{i_j} et un concept atomique unique $\overline{V_{i_j}}$.
Supposons que $Q_{i_1} \equiv V_1 \sqcap \ldots \sqcap V_n$, où $V_i \in \mathcal{V}$ pour tout $i \in \{1, ..., n\}$
alors $Q_{i_1} \equiv (D_1 \sqcap \ldots \sqcap D_n) \sqcap (\overline{V_1} \sqcap \ldots \sqcap \overline{V_n})$.
Tout concept qui subsume Q_{i_1} doit contenir dans sa description une conjonction d'éléments appartenant à $\{\overline{V_1}, \ldots, \overline{V_n}\}$.
Autrement dit, tout concept qui subsume Q_{i_1} réfère un sous-ensemble de $\{\overline{V_1}, \ldots, \overline{V_n}\}$.

Comme tout concept $\overline{V_i}$ où $i \in \{1, ..., n\}$, est un concept atomique unique associé à une seule vue V_i, tout concept qui subsume Q_{i_1} réfère un sous-ensemble de $\{V_1, \ldots, V_n\}$.

□

L'énoncé du **lemme 4.1.3** est :

Soit $\mathcal{V}$ une terminologie dans $\mathcal{L}$. Soient $Q_{i_1} \equiv \sqcup_{j=1}^{n} C_j$ et $Q_{i_2} \equiv \sqcup_{k=1}^{p} C_k$, telles que $\forall j \in \{1, ..., n\}$ et $\forall k \in \{1, ..., p\}$, C_j et C_k sont des conjonctions de vues de $\mathcal{V}$ satisfiables.

$Q_{i_1} \sqsubseteq Q_{i_2}$ si et seulement si pour chaque chaque C_j dans Q_{i_1}, il existe C_k dans Q_{i_2} tel que $C_j \sqsubseteq C_k$.

Démonstration

($\Leftarrow$) Soient Q_{i_1} et Q_{i_2} deux disjonctions de conjonctions de vues de $\mathcal{V}$. Soit $Q_{i_1} \equiv \sqcup_{j=1}^{n} C_j$ et $Q_{i_2} \equiv \sqcup_{k=1}^{p} C_k$. Nous montrons que si pour chaque C_j dans Q_{i_1}, il existe C_k dans Q_{i_2} tel que $C_j \sqsubseteq C_k$ alors $Q_{i_1} \sqsubseteq Q_{i_2}$.

Pour chaque C_j, $j \in \{1, ..., n\}$, dans Q_{i_1}, il existe C_k, avec $k \in \{1, ..., p\}$, tel que $C_j \sqsubseteq C_k$. Ainsi tout C_j, $j \in \{1, ..., n\}$, dans Q_{i_1}, est subsumé par la disjonction des éléments de $\{C_1, \ldots, C_k\}$. Ainsi la disjonction $Q_{i_1} \equiv \sqcup_{j=1}^{n} C_j$ est subsumée par la disjonction $Q_{i_2} \equiv \sqcup_{k=1}^{p} C_k$.

($\Rightarrow$) Soient Q_{i_1} et Q_{i_2} deux disjonctions de conjonctions de vues de $\mathcal{V}$.

Supposons $Q_{i_1} \equiv \sqcup_{j=1}^{n} C_j$ et $Q_{i_2} \equiv \sqcup_{k=1}^{p} C_k$. Pour tout $j \in \{1, ..., n\}$ et pour tout $k \in \{1, ..., p\}$, C_i et C_k sont des conjonctions de vues. Supposons alors que pour $j \in \{1, ..., n\}$, $C_j \equiv \sqcap_{l=p_1}^{p_{n_j}} V_l$ et que pour $k \in \{1, ..., p\}$, $C_k \equiv \sqcap_{o=m_1}^{m_{n_k}} V_o$.

Nous montrons que, si $Q_{i_1} \sqsubseteq Q_{i_2}$ alors pour chaque C_j, $j \in \{1, ..., n\}$, de Q_{i_1}, il existe C_k, $k \in \{1, ..., p\}$, de Q_{i_2} tel que $C_j \sqsubseteq C_k$.

$Q_{i_1} \sqsubseteq Q_{i_2}$ alors tout C_j, avec $j \in \{1, ..., n\}$, de Q_{i_1} est subsumé par Q_{i_2}.

On a pour tout $j \in \{1, ..., n\}$, $C_j \equiv \sqcap_{l=p_1}^{p_{n_j}} V_l$ où chaque V_l, $l \in \{p_1, ..., p_{n_j}\}$, est une vue de $\mathcal{V}$. Comme $\mathcal{V}$ est une terminologie primitive régularisée, déployée et normalisée, pour tout $j \in \{1, ..., n\}$, $C_j \equiv (\sqcap_{l=p_1}^{p_{n_j}} D_l) \sqcap (\sqcap_{l=p_1}^{p_{n_j}} \overline{V_l})$ où tout $\overline{V_l}, l \in \{p_1, ..., p_{n_j}\}$ est un concept atomique.

Comme Q_{i_2} subsume C_j, avec $j \in \{1, ..., n\}$ alors Q_{i_2} subsume $\sqcap_{l=p_1}^{p_{n_j}} \overline{V_l}$. Ainsi soit Q_{i_2} est équivalent au concept $\top$, soit Q_{i_2} contient un concept C_k, $k = \{1, ..., p\}$, de la disjonction qui subsume $\sqcap_{l=p_1}^{p_{n_j}} \overline{V_l}$.

Comme Q_{i_2} est une disjonction de conjonction de vues de $\mathcal{V}$ et que $\mathcal{V}$ est une terminologie primitive régularisée, déployée et normalisée, Q_{i_2} ne peut pas être équivalent au concept $\top$. De même C_k de Q_{i_2}, où $k = \{1, ..., p\}$, subsume $\sqcap_{l=p_1}^{p_{n_j}} \overline{V_l}$ uniquement (lemme 4.1.2) si C_k réfère un sous-ensemble de vues de $\{V_{p_1}, ..., V_{p_{n_j}}\}$.

Ainsi, plus généralement, pour chaque C_i de Q_{i_1}, si $Q_{i_1} \sqsubseteq Q_{i_2}$, Q_{i_2} contient dans sa description un concept C_k qui réfère un sous-ensemble de vues de C_i. Autrement dit, selon le lemme 4.1.2, si $Q_{i_1} \sqsubseteq Q_{i_2}$ alors pour tout concept C_i de Q_{i_1}, il existe un concept C_k qui subsume C_i.

□

L'énoncé du **lemme 4.1.1** est rappelé ci-dessous :

Soient $\mathcal{V}$ une terminologie dans $\mathcal{L}$ et Q une requête dans $\mathcal{L}$, où $\mathcal{L} \in \{\mathcal{FL}_0(\mathcal{O}_v), \mathcal{ALN}(\mathcal{O}_v)\}$.

Soit $\{Q_1, \ldots, Q_n\}$, l'ensemble de toutes les réécritures conjonctives maximales de Q en

termes de $\mathcal{V}$.

Q' est une réécriture maximale de Q en termes de $\mathcal{V}$ si et seulement si $Q' \equiv \sqcup_{i=1}^{n} Q_i$.

