Traitement de requêtes conjonctives avec négation : algorithmes et expérimentations

Khalil Ben Mohamed

ACADÉMIE DE MONTPELLIER
UNIVERSITÉ **M**ONTPELLIER **II**
Sciences et **T**echniques du **L**anguedoc

THÈSE

présentée au Laboratoire d'Informatique de Robotique
et de Microélectronique de Montpellier pour
obtenir le diplôme de doctorat

Spécialité	:	**Informatique**
Formation Doctorale	:	**Informatique**
École Doctorale	:	**Information, Structures, Systèmes**

Traitement de requêtes conjonctives avec négation
Algorithmes et expérimentations

par

Khalil BEN MOHAMED

Soutenue le 8 décembre 2010, devant le jury composé de :

Directrice de thèse
Marie-Laure MUGNIER, professeur ..... LIRMM, Université Montpellier II

Co-Directeur de thèse
Michel LECLÈRE, maître de conférences ..... LIRMM, Université Montpellier II

Rapporteurs
Marie-Christine ROUSSET, professeur ..... LIG, Université de Grenoble
Lakhdar SAIS, professeur ..... CRIL, Université d'Artois

Président du jury
Lhouari NOURINE, professeur ..... LIMOS, Université Blaise Pascal

Examinateur
Rémi COLETTA, maître de conférences ..... LIRMM, Université Montpellier II

À mes parents...
À la reine de mon cœur...
À ma petite princesse...

Remerciements

Par qui commencer... comment ne pas oublier quelqu'un... voilà les nouveaux problèmes (au moins NP-difficiles... si si je vous assure !) sur lesquels je me penche au moment d'écrire ces quelques lignes.

Je vais donc me lancer et commencer par remercier mes directeurs de thèse, Marie-Laure et Michel, qui m'ont tout simplement « bichonné » (pour reprendre les termes de qui se reconnaîtra peut-être) durant ces trois années qui furent parfois joyeuses, parfois déprimantes... J'ai eu la chance d'être encadré par une grande femme et un grand homme au sens noble du terme. En tant qu'être humain, j'ai été sensiblement impressionné par leur grandeur d'âme, leur simplicité et leur joie de vivre communicative. En tant que doctorant, j'ai énormément apprécié leurs conseils avisés et la clarté de leurs explications, leur patience durant mes coups de mou... Merci pour tout... MERCI BEAUCOUP !

Je remercie les rapporteurs de ma thèse, Marie-Christine ROUSSET et Lakhdar SAIS. Merci d'avoir accepté de rapporter ma thèse et merci pour vos retours pertinents. Je tiens également à remercier Lhouari NOURINE et Rémi COLETTA pour avoir accepté de faire partie de mon jury de thèse en tant qu'examinateurs.

Je tiens à remercier tous les membres de la « famille » RCR-GraphIK que j'ai eu l'occasion de rencontrer, dont Michel C., Jean-François, Alain, Eric, Rallou, Nicolas M., Olivier, Madalina, Jérôme, Jean-Rémi, Nicolas D., Michaël, Bruno et Cécile. Merci pour les bons moments passés lors des réunions d'équipe, sans oublier les bonnes pizzas que j'ai pu découvrir avec vous... l'équipe GraphIK, c'est aussi ça !

Je remercie toutes les personnes du LIRMM, laboratoire où j'ai effectué ma thèse, du

technicien de surface au directeur, en passant par les secrétaires (une mention spéciale pour Caroline YCRE, qui m'a aidé encore et encore pour les missions que j'ai eu à effectuer), les enseignants, les chercheurs... merci notamment à Marianne HUCHARD d'avoir accepté d'être ma tutrice de monitorat et d'avoir fait partie de mon comité de thèse, ainsi qu'à Stéphane GEORGES pour la maintenance continuelle du serveur RCR sur lequel j'ai travaillé. Je remercie les doctorants et autres jeunes gens bien sympas que j'ai eu l'occasion de côtoyer de près ou de loin durant ces trois années (autour d'un bon vieux Carcassonne ou au cours d'une partie à la vie à la mort de paintball par exemple), dont parmi eux et sans être exhaustif Gilles (mention spéciale à l'artiste !), Philippe, Nicolas P. (mention spéciale tennis !), Zeina, Fadi, Yuan, Julien, Xavier, Nadia, Raluca, Flop (mention spéciale pour l'aide apportée lors de la rédaction de thèse ! MERCI), Ben, Lisa, Jean-Rémi, Michaël (surnommé « scooter »- même sous la torture je ne dirai pas qui m'a révélé ça) et Nicolas (mention spéciale pétanque ! Et oui c'est aussi ça un chercheur...), Bruno, Cécile and so on.