La démonstration du lemme 4.1.1 est la suivante : *Démonstration*

($\Rightarrow$) Nous montrons que si Q' est une réécriture maximale et contenue de Q en termes de $\mathcal{V}$ alors Q' est l'union de toutes les réécritures conjonctives maximales et contenues de Q.

a) Pour montrer cela, nous supposons d'abord qu'un des éléments de $\{Q_1, \ldots, Q_n\}$ n'est pas une réécriture conjonctive maximale et contenue de Q, et montrons que $Q' \equiv (\sqcup_{i=1}^{n} Q_i)$ ne peut pas être une réécriture maximale et contenue de Q en termes de $\mathcal{V}$.
Supposons que Q_k, $k \in \{1, ..., n\}$ n'est pas une réécriture conjonctive maximale et contenue dans Q en termes de $\mathcal{V}$. Soit Q'_k est une réécriture conjonctive maximale et contenue de Q en termes de $\mathcal{V}$.
Alors il existe Q'' formée à partir de Q' en remplaçant Q_k par Q'_k, i.e. $Q'' \equiv (\sqcup_{i=1}^{k-1} Q_i) \sqcup Q'_k \sqcup (\sqcup_{i=k+1}^{n} Q_i)$.
On a alors $Q' \sqsubseteq Q''$ d'après le lemme 4.1.3. En effet, pour tout Q_i, $i \in \{1, ..., n\} \backslash \{k\}$, de Q', il existe Q_j, $j \in \{1, ..., n\} \backslash \{k\}$, tel que $Q_i \equiv Q_j$ et pour Q_k de Q', il existe Q'_k de Q'' tel que $Q_k \sqsubseteq Q'_k$.
On veut montrer que Q'' n'est pas équivalente à Q'. Pour cela, il suffit de montrer que $Q'' \not\sqsubseteq Q'$. Suposons que $Q'' \not\sqsubseteq Q'$, alors d'après le lemme 4.1.3 pour tout Q'_i dans Q'', il existe Q_i dans Q' tel que $Q'_i \sqsubseteq Q_i$.
Soit Q'_k dans Q'', cherchons Q_i dans Q' tel que $Q'_k \sqsubset Q_i$.
Soit Q_i de Q' tel que $Q'_k \sqsubseteq Q_i$.
Alors on a $Q_k \sqsubseteq Q'_k \sqsubseteq Q_i$ avec $Q'_k \not\equiv Q_k$.
Ainsi il y a une redondance dans Q', ce qui contredit l'hypothèse selon laquelle les réécritures sont non redondantes.
D'où $Q' \not\equiv Q''$ et Q' ne peut pas être maximale. Ainsi Q' est formée des réécritures conjonctives maximales de Q.

b) Pour terminer la preuve de la complétude du lemme 4.1.1, nous montrons que Q' est formée de l'union de toutes les réécritures conjonctives maximales de Q, i.e. de $\{Q_1, \ldots, Q_n\}$. Pour cela, nous supposons que Q' n'est pas formée de l'union de tous les éléments de $\{Q_1, \ldots, Q_n\}$.
Supposons que $Q' \equiv Q_1 \sqcup ... \sqcup Q_{k-1} \sqcup Q_{k+1} \sqcup ... Q_n$ et que Q' est une réécriture maximale et contenue de Q en termes de $\mathcal{V}$. Soit Q'' une description équivalente à $\sqcup_{i=1}^{n} Q_i$.
On alors $Q' \sqsubseteq Q''$ d'après le lemme 4.1.3.
Il reste alors à montrer que $Q' \not\equiv Q''$, soit que $Q'' \not\sqsubseteq Q'$.
Pour cela, on suppose que $Q'' \sqsubseteq Q'$. Alors d'après le lemme 4.1.3, on a pour tout Q_i de Q'', il existe Q_j de Q' tel que $Q_i \sqsubseteq Q_j$.
Soit Q_k de Q'', il doit exister Q'_k de Q' tel que $Q_k \sqsubseteq Q'_k$.
Or $\{Q_1, \ldots, Q_n\}$ est l'ensemble de toutes les réécritures conjonctives maximales de Q, non redondantes. Ainsi aucun Q_i, $i \in \{1, \ldots, n\} \backslash \{k\}$ ne subsume Q_k ou n'est subsumé par Q_k. Par conséquent, il n'existe pas de Q'_k dans Q' qui subsume Q_k.
Ainsi si Q' n'est pas formée de toutes les réécritures conjonctives maximales (non équivalente) de Q alors Q' une réécriture maximale et contenue de Q en termes de $\mathcal{V}$.
Ainsi Q' est formée de l'union de toutes les réécritures conjonctives de $\{Q_1, \ldots, Q_n\}$.

($\Leftarrow$) Nous montrons que si Q' est l'union de toutes les réécritures conjonctives maximales et

contenues de Q alors Q' est une réécriture maximale et contenue de Q.
Pour montrer cela, nous supposons que Q' n'est pas maximale, i.e. il existe Q_1 tel que $Q' \sqsubseteq Q_1 \sqsubseteq Q$ et $Q_1 \not\equiv Q'$.
Selon les lemmes 4.1.2 et 4.1.3, Q_1 subsume Q' si et seulement si pour chaque élément Q_i de la disjonction Q', il existe Q_1^j dans Q_1 qui subsume Q_i, i.e. $Q_i \sqsubseteq Q_1^j$.
Cependant, par définition, tous les Q_i sont maximaux. Alors Q_1 doit comporter dans sa description un élément supplémentaire dans la disjonction Q_1^{add} qui est subsumé par Q.
Or par hypothèse, Q' est l'union de toutes les réécritures conjonctives maximales de Q.
Alors Q_1^{add} ne peut être que redondant par rapport à $\{Q_1, \ldots, Q_n\}$.
Ainsi il n'existe pas de Q_1 qui subsume Q' et Q' est maximale.

□

A.4 Résolution de *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{L}$)*

Le **lemme 4.1.4** est énoncé ci-dessous :
Soit $Q' \equiv \sqcap_{i=1}^{n} V_i$ tel que $Q' \sqsubseteq Q$.
Q' est une réécriture conjonctive maximale de Q en termes de $\mathcal{V}$ si et seulement si pour chaque concept Q'' obtenu en supprimant dans Q' une vue V_j, avec $j \in \{1, ..., n\}$, on a $Q'' \not\sqsubseteq Q$.

Démonstration

($\Rightarrow$) Supposons que $Q' \equiv V_1 \sqcap \ldots \sqcap V_n$ est une réécriture conjonctive maximale et contenue de Q en termes de $\mathcal{V}$.
Nous montrons que pour n'importe quel concept Q'' obtenu en supprimant une vue V_j de Q', avec $j \in \{1, ..., n\}$, alors on a $Q'' \not\sqsubseteq Q$.
Supposons que $Q'' \equiv V_1 \sqcap ... \sqcap V_{j-1} \sqcap V_{j+1} \sqcap ... \sqcap V_n$, obtenue en enlevant de Q' une vue V_j, avec $j \in \{1, ..., n\}$, est subsumé par Q (i.e. $Q'' \sqsubseteq Q$). Nous avons également $Q' \sqsubseteq Q''$. Nous montrons alors que Q' n'est pas une réécriture conjonctive, maximale et contenue de Q.
Pour montrer cela, nous montrons que Q'' n'est pas équivalente à Q'.
Soient $Q' \equiv V_1 \sqcap ... \sqcap V_n$ and $Q'' \equiv V_1 \sqcap ... \sqcap V_{j-1} \sqcap V_{j+1} \sqcap ... \sqcap V_n$. Selon l'hypothèse du monde ouvert,
$Q'' \equiv (D_1 \sqcap \ldots \sqcap D_{(j-1)} \sqcap D_{(j+1)} \sqcap \ldots \sqcap D_n) \sqcap (A_1 \sqcap \ldots \sqcap A_{(j-1)} \sqcap A_{(j+1)} \sqcap \ldots \sqcap A_n)$
$Q' \equiv (D_1 \sqcap \ldots \sqcap D_{(j-1)} \sqcap D_{(j+1)} \sqcap \ldots \sqcap D_n) \sqcap (A_1 \sqcap \ldots \sqcap A_{(j-1)} \sqcap A_{(j+1)} \sqcap \ldots \sqcap A_n) \sqcap D_i \sqcap A_i$.
Q' et Q'' ne sont pas équivalentes car A_i ne subsume pas $A_1 \sqcap \ldots \sqcap A_{(j-1)} \sqcap A_{(j+1)} \sqcap \ldots \sqcap A_n$.
En effet A_i est un concept atomique associé à une et une seule vue V_i qui apparaît elle-même une seule fois dans Q'.
Ainsi il existe Q'' tel que $Q'' \not\equiv Q'$ et $Q' \sqsubseteq Q'' \sqsubseteq Q$. Ainsi Q' ne peut pas être une réécriture conjonctive maximale et contenue de Q en termes de $\mathcal{V}$.
($\Leftarrow$) Nous devons montrer que
si pour tout $Q'' \equiv V_{i_1} \sqcap \ldots \sqcap V_{i_{n-1}}$, tel que $\{V_{i_1}, ..., V_{i_{n-1}}\} \subseteq \{V_1, ..., V_n\}$, nous avons $Q'' \not\sqsubseteq Q$, alors $Q' \equiv V_1 \sqcap \ldots V_n$ est une réécriture conjonctive maximale et contenue de Q en termes de $\mathcal{V}$.
Autrement dit nous devons montrer qu'il n'y a pas de Q'' telle que $Q' \sqsubseteq Q''$ et $Q'' \sqsubseteq Q$ et

$Q' \not\equiv Q''$.
Nous désignerons par (*) l'hypothèse suivante : pour tout $Q'' \equiv V_{i_1} \sqcap \ldots \sqcap V_{i_{n-1}}$, tel que $\{V_{i_1}, ..., V_{i_{n-1}}\} \subseteq \{V_1, ..., V_n\}$, nous avons $Q'' \not\sqsubseteq Q$.
Par hypothèse $Q' \equiv V_1 \sqcap \ldots V_n$ est subsumé par Q ($Q' \sqsubseteq Q$).
Supposons que Q' n'est pas une réécriture conjonctive maximale et contenue de Q en termes de $\mathcal{V}$. Alors il existe Q_1 une conjonction de vues en termes de $\mathcal{V}$, telle que $Q' \sqsubseteq Q_1 \sqsubseteq Q$ et $Q' \not\equiv Q_1$. Selon le lemme 4.1.2, Q_1 subsume strictement Q' si Q_1 réfère uniquement un strict sous-ensemble de vues de Q' et dans ce cas $Q_1 \not\equiv Q'$.
Soient $Q' \equiv V_1 \sqcap ... \sqcap V_n$ et $Q_1 \equiv V_1 \sqcap ... \sqcap V_{j-1} \sqcap V_{j+1} \sqcap ... \sqcap V_n$.
$Q_1 \equiv (D_1 \sqcap \ldots \sqcap D_{(j-1)} \sqcap D_{(j+1)} \sqcap \ldots \sqcap D_n) \sqcap (A_1 \sqcap \ldots \sqcap A_{(j-1)} \sqcap A_{(j+1)} \sqcap \ldots \sqcap A_n)$
et $Q' \equiv (D_1 \sqcap \ldots \sqcap D_{(j-1)} \sqcap D_{(j+1)} \sqcap \ldots \sqcap D_n) \sqcap (A_1 \sqcap \ldots \sqcap A_{(j-1)} \sqcap A_{(j+1)} \sqcap \ldots \sqcap A_n) \sqcap D_i \sqcap A_i$.
and A_i is an atomic concept.
Ainsi il existe Q_1 qui réfère un sous-ensemble de $\{V_1, ..., V_n\}$ et qui est subsumé par Q. Alors chaque conjonction de vues qui réfère des surensembles de vues de Q_1 est subsumé par Q. Ceci réfute l'hypothèse (*) selon laquelle il n'y a pas de sous-ensemble de $\{V_1, ..., V_n\}$ dont la conjonction de leurs éléments est subsumé par Q.
Ainsi Q' est une réécriture conjonctive maximale et contenue de Q en termes de $\mathcal{V}$.

□

A.5 Formes des réécritures dans $\mathcal{FL}_0(\mathcal{O}_v)$

L'énoncé du **lemme 4.2.1** est :

Soit le problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)* où $Q \equiv \forall w.P$.

Soit l la cardinalité du plus grand des ensembles de valeurs qui apparaît dans les vues ou la requête.

Soit $Q' \equiv V_1 \sqcap \ldots \sqcap V_k$.

Si $\{V_1, \ldots V_k\}$ est un sous-ensemble **minimal** de $\mathcal{V}$ tel que $w \in V_{Q'}(P)$ alors une des conditions suivantes est vérifiée :

a) $k = 1$, $P = A$ et $\forall w.P \in V_{i_1}$ ou bien,

b) $1 \leq k \leq l + 1$, $P = E$ et $\{V_1 ... V_k\}$ est tel que : pour $j \in \{1, \ldots, k\}$, $\forall w.E_j \in V_j$, et $\cap_{j=1}^{k} E_j \subseteq E$.

Démonstration

La preuve de ce lemme est donnée par la preuve du lemme 4.2.2 en annexe A.6. □

L'énoncé du **théorème 4.2.1** est rappelé ci-dessous :

Soit le problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{FL}_0(\mathcal{O}_v)$)* où $Q \equiv \forall w.P$.

Soient n le nombre d'atomes dans Q et l la cardinalité du plus grand des ensembles de valeurs apparaissant dans les vues ou la requête.

Les réécritures conjonctives maximales de Q en termes des vues $\mathcal{V}$, dans la logique $\mathcal{FL}_0(\mathcal{O}_v)$, contiennent au plus $n * (l + 1)$ vues de $\mathcal{V}$.

Démonstration

La justification du théorème 4.2.3 vient de l'observation que si Q' est une réécriture maximale et conjonctive de Q alors $Q' \sqsubseteq \forall w.P$, pour chaque $\forall w.P \in Q$. Ainsi, une borne supérieure de la taille de Q' peut être calculée à partir du nombre maximal de vues nécessaires pour réécrire un atome de$Q$. A partir du théorème 3.3.1 et du lemme 4.1.4, pour calculer les réécritures d'un atome $\forall w.P \in Q$, il est nécessaire de considérer des concepts Q' faits de sous-ensembles minimaux de $\mathcal{V}$ tels que $w \in V_{Q'}(P)$. Le pire des cas pour réécrire un atome est lorsque cet atome est une restriction de valeur vers un ensemble de valeurs et qu'on dispose uniquement de vues subsumées par une restriction de valeurs vers des ensembles de valeurs de taille l et que ces ensembles ne se distinguent que par *une seule* valeur. Alors $l + 1$ vues peuvent être nécessaires pour réécrire l'atome.
Par exemple supposons que $Q \equiv \forall numDepartement.\{63, 45, 87\}$ et
$V_1 \sqsubseteq \forall numDepartement.\{43, 44, 46, 47\}$
$V_2 \sqsubseteq \forall numDepartement.\{43, 44, 46, 48\}$
$V_3 \sqsubseteq \forall numDepartement.\{43, 44, 47, 48\}$
$V_4 \sqsubseteq \forall numDepartement.\{43, 46, 47, 48\}$
$V_5 \sqsubseteq \forall numDepartement.\{44, 46, 47, 48\}$.
Ici l est égal à 4 et la conjonction des $l + 1 = 5$ vues est nécessaire pour réécrire $Q \equiv \forall numDepartement.\{63, 45, 87\}$.

□

A.6 Formes des réécritures dans $\mathcal{ALN}(\mathcal{O}_v)$

L'énoncé du **lemme 4.2.2** est :

Soit le problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)* où $Q \equiv \forall w.P$.