Je désire particulièrement remercier mon partenaire (légendaire) de pétanque Jonathan LACOMBE pour les tournois de dingue que nous avons effectués ensemble, les parties de nuit, sous la pluie ou la neige avec les mains gelées, les parties de PES au bout de la nuit... Si tu me lis, je souhaite te dire que je suis très content et très fier de t'avoir côtoyé durant de si longues années (8 ans déjà !) et d'avoir pu compter à mes côtés un Homme (avec un grand « H ») qui mérite un réel respect.

Je remercie ma Famille avec un grand « F », en commençant par mes exceptionnels parents Anne-Marie et Abdelkader que j'aime énormément et sans qui je n'en serais pas là aujourd'hui, ma chère et tendre épouse Naoual qui m'a supporté et surtout toujours encouragé et aimé malgré les nuits blanches et le stress de la rédaction et de la soutenance, ma petite princesse bien-aimée Basma (qui signifie sourire en français) que j'aime « trop trop trop et à fond la caisse » comme elle le dit si bien du haut de ses 2 ans et demi, tous mes frères et sœurs sans exception (Myriam, Abdel-Illah, Khadija, Ilias, Anas, Aïcha et Tayib), mes grands-parents qui voulaient tant que je réussisse et qui nous ont malheureusement quitté il y a peu, mes neveux et nièces (Daoud, Bayane, Moussa, Nahila, Fayssal, Selmèn et le ou la prochaine à venir du côté de mon cher frérot), mes oncles et tantes, mes cousins et cousines, ma belle-famille dans son intégralité (mention spéciale à mes beaux-parents Allal et Fadma, aux deux Mohamed, à Mourad, Mustafa, Concha, Hayet, Abdessamad - courage pour la thèse !) and so on.

Pour finir, si j'ai donné une réponse à la question « par qui commencer ? » dès le départ, je crains de n'avoir qu'une solution approchée à la seconde question « comment ne pas oublier quelqu'un ? ». Ainsi je m'excuse devant toutes les personnes qui auraient dû figurer dans ces remerciements et que j'ai oubliées, et les remercie par avance de me pardonner. Merci à toutes et tous, et bonne continuation !

Table des matières

Remerciements iii

Table des matières v

1 Introduction 1

2 Notions de base 5
2.1 Notions classiques en bases de données 5
2.1.1 Schéma et instance de bases de données 6
2.1.2 Les requêtes 7
2.1.3 Problèmes de base 9
2.1.4 Introduction de la négation et hypothèse du monde clos 11
2.2 Notions classiques en bases de connaissances 14
2.2.1 La base de connaissances 14
2.2.2 Les logiques de description et les graphes conceptuels 17
2.2.3 Les correspondances et différences entre les BD et les BC 18
2.3 Reformulation en logique classique 19
2.3.1 Notions de base 19
2.3.2 Vision graphes 22
2.4 État de l'art algorithmique sur le problème *Déduction* 24
2.4.1 Impact de l'ajout de la négation sur le problème *Déduction* 24
2.4.2 L'approche de Ullman 26
2.4.3 L'approche de Wei et Lausen 27
2.4.4 L'approche de Leclère et Mugnier 29

3 Méthodologie expérimentale **35**
3.1 Les jeux de données . . . 36
3.2 Paramètres de génération . . . 36
3.3 Algorithme de génération aléatoire . . . 38
3.4 Instances difficiles et phénomène de seuil . . . 39
3.5 Méthodologie de test . . . 41
3.6 Influence des différents paramètres sur la difficulté . . . 41
3.6.1 Influence des densités des graphes source et cible . . . 43
3.6.2 Influence du pourcentage de négation . . . 46
3.6.3 Influence du pourcentage de sommets constantes . . . 47
3.6.4 Influence du nombre de sommets termes du graphe cible . . . 51
3.6.5 Influence de l'arité des relations . . . 52
3.6.6 Influence du nombre de relations . . . 55

4 Affinements et nouveaux algorithmes **59**
4.1 Raffinements de l'algorithme `recCheck` . . . 59
4.1.1 Choix du prochain littéral de complétion . . . 60
4.1.2 Choix du fils à parcourir en priorité . . . 63
4.1.3 Filtrage dynamique . . . 65
4.1.4 L'algorithme `recCheckPlus` . . . 68
4.2 Analyse et reformulation de l'algorithme de Wei et Lausen . . . 69
4.2.1 Propriétés fondamentales . . . 70
4.2.2 L'algorithme de Wei et Lausen . . . 73
4.2.3 Nouveaux algorithmes basés sur la propriété de Wei et Lausen . . . 78
4.3 Algorithme UNSATCheck . . . 84
4.3.1 La méthode . . . 86
4.3.2 Taille de la formule propositionnelle obtenue . . . 91
4.4 Comparaison des algorithmes . . . 92

5 Extensions **97**
5.1 Prise en compte des constantes . . . 97
5.2 Réponse à une requête non-booléenne . . . 105
5.3 Prise en compte d'une ontologie légère . . . 109

6 Conclusion **113**

Table des figures **123**

Liste des tableaux **126**

CHAPITRE

1 Introduction

Dans cette thèse, nous nous intéressons à des problèmes à la croisée de deux domaines, les bases de données et les bases de connaissances. Bien que les objectifs premiers de ces deux domaines soient différents (stockage et récupération des données stockées pour l'un et inférence de nouvelles connaissances à partir de celles connues pour l'autre), les problèmes liés à l'interrogation de la base sont centraux pour chacun d'eux.