Soient l la cardinalité du plus grand des ensembles de valeurs qui apparaît dans les vues ou la requête, et p la profondeur maximale[18]

Soit $Q' \equiv V_{i_1} \sqcap \ldots \sqcap V_{i_k}$ où $\{V_{i_1}, \ldots V_{i_k}\}$ est un sous-ensemble **minimal** de $\mathcal{V}$ tel que $w \in V_{Q'}(P) \cup E(Q')$ alors une des conditions suivantes est vérifiée :

a) $k = 1$,
  a.1) $P \in \{A, \neg A\}$ et $\forall w.P \in V_{i_1}$ ou,
  a.2) $P = (\geq nR)$ et $\forall w.(\geq pR) \in V_{i_1}$ avec $p \geq n$ ou,
  a.3) $P = (\leq nR)$ et $\forall w.(\leq pR) \in V_{i_1}$ avec $p \leq n$

b) $1 \leq k \leq l + 1$,
  b.1) $P = E$ et $\{V_{i_1} ... V_{i_k}\}$ est tel que : pour $j \in [1, k]$, $\forall w.E_{i_j} \in V_{i_j}$, et $\cap_{j=i_1}^{i_k} E_j \subseteq E$.
  b.2) $P = (\leq n\, R_v)$, avec $R_v \in \mathcal{R}_v$, $\{V_{i_1} ... V_{i_k}\}$ est tel que : pour $j \in [1, k]$, $\forall w.E_{i_j} \in V_{i_j}$, et $|\cap_{j=i_1}^{i_k} E_j| \leq n$.

c) $1 \leq k \leq l + p$ et il existe un préfixe $w'v$ de w tel que $\forall w'.(\leq 0v) \in \sqcap_{j=i_1}^{i_k} V_j$.

Démonstration

[18] la profondeur d'un atome $\forall w.P$ est égal à la longueur du mot w, i.e. le nombre de rôles dans w.

Nous avons $Q' \equiv V_{i_1} \sqcap \ldots \sqcap V_{i_k}$ tel que $\{V_{i_1}, \ldots V_{i_k}\}$ un sous-ensemble minimal de $\mathcal{V}$ et tel que $w \in V_{Q'}(P) \cup E(Q')$.

- $w \in V_{Q'}(P)$
  - Si $P \in \{A, \neg A\}$ alors selon les règles de normalisation, une des vues contient $\forall w.P$. Ainsi comme l'ensemble $\{V_{i_1}, \ldots V_{i_k}\}$ est minimal, une vue qui contient $\forall w.P$ est suffisante pour réécrire $Q \equiv \forall w.P$ et $k = 1$.
  - Si $P = (\geq nr)$ alors selon les règles de normalisation, une des vues contient $\forall w.(\geq mr)$, avec $m \geq n$. Ainsi comme l'ensemble $\{V_{i_1}, \ldots V_{i_k}\}$ est minimal, une vue qui contient $\forall w.(\geq mr)$ est suffisante pour réécrire $Q \equiv \forall w.P$ et $k = 1$.
  - Si $P = (\leq\ n\ r)$, i.e., $\forall w.(\leq\ m\ r)$ avec $m \leq n$ appartient à Q' et comme ci-dessus une vue est suffisante pour réécrire $Q \equiv \forall w.P$ et $k = 1$.
  - Si $P = (\leq\ n\ r)$ et $r \in \mathcal{R}_v$ (*), selon les règles de normalisation 8) et 9), alors on peut trouver ($k \geq 1$) vues qui contiennent respectivement un conjonct $\forall w.r.E_{i_j}$ tel que $\cap_{j=1}^{k} E_{i_j} \subseteq E'$ et $card(E') \leq n$. Le pire des cas intervient quand les ensembles E_{i_j} ont pris deux à deux, une seule valeur distincte et, nous devons inférer $\forall w.(\leq\ n\ r)$, avec $n = 0$. Si le nombre maximal de valeurs dans les ensembles E_{i_j} est l alors dans le pire des cas, $(l+1)$ ensembles de valeurs sont nécessaires pour obtenir l'ensemble vide. De cette façon, si $r \in \mathcal{R}_v$ et $P = (\leq\ n\ r)$, dansle pire des cas, $(l+1)$ vues sont nécessaires pour réécrire $Q \equiv \forall w.(\leq\ n\ r)$ et $1 \leq k \leq (l+1)$.
  - Si $P = E$, selon la règle de normalisation 8, $k \geq 1$ vues contiennent respectivement un conjonct $\forall w.E_{i_j}$ tel que $\cap_{j=1}^{k} E_{i_j} \subseteq E'$ et $E' \subseteq E$. Le pire des cas arrive quand les ensembles E_{i_j} ont pris deux à deux, une seule valeur distincte, et qu'on doit inférer $\forall w.E$, avec $E = \emptyset$. Si le nombre maximal de valeurs dans les ensembles E_{i_j} est l alors $(l+1)$ ensembles de valeurs sont nécessaires pour obtenir l'ensemble vide. AInsi si $P = E$, dans le pire des cas, $(l+1)$ vues sont nécessaires pour réécrire $Q \equiv \forall w.E$ et $1 \leq k \leq (l+1)$.
- $w \in E(Q')$.
  Supposons que $w = r_1 u$. Dans le pire des cas, $Q' \sqsubseteq \leq\ 0\ r_1$ où le concept $\leq\ 0\ r_1$ peut être induit par la conjonction des $(p+1)$ vues suivantes :
  $V_{i_1} \sqsubseteq \forall r_1.r_2.\ldots.r_p.(\leq qr)$
  $V_{i_2} \sqsubseteq \forall r_1.r_2.\ldots.r_p.(\geq mr)$, with $q < m$
  $V_{i_3} \sqsubseteq \forall r_1.r_2.\ldots.r_{p-1}.(\geq 1r_k)$,
  ...
  $V_{i_{(p+1)}} \sqsubseteq \forall r_1.(\geq 1r_2)$.
  Cette séquence de concepts a été mise en évidence dans [53]. Cependant on peut aussi obtenir une conjonction de vues qui , comme V_{i_1}, est subsumée par $\forall w'.r_1.r_2.\ldots.r_p.(\leq\ q\ r)$, si $r \in \mathcal{R}_v$, comme vu en (*). De telles conjonctions de vues peuvent jouer le rôle de la vue V_{i_1}. Ainsi, au plus $l+1+p+1$ vues sont nécessaires pour obtenir une réécriture $Q' \sqsubseteq \leq\ 0\ r_1$. De cette façon, si $w \in E(Q')$, dans le pire des cas, $(1 \leq k \leq l+p+2)$ vues sont nécessaires pour obtenir une réécriture $Q' \sqsubseteq \forall w'.(\leq\ 0\ v)$

Après regroupement des différents type de réécritures par leur taille, on retrouve les différents cas du lemme. □

A.7 Complétude le l'algorithme de réécriture

L'énoncé du **théorème 4.3.1** est :
Soit un problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)*.
L'algorithme *CalculReec* de calcul des réécritures conjonctives maximales de Q est juste et complet.

Démonstration
Preuve de la complétude de *CalculReec*.
Soit un problème *conj_rewrite(Q,$\mathcal{V}$,$\mathcal{ALN}(\mathcal{O}_v)$)*.
Soit $Q \equiv \sqcap_{i=1}^{n} \forall w_i.P_i$.
Soit Q' une réécriture conjonctive maximale de Q telle que $Q' \equiv V_1 \sqcap V_2 \sqcap \ldots \sqcap V_m$.
On veut montrer que Q' appartient à la sortie de l'algorithme *CalculReec*, i.e. que l'algorithme de réécriture *CalculReec* calcule effective toutes les réécritures conjonctives maximales.
Q' est une réécriture maximale de Q alors $Q' \sqsubseteq \forall w_i.P_i, \forall i \in \{1, \ldots, n\}$.
Soit $Q'_i \equiv V_{i_1} \sqcap \ldots \sqcap V_{i_m}$ où $\{V_{i_1}, \ldots, V_{i_m}\}$ est une sous-ensemble minimal de $\{V_1, \ldots, V_m\}$ tel que $Q'_i \sqsubseteq \forall w_i.P_i$ avec $i \in \{1, \ldots, n\}$.
Ainsi d'après le lemme 4.2.2, Q'_i vérifie un des cas (a), (b) ou (c), et $Q'_i \in B(w_i, P_i)$ d'après l'algorithme textitCalculReec, et par appel de l'algorithme 2, *Construire_Panier*.
Par conséquent, la conjonction des Q'_i tels que définis précédemment, i.e. la conjonction $\sqcap_{i=1}^{n} Q'_i$ appartient au produit cartésien des paniers $B(w_i, P_i)$.
De plus on a $Q' \equiv \sqcap_{i=1}^{n} Q'_i$ puisque Q' est une réécriture conjonctive maximale. Ainsi Q' appartient au produit cartésien des paniers $B(w_i, P_i)$.

□

A.8 Formulation ensembliste de la réécriture

L'énoncé du **lemme 5.1.1** est :
Soient n un entier, w et wR des mots et E un ensemble.
Soit deux problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,wR)*.
Soit $b = \sqcap_{j=i_1}^{i_k} V_j$ avec $k \geq 1$.
1. b est une solution de *E_conj_rewrite(E,w)*
   ssi $\{V_{i_1}, ..., V_{i_k}\} \subseteq \mathcal{V}_w$ et $\{E_{i_1}, ..., E_{i_k}\} \in S_1(w, E)$
2. b est une solution de *N_conj_rewrite(n,wR)*
   ssi $\{V_{i_1}, ..., V_{i_k}\} \subseteq \mathcal{V}_{w.R}$ et $\{E_{i_1}, ..., E_{i_k}\} \in S_2(w.R, n)$

Démonstration

Soient les problèmes *E_conj_rewrite(E,w)* et *N_conj_rewrite(n,wR)*.
$\Rightarrow$:
– Soit b une solution de *E_conj_rewrite(E,w)*
  Alors b est formée d'un sous-ensemble minimal M de $\mathcal{V}_w$ telle que $b \equiv \forall w.E'$ et $E' \subseteq E$ d'après le théorème 3.4.1.

Soit X l'ensemble des E_j correspondants aux vues de M. Alors $\cap\{x \in X\} = E'$. De plus comme b est formé d'un sous-ensemble minimal de $\mathcal{V}_w$, il n'existe pas de sous-ensemble X' de X tel que $\cap\{x' \in X'\} = E''$. Par conséquent $X \in S_1(w, E)$.

– Soit b une solution de *N_conj_rewrite(n,wR)*
Alors b est formée d'un sous-ensemble minimal M de $\mathcal{V}_{wR}$ telle que $b \equiv \forall w.E'$ et $|E'| \leq n$ d'après le théorème 3.4.1.
Soit X l'ensemble des E_j correspondants aux vues de M. Alors $\cap\{x \in X\} = E'$. De plus comme b est formé d'un sous-ensemble minimal de $\mathcal{V}_w$, il n'existe pas de sous-ensemble X' de X tel que $\cap\{x' \in X'\} = E''$ et $|E''| \leq n$. Par conséquent $X \in S_2(wR, n)$.

$\Leftarrow$:

– Soit un atome $\forall w.E$ et un ensemble de vues $\{V_{i_1}, ..., V_{i_p}\} \subseteq \mathcal{V}_w$ telles que $\{E_{i_1}, ..., E_{i_p}\} \in S_1(w, E)$. Ainsi $\{E_{i_1}, ..., E_{i_p}\}$ est un sous-ensemble minimal de F_w tel que $\cap_{j=1}^{p} E_{i_j} \subseteq E$. Alors $\{V_{i_1}, ..., V_{i_p}\}$ est un sous-ensemble minimal de $\mathcal{V}$ dont la conjonction de ses éléments est subsumée par $\forall w.E$. Ainsi $\sqcap_{j=i_1}^{i_p} V_j$ est une solution de *E_conj_rewrite(E,w)*.
– Soit un atome $\forall w.(\leq\ nR)$ et un ensemble de vues $\{V_{i_1}, ..., V_{i_p}\} \subseteq \mathcal{V}_{w.R}$ telles que $\{E_{i_1}, ..., E_{i_p}\} \in S_2(w.R_v, n)$. Ainsi $\{E_{i_1}, ..., E_{i_p}\}$ un sous-ensemble minimal de $F_{w.R}$ tel que $|\cap_{j=1}^{p} E_{i_j}| \leq n$. Alors $\{V_{i_1}, ..., V_{i_p}\}$ un sous-ensemble minimal de $\mathcal{V}_{w.R_v}$ dont la conjonction des ses éléments est subsumée par $\forall w.(\leq\ nR)$. Donc $\sqcap_{j=i_1}^{i_p} V_j$ est une solution de *N_conj_rewrite(n,wR)*.

□

A.9 Un cadre de découverte des connaissances dans les bases de données

L'énoncé du **lemme 5.3.1** est :
P_1 est anti-monotone par rapport à $\subseteq$.

Démonstration

Soit $X \in \mathcal{L}$ tel que $P_1(X, E)$ est vrai. On doit montrer que pour tout $Y \in \mathcal{L}$ tel que $Y \subseteq X$, $P_1(Y, E)$ est vrai.
Soit $X = \{E_{i_1}, ..., E_{i_n}\}$, $Y = \{E_{j_1}, ..., E_{j_k}\}$, avec $Y \subseteq X$
Comme $P_1(X, E)$ est vrai, $\cap_{j=1}^{n} E_{i_j} \not\subseteq E$.
Comme $Y \subseteq X$ alors $\cap_{j=1}^{n} E_{i_j} \subseteq \cap_{q=1}^{k} E_{j_q}$,
et donc $\cap_{j=1}^{n} E_{i_j} \not\subseteq E$ ou de manière équivalente, $P_1(Y, E)$ est vrai.

□

L'énoncé du **théorème 5.3.1** est
Soit le problème *E_conj_rewrite(E,w)*.
$S_1(w, E)$ = $\mathcal{B}d^{-}(IEE)$

Démonstration

Soit IEE = $\{X \in \mathcal{L}_w$ tel que $P_1(E, X)$ est vrai $\}$. $\mathcal{B}d^+$(IEE) regroupe les motifs les plus spécifiques qui vérifient $Pred_1$. La bordure négative $\mathcal{B}d^-(IEE)$ regroupe les motifs les plus généraux qui ne vérifient pas P_1, i.e. les plus petits sous-ensembles de $\mathtt{F}_w$ dont l'intersection des éléments est incluse dans E, ce qui correspond trivialement à $S_1(w, E)$. □

L'énoncé du lemme 5.3.2 est rappelé ci-dessous :
P_2 est anti-monotone selon $\subseteq$.

Démonstration

Démonstration du lemme 5.3.2
Soit $X \in \mathcal{L}_w$ tel que $P_2(n, X)$ est vrai. Nous devons montrer que pour tout $Y \in \mathcal{L}_w$ tel que $Y \subseteq X$, $P_2(n, Y)$ est vrai.
Soit $X = \{E_{i_1}, ..., E_{i_n}\}$, $Y = \{E_{j_1}, ..., E_{j_k}\}$, avec $Y \subseteq X$.
Comme $P_2(n, X)$ est vrai, $|\cap_{j=1}^{n} E_{i_j}| > n$.
Comme $Y \subseteq X$ alors $\cap_{j=1}^{n} E_{i_j} \subseteq \cap_{q=1}^{k} E_{j_q}$,
et $|\cap_{q=1}^{k} E_{i_j}| > n$ et donc $P_2(n, Y)$ est vrai. □

L'énoncé du **théorème 5.3.2** est :
Soit le problème *N_conj_rewrite(n,w)*.
$S_2(w, n)$ = $\mathcal{B}d^-(IEN)$

Démonstration

Soit IEN = $\{X \in \mathcal{L}_w$ tel que $P_2(n, X)$ est vrai $\}$. $\mathcal{B}d^+(IEN)$ rassemble les motifs les plus spécifiques qui vérifient P_2. La bordure négative $\mathcal{B}d^-(IEN)$ rassemble les motifs les plus généraux qui ne vérifient pas P_2, i.e., les plus petits sous-ensembles de $\mathcal{L}_w$ dont la cardinalité de l'intersection des éléments est supérieure ou égale à n, ce qui correspond à l'ensemble $S_2(w, n)$. □

Annexe B

Prototype

B.1 Description de la base de données

Le schéma conceptuel de cette base de données est traduit en relationnel comme indiqué ci-après. Le schéma de base de données $\mathcal{B}$ est :

$\mathcal{B} = \{T\_Vues\_Concept, T\_Vues\_Valeurs, T\_Q\_Concept, T\_Q\_Valeurs, T\_Bucket\}$ où

- T_Vues_Concept représente les vues contraintes par un atome menant à un concept.
- T_Vues_Valeurs représente les vues contraintes par un atome menant à un ensemble de valeurs.
- T_Q_Concept représente les atomes de la requête contraints par un concept.
- T_Q_Valeurs désigne les atomes de la requête contraints par un ensemble de valeurs.
- T_Bucket liste les réécritures de chaque atome d'une requête.