Nous considérons ainsi deux problèmes fondamentaux : l'*inclusion de requêtes* et l'*évaluation d'une requête booléenne* (c'est-à-dire à réponse oui/non). Le problème de l'inclusion de requêtes prend en entrée deux requêtes q_1 et q_2, et cherche à déterminer si l'ensemble des réponses à q_1 est inclus dans l'ensemble des réponses à q_2 pour toute base de données/connaissances B. Il est fondamental en bases de données, où il est au cœur de l'optimisation des mécanismes d'évaluation de requêtes. Le problème de l'évaluation d'une requête booléenne prend en entrée une requête booléenne q et une base B et cherche à déterminer si B répond positivement à q.

Nous étudions plus précisément l'impact de l'introduction, dans les requêtes et la base de données/connaissances, d'une forme de négation basique mais néanmoins fondamentale, *la négation atomique*. Celle-ci nous permet d'exprimer des connaissances de la forme « cette entité n'a pas cette propriété » ou « cette relation n'apparaît pas entre ces entités ». Lorsque la négation intervient, on doit distinguer deux hypothèses fondamentales qui sont respectivement appelées *hypothèse du monde clos* et *hypothèse du monde ouvert*. Faire l'hypothèse du monde clos correspond à supposer que l'on a *une connaissance complète du monde*. Il en résulte que l'absence d'une information factuelle est automatiquement traduite par sa négation. L'hypothèse du monde ouvert suppose quant à elle *une connaissance incomplète du monde*. De ce fait, les informations manquantes sont simplement inconnues. La première hypothèse est classiquement faite en bases de données et la seconde en bases de connaissances. Toutefois, le mouvement actuel tend au rappro-

chement des deux approches, notamment sous la poussée du web sémantique : les bases de connaissances s'intéressent aux inférences sur de grandes bases de faits, en particulier sous l'hypothèse du monde clos, et les bases de données à l'interrogation en monde ouvert.

Nous considérons des requêtes *conjonctives*, qui forment une classe de requêtes couramment utilisées et sont considérées comme les requêtes de base en bases de données, auxquelles nous ajoutons la négation atomique. Intéressons-nous au problème de l'évaluation d'une telle requête, sous sa forme booléenne. Sous l'hypothèse du monde clos, seuls les faits positifs sont stockés dans la base B. Le problème est alors *NP-complet* et il est classiquement étudié en bases de données. Sous l'hypothèse du monde ouvert, B contient des faits positifs et négatifs. Le problème devient Π_2^p-complet. Il est alors équivalent au problème *d'inclusion de requêtes* (notons que l'hypothèse choisie n'a pas d'influence sur ce problème). Ces deux problèmes peuvent être reformulés dans un cadre plus abstrait, celui de la logique du premier ordre, et plus précisément dans le fragment existentiel conjonctif muni de la négation atomique (sans fonction). Les deux problèmes correspondent alors au problème de la *déduction* dans ce fragment, qui est le problème étudié au fil de cette thèse. On a également une équivalence immédiate avec le problème d'implication de clauses (sans fonction) qui est à la base des techniques d'ILP (Inductive Logic Programming), domaine à la croisée de l'apprentissage automatique et de la programmation logique. Les résultats de cette thèse s'appliquent donc à toute cette famille de problèmes.

Nous nous plaçons dans un cadre logique, mais également graphique (au sens de la théorie des graphes). En effet, nous représentons les requêtes et les faits sous forme de graphes. Ces graphes peuvent être vus comme des graphes conceptuels *simplifiés*, appelés *graphes polarisés*. Cette représentation graphique permet une meilleure lisibilité (pour de petits graphes) mais surtout de s'intéresser à la structure des objets et de manipuler des notions de graphe qui n'ont pas d'équivalent simple ou naturel en logique.