Les relations sont définies sur l'univers des attributs $\mathcal{U} = \{Mot, Valeurs, Vue, Concept, Atome, K\_atome, ConjonctionVues\}$.

Les schémas de relations sont alors définis comme suit :

- schéma(T_Vues_Concept)=$\{Mot, Concept, Vue\}$
- schéma(T_Vues_Valeurs)=$\{Mot, Valeurs, Vue\}$
- schéma(T_Q_Concept)=$\{K\_atome, Mot, Concept\}$
- schéma(T_Q_Valeurs)=$\{K\_atome, Mot, Valeurs, Vue\}$
- schéma(T_Bucket)=$\{Atome, ConjonctionVues\}$

Les attributs $Mot, Concept, Vue$ sont des chaînes de caractères. Les attributs $Valeurs, ConjonctionVues$ sont des tables imbriquées (nestedtable), et $Atome$ et K_atome sont des entiers. Plus précisément, l'attribut Mot désigne une succession de rôles d'un atome. Les attributs $Valeurs$ et $ConjonctionVues$ représentent respectivement des ensembles de valeurs pour la représentation de contraintes de valeurs et des ensembles de noms de vues pour spécifier les réécritures d'un panier. Comme la taille de ces ensembles ne peut pas être fixée, ils sont représentés à l'aide des tables imbriquées (nested table).

Les attributs $Mot, Concept$ constituent la clé primaire de T_Vues_Concept. Les attributs $Mot, Valeurs$ constituent la clé primaire de T_Vues_Valeurs. Les attributs $Atome, ConjonctionVues$ forment la clé primaire de T_Bucket. L'attribut K_atome est une clé primaire pour T_Q_Concept et T_Q_Valeurs. tandis que $Atome$ de la relation T_Bucket est

une clé étrangère sur K_atome [19].

Les tables B.1, B.2 et B.3 illustrent la base de données avec des exemples de vues (table B.1), une requête (table B.2) et des paniers (table B.3).

Mot	$Concept$	Vue
$r_1.r_2.r_3$	C_1	V_1
$r_8.r_7$	C_2	V_1
$r_1.r_2.r_3$	C_1	V_2
$r_7.r_{10}.r_{12}$	C_3	V_3
$r_8.r_7$	C_2	V_3
...	...	...

Mot	$Valeurs$	Vue
$r_3.r_4$	$\{1,2,3,4\}$	V_1
$r_5.r_2.r_1$	$\{m_1,m_2,m_3\}$	V_1
$r_2.r_5$	$\{2,3,4,5,6\}$	V_2
$r_5.r_2.r_1$	$\{m_2,m_5,m_8\}$	V_2
$r_2.r_5$	$\{1,2,3,7,9\}$	V_3
...	...	...

TAB. B.1 – T_Vues_Valeurs

K_atome	Mot	$Concept$
1	$r_1.r_2.r_3$	C_1
4	$r_1.r_{10}.r_{12}$	C_3

K_atome	Mot	$Valeurs$
2	$r_5.r_2.r_1$	$\{m_2,m_3\}$
3	$r_3.r_4$	$\{7,8\}$

TAB. B.2 – T_Q_Valeurs

$Atome$	$ConjonctionVues$
1	V_1, V_2, V_3
1	V_4, V_7, V_8
2	V_2, V_{10}
...	...

TAB. B.3 – T_Bucket : Réécritures des atomes

Nous nous intéressons maintenant au module *ComputeBucket* (B1) pour construire les paniers d'une requête.

B.2 Moteur de réécriture *ComputeBucket*

Le moteur de réécriture *ComputeBucket* permet de calculer le panier de chaque atome de Q, i.e. les réécritures conjonctives maximales de chaque atome de Q, à partir de T_Q_Concept et T_Q_Valeurs. Il s'appuie sur le lemme 4.2.1 qui stipule que pour un atome $\forall w.P$:

- si P est un concept atomique $P = A$, alors les réécritures conjonctives maximales de l'atome $\forall w.A$ sont les vues qui contiennent dans leur description, exactement cet atome.
- si P est un ensemble de valeurs $P = E$, alors les réécritures conjonctives maximales de l'atome $\forall w.E$ sont les solutions de ***E_conj_rewrite(w,E)***.

[19] Il y a dépendance d'inclusion entre $Atome$ et K_atome : $Atome \subseteq K\_atome$.

Les entrées et les sorties de *ComputeBucket* sont les suivantes.
Entrée : une requête Q : T_Q_Concept et T_Q_Valeurs ;
les vues $\mathcal{V}$: T_Vues_Concept et T_Vues_Valeurs ;
Sortie : $Paniers$: T_Bucket
ComputeBucket fonctionne comme suit.
Soit a est un tuple de T_Q_Concept ou de T_Q_Valeurs.

La première partie de l'algorithme (ligne 4 de l'algorithme 3) consiste à trouver les vues qui contiennent exactement l'atome $\forall a[mot].a[concept]$ si l'atome a est contraint par un concept (ligne 9 de l'algorithme 3). Chaque vue ainsi identifiée est ensuite insérée dans T_Bucket (ligne 10).

La deuxième partie de l'algorithme consiste à trouver, si l'atome a est contraint par un ensemble de valeurs, les conjonctions de vues qui sont subsumées par $\forall a[mot].a[valeurs]$, avec $E = a[valeurs]$ (ligne 15 de l'algorithme 3). Dans un premier temps, les vues candidates à la réécriture de a sont stockées dans une table $Cand$ (lignes $17 - 24$). Ensuite, on calcule effectivement les conjonctions de vues subsumées par $\forall a[mot].a[valeurs]$, grâce à l'algorithme *ComputeEConj_1* décrit en sous-section B.3.1 (ligne 25) ou l'application *ComputeEConj_2* décrite succintement en chapitre 6. *ComputeEConj_1* stocke ces conjonctions de vues dans la table B sous forme d'ensembles de vues, appelés $motifs$. *ComputeEConj_2* renvoie les éléments de la bordure négatives sous forme d'un fichier texte qui sera lu pour mettre à jour la table B. Ensuite, la table B est parcourue (lignes $26 - 28$) afin d'insérer ces conjonctions dans le panier de $\forall a[mot].a[concept]$.

B.3 Principe d'APriori

Les deux implémentations de *ComputeEConj* sont des adaptations de l'algorithme de découverte de connaissances APriori. Le principe d'APriori est décrit ci-dessous à l'aide de la figure B.1.

Dans cette figure, on suppose disposer au niveau 1 de 5 motifs fréquents : $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{E\}$. On veut générer les motifs fréquents de niveau 2. Premièrement, on génère les candidats de niveau 2 à partir des motifs fréquents de niveau 1. Pour cela on associe à chaque item I de l'arbre au niveau k, chacun des ses "frères" plus grands que I dans l'ordre lexicographique, i.e. les items de son niveau ayant le même père que I, plus grand que I. Par exemple pour l'item B, on considère uniquement ses frères C, D et E pour obtenir les candidats de niveau 2. Dans la figure B.1, au niveau 2, on obtient les motifs $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{A, E\}$, $\{B, C\}$, $\{B, D\}$, $\{B, E\}$, $\{C, D\}$, $\{C, E\}$, $\{D, E\}$.

Deuxièmement on élague les candidats, c'est-à-dire on supprime les candidats dont un sous-ensemble a été identifié comme étant non fréquent. Tous les sous-ensembles des candidats de niveau 2, sont les motifs de niveau 1 qui sont tous fréquents. Aussi aucun candidat de niveau 2 n'est supprimé par élagage.

Finalement, on vérifie si le prédicat est vérifié par les candidats de niveau 2. Dans la figure B.1, le test du prédicat a conduit à la suppression des motifs $\{A, E\}$, $\{B, C\}$, $\{B, E\}$, $\{C, D\}$, $\{C, E\}$, $\{D, E\}$. Ainsi les motifs fréquents de niveau 2 sont $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, et $\{B, D\}$. Notons que les motifs éliminés à cette étape appartiennent à la bordure négative de la théorie et qu'ils forment la sortie de l'algoritme *ComputeEConj_1*.

Algorithme 3 *ComputeBucket*

Entrée(s) : T_Q_Concept, T_Q_Valeurs, T_Vues_Concept et T_Vues_Valeurs
Sortie(s) : T_Bucket

/* VARIABLES*/
$compt$: Integer
B($motifs$) : Table d'ensembles de vues
/* PREMIERE PARTIE : CALCUL DES VUES SUBSUMEES PAR $\forall a[mot].a[concept]$*/
pour chaque tuple a de T_Q_Concept **faire**
  si $a \in$ T_Q_Concept **alors**
    /* RECHERCHE_DES_CONJONCTIONS_DE_1_VUE */
    pour chaque tuple v de T_Vues_Concept **faire**
      si ($a(mot) = v(mot)$) and ($a(concept) = v(concept)$) **alors**
        Insert into T_Bucket ($a(K\_atome)$,$v[vue]$)
      fin si
    fin pour
  fin si
fin pour
/*DEUXIEME PARTIE : RECHERCHE DES CONJONCTIONS DE K VUES SUBSUMEES PAR $\forall a[mot].(a[valeurs])$*/
pour chaque tuple a de T_Q_Valeurs **faire**
  /*RECHERCHE DES VUES CANDIDATES A LA REECRITURE*/
  $compt := 0$
  pour chaque tuples v de T_Vues_Valeurs **faire**
    si $a(mot) = v(mot)$ **alors**
      $compt := compt + 1$
      Insert into $Cand$ ($v(mot$,$v(valeurs)$,$v(vues)$,$compt$)
    fin si
  fin pour
  /*CALCUL DES CONJONCTIONS DE VUES SUBSUMEES PAR $\forall a[mot].(a[valeurs])$*/
  B :=*ComputeEConj_1* ($Cand$,$a(valeurs)$)
  pour chaque tuples b de B **faire**
    Insert into T_Bucket ($a(K\_atome)$,$b(motif)$)
  fin pour
fin pour

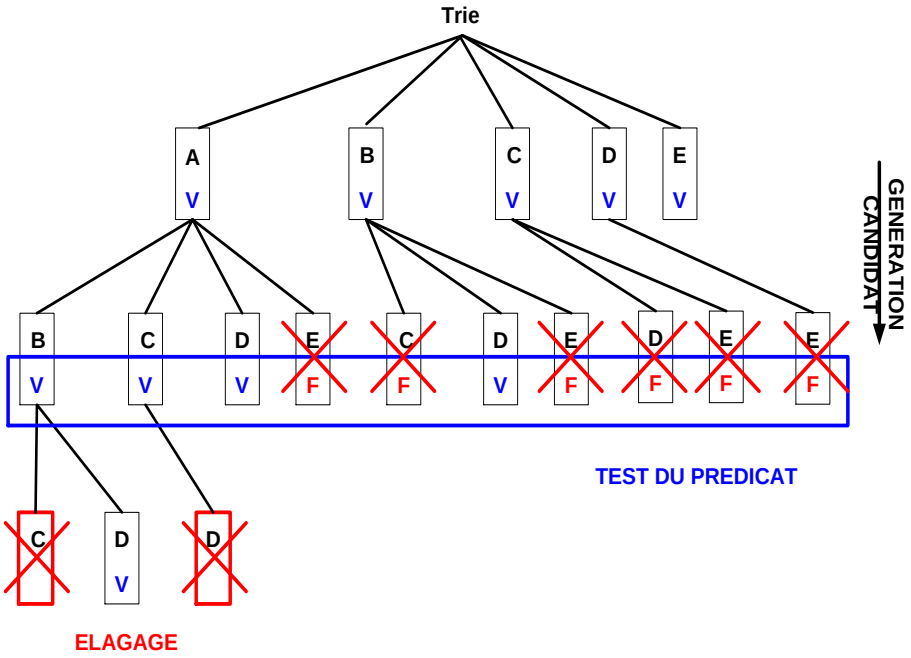

FIG. B.1 – Illustration de l'algorithme APRIORI

Générons à présent les motifs fréquents de niveau 3. Les candidats de niveau 3 sont $\{A, B, C\}$, $\{A, B, D\}$ et $\{A, C, D\}$. La phase d'élagage a permis d'identifier que les motifs $\{B, C\}$ et $\{C, D\}$ inclus dans les motifs $\{A, B, C\}$ et $\{A, C, D\}$ ne sont pas fréquents. Les candidats $\{A, B, C\}$ et $\{A, C, D\}$ sont donc éliminés. Il reste alors à effectuer le test du prédicat sur le candidat $\{A, B, D\}$. Au niveau 3, seul le motif $\{A, B, D\}$ est fréquent.

L'algorithme prend fin à ce niveau car il n'est plus possible de générer d'autres candidats. Les motifs fréquents de la figure B.1, i.e. les éléments de la théorie, sont donc $\{A\}$, $\{B\}$, $\{C\}$, $\{E\}\}$, $\{A, B\}$, $\{A, C\}$, $\{A, D\}$, $\{B, D\}$ et $\{A, B, D\}$.

B.3.1 Implémentation de *ComputeEConj_1*

Soit $Cand$ une relation qui liste les items candidats au niveau 1. Autrement dit $Cand$ correspond à l'ensemble F_w [20], décrit en section 5.1.

L'algorithme *ComputeEConj_1* prend en entrée les ensembles de valeurs $F_w = \{E_1, ..., E_n\}$ issus de $Cand$ et un ensemble de valeurs E tel que $E = a(valeurs)$ où a est le tuple (l'atome) en cours de traitement.

Comme APriori, *ComputeEConj_1* se décompose en trois étapes : la génération, l'élagage et le test du prédicat. Les phases de génération des candidats et d'élagage sont similaires à celles d'APriori. En revanche, le test du prédicat "être fréquent" est remplacé par un test de "non-inclusion", qui traduit le test du prédicat $P1$, introduit en section 5.3. Le test de "non-inclusion" est illustré ci-dessous.

[20] F_w est l'ensemble associé à $\mathcal{V}_w$ qui liste les vues ayant une contrainte de valeur sur le mot w

Supposons que *ComputeEConj_1* est appelé avec $a(valeurs) = E$.

Soit le motif $\{1\}$ qui identifie l'ensemble E_1. Le test est le suivant : $E_1 \not\subseteq E$?

Soit le motif $\{1, 2, 3\}$ qui identifie les ensembles $\{E_1, E_2, E_3\}$. Le test est le suivant : $(E_1 \cap E_2 \cap E_3) \not\subseteq E$?

Les ensembles de E_i qui vérifient ce test seront appelés "les motifs intéressants". On souhaite calculer les plus petits ensembles de F_w qui ne vérifient pas ce test. Ces ensembles appartiennent à la bordure négative de la théorie $\mathcal{B}d^-(\mathcal{T}h(r, \mathcal{L}, P_1))$. Aussi le résultat visé en sortie de *ComputeEConj_1* est la bordure négative de $\mathcal{T}h(r, \mathcal{L}, P_1)$.

Structures de données

L'implémentation de *ComputeEConj_1* est totalement couplée à un SGBD. Pour tirer profit de l'efficacité de la structure de stockage des TRIE, nous implémentons un TRIE en relationnel à l'aide de la relation $Trie$ (item integer, pere rowid, niveau integer, valueR nestedtable).

La relation $Trie$ stocke les motifs vérifiant le test de "Non-Inclusion". Le premier attribut de F permet de stocker les noeuds, i.e. les items, du TRIE. Le deuxième attribut de F permet d'identifier son père direct. Le troisième attribut de F donne le niveau dans l'arbre. L'attribut $valueR$ stocke la valeur de l'intersection des ensembles composant le motif.

Exemple 49

Soient trois ensembles $\{E_1, E_2, E_3\}$ tels que $E_1 = \{10, 12, 14\}$, $E_2 = \{10, 12, 20\}$, et $E_3 = \{12, 14, 15\}$, identifiés respectivement par les items $\{1, 2, 3\}$.

La relation $Trie$ pourrait être peuplée comme illustré par la figure B.4 où ti désigne l'identifiant unique (rowid) du tuple i.

	Item	Pere	Niveau	valueR
t1	1	null	1	$\{10, 12, 14\}$
t2	2	null	1	$\{10, 12, 20\}$
t3	3	null	1	$\{12, 14, 15\}$
t4	2	t1	2	$\{10, 12\}$
t5	3	t1	2	$\{12, 14\}$
t6	3	t2	2	$\{12\}$
t7	3	t4	3	$\{12\}$

TAB. B.4 – Un exemple de Trie en relationnel

○

ComputeEConj_1 utilise trois relations :

- F : ensemble des motifs fréquents,
  create table F (item integer, pere rowid, niveau integer, valueR nestedtable)
- B : ensemble des motifs de la bordure négative,
  create table B (motif NestedTable_integer)
- C : ensemble des motifs candidats d'un niveau donné
  create table C (item integer, pere rowid, niveau integer, valueR nestedtable)

La structure Trie est utilisée par les deux relations F et C.

ComputeEConj_1 décrit par l'algorithme 4, suit le même processus que l'algorithme APriori. En effet dans un premier temps, *ComputeEConj_1* parcourt la table $Cand$ pour identifier les motifs de niveau 1 qui vont peupler F (lignes $2-9$). Dans un deuxième temps (lignes $11-16$), tant qu'il est possible de générer des motifs, i.e. tant que les items de niveau k ont des "frères", *ComputeEConj_1* appelle les algorithmes 5 et 6 pour successivement générer les candidats de niveau supérieur à 1 et tester s'ils vérifient le prédicat de non inclusion.

Algorithme 4 *ComputeEConj_1* ($Cand$,E)

Entrée(s) : $Cand$, E
Sortie(s) : B

1: /* Génération des candidats de niveau 1 */
2: $k := 1$
3: **pour chaque** tuple $c \in Cand$ **faire**
4: **si** $c(valeurs)$ minus $E = \emptyset$ **alors**
5: insert into B values ($c(compteur)$)
6: **sinon**
7: insert into F values ($c(compteur)$,null,k,$c(valeurs)$)
8: **fin si**
9: **fin pour**
10: $card$:= (select count(*) from F where $niveau = 1$)
11: **tant que** $card > 1$ **faire**
12: $k := k + 1$
13: /*Génération des candidats de niveau (k) : algorithme 5*/
14: Gener_cand(k)
15: /*Test des candidats : algorithme 6*/
16: Test_Pred(E)
17: $card$:= (Select max(count(*) from F where $niveau = k$ group by niveau,pere)
18: **fin tant que**

L'algorithme 5 permet la génération des candidats et l'élagage des candidats, comme décrits en sous-section 6.2.1. La phase de génération des candidats (lignes $1-8$) est réalisée grâce à une requête SQL. Cette requête génère les motifs de niveau k, à partir des motifs de niveau $k-1$ (lignes $5-6$) ayant le même père (ligne 7). Elle calcule également la valeur de l'intersection des ensembles de valeurs qui permettra le test du prédicat.

La phase d'élagage (lignes $10-17$) vérifie pour chaque motif généré au niveau k s'il a un sous-ensemble qui est dans la bordure négative (ligne 14). Dans ce cas, le motif ne peut pas vérifier le prédicat et il est supprimé de $Cand$ (ligne 15).

L'algorithme 6 vérifie pour chaque candidat généré au niveau k si l'intersection des ensembles de valeurs est incluse dans E (lignes $1-2$). Si le test est vrai alors le prédicat n'est pas vérifié et le motif est inséré dans la bordure négative (lignes $3-5$). Dans le cas contraire, le motif est inséré dans le trie F (ligne 7).

Algorithme 5 Gener_cand(k)

Entrée(s) : C, B, $Cand$, k
Sortie(s) : C

```
insert into C (
select F2.item, F1.rowid,k, F1.valueR ∩ Cand.val
from F F1, F F2, Cand
where F2.item > F1.item
and F1.niveau=k − 1
and F2.niveau=k − 1
and NVL(F1.pere,chartorowid('0'))= NVL(F2.pere,chartorowid('0')))
and F2.item=Cand.compt
/* Suppression des candidats dont un sous-ensemble n'est pas fréquent*/
pour chaque tuple c ∈ C faire
    Iset :=select item From F start with F.rowid=c[pere] connect by F.rowid=PRIOR F.pere ;
    Iset :=Iset union c(item)
    pour chaque tuple b ∈ B faire
        si b(motif) ⊆ Iset alors
            delete c from C
        fin si
    fin pour
fin pour
```

Algorithme 6 Test_Pred(E)

Entrée(s) : C, E
Sortie(s) : B, F

```
pour chaque tuple c ∈ C faire
    si c(valueR) ⊆ E alors
        Iset :=select item from F start with F.rowid=c(pere) connect by F.rowid=PRIOR
        F.pere
        Iset :=Iset union c(item)
        insert into B (Iset)
    sinon
        insert into F (c)
    fin si
fin pour
```

TITRE : Réécriture de requêtes en termes de vues en présence de contraintes de valeurs pour un système d'intégration de sources de données agricoles

RESUME

Cette thèse traite le problème de la réécriture de requêtes en termes de vues en présence de contraintes de valeurs. Les contraintes de valeurs permettent de restreindre le domaine de valeurs d'un attribut donné. Elles connaissent un regain d'intérêt depuis leur utilisation dans le cadre du langage d'ontologie du web OWL. Ce travail est motivé par une application réelle qui vise à permettre l'intégration flexible d'un grand nombre de sources de données agricoles. L'étude de la décidabilité de ce problème est basée sur le cadre des logiques de description. Ce travail montre également que le problème de calculer les réécritures engendrées par les contraintes de valeurs se rattache au cadre de découverte des connaissances dans les bases de données introduit par Mannila and Toivonen. Ceci nous permet d'adapter des algorithmes existants et efficaces pour calculer ces réécritures et proposer un prototype de réécriture qui passe à l'échelle.

MOTS-CLES : Systèmes d'intégration de données, Réécriture de requêtes en termes de vues, Contraintes de valeurs, Logiques de description, Fouille de données.

TITLE : Query rewriting using views in presence of value constraints for an integration system of agricultural data sources

ABSTRACT

This thesis addresses the problem of query rewriting using views in the presence of value constraints. Value constraints allow for restricting value domain of a given attribute. They have been receiving a lot of attention because of their usefulness in the ontology web language OWL and in practical applications This work is motivated by a real application that aims at providing a scalable and flexible integration system for agricultural data sources. The investigation of this problem is based on the description logics framework that allows for studying the problem decidability. This work provides also a mapping between the problem of computing rewritings due to the presence of value constraints and the framework of Knowledge Discovery in Database (KDD) introduced by Mannila and Toivonen. That enables us to reuse existing and efficient algorithms to effectively compute rewritings and propose a scalable prototype of rewriting.

KEYWORDS : Data integration system, Query rewriting using views, Value constraints, Description logics, Knowledge discovery in database