Le travail présenté dans cette thèse a été d'étudier expérimentalement des algorithmes de résolution du problème de déduction. D'un point de vue algorithmique, nous nous intéressons à des schémas d'algorithme déjà proposés que nous analysons et affinons, et nous présentons également de nouveaux algorithmes. D'un point de vue expérimental, n'ayant trouvé aucun jeu de tests adapté à notre étude, nous proposons un algorithme de génération d'instances aléatoires et mettons en place une méthodologie expérimentale. Nous définissons les paramètres de génération d'une instance aléatoire et analysons l'influence des différents paramètres sur la difficulté de résolution. En nous servant des valeurs de paramètres permettant de construire des instances « difficiles », nous comparons expérimentalement les algorithmes entre eux. Nous proposons également trois extensions du cadre étudié : la prise en compte de constantes dans les requêtes, la recherche des réponses à une requête non-booléenne et la prise en compte d'une « ontologie légère ». Nous montrons que les algorithmes présentés dans cette thèse s'étendent sans difficulté à ces extensions.

Organisation du mémoire

Le chapitre 2 présente le contexte de notre étude. Nous introduisons des notions classiques en bases de données et en bases de connaissances, notamment les requêtes conjonctives, les problèmes étudiés, les hypothèses du monde clos et du monde ouvert et les ontologies, puis nous mettons en avant les similarités et les différences intrinsèques aux deux approches « bases de données » et « bases de connaissances ». Nous présentons ensuite une reformulation des problèmes étudiés dans un cadre plus abstrait, celui du fragment existentiel conjonctif de la logique du premier ordre (par exemple des formules logiques sont associées aux requêtes...), et nous précisons nos choix, à savoir l'hypothèse du monde ouvert, une ontologie « triviale » (c'est-à-dire un ensemble de prédicats et de constantes en logique) et le problème étudié en terme de déduction. Nous donnons également une vision des objets étudiés sous forme de graphes conceptuels simplifiés. Finalement, nous présentons trois travaux antérieurs concernant le problème de déduction ou un problème directement équivalent [Ullman, 1997][Wei et Lausen, 2003][Leclère et Mugnier, 2007], les deux derniers améliorant le premier tout en adoptant des approches différentes.

Le chapitre 3 est consacré à la présentation de la méthodologie expérimentale utilisée dans le reste de cette thèse. Nous présentons les paramètres de génération d'une instance aléatoire du problème de déduction étudié et proposons un algorithme de génération de telles instances. Nous discutons ensuite des instances dites « difficiles », ainsi que du phénomène de seuil (aussi connu sous le nom de *transition de phase*) associé, et mettons en évidence une transition de phase de la phase non-déduction à la phase déduction. Finalement, nous analysons l'influence de chaque paramètre sur la difficulté de résolution d'une instance et localisons des pics de difficulté « raisonnables », c'est-à-dire des valeurs de paramètres pour lesquelles les instances générées sont difficiles tout en restant généralement solubles en un temps « raisonnable » (ne dépassant pas une valeur prédéfinie). Nous proposons alors un protocole d'expérimentation qui nous sert dans les chapitres suivants afin de tester nos propositions de raffinements et de comparer différents algorithmes.

Dans le chapitre 4, nous proposons trois raffinements de l'algorithme `recCheck` de Leclère et Mugnier [2007] et nous les analysons expérimentalement. Nous montrons que ces raffinements optimisent énormément l'algorithme `recCheck`, à la fois en temps de calcul et en taille de l'espace de recherche exploré. Nous présentons alors un nouvel algorithme raffiné appelé `recCheckPlus`. Puis nous analysons l'algorithme de Wei et Lausen [2003] ainsi que les propriétés sur lesquelles il repose, et nous mettons en évidence la présence d'une erreur dans l'algorithme. En nous basant sur la propriété fondamentale de Wei et Lausen [2003], nous proposons alors deux algorithmes, respectivement `breadthCheck` et `recANDcheck`, qui corrigent l'erreur et parcourent l'espace de recherche avec deux stratégies différentes. Enfin, nous proposons l'algorithme `UNSATCheck`, basé sur une approche totalement différente des précédentes, qui conduit à tester l'insatisfiabilité d'une forme clausale propositionnelle, ce qui permet d'utiliser un solveur SAT. Ensuite nous compa-

rons expérimentalement les quatre algorithmes. Nous montrons que l'algorithme `breadthCheck` donne de très mauvais résultats par rapport aux deux autres et que l'algorithme `recCheckPlus` donne généralement les meilleurs résultats. Une comparaison avec un solveur logique a été entreprise, mais celle-ci étant encore préliminaire, nous la présentons dans nos perspectives (voir chapitre 6).

Le chapitre 5 propose trois extensions au cadre de travail étudié dans cette thèse, et nous montrons que les algorithmes présentés peuvent s'adapter sans difficulté à chacune d'elles. La première consiste à prendre en compte les constantes dans les requêtes. Nous montrons que les constantes peuvent faciliter la résolution du problème de deux manières : nous pouvons utiliser des parties de la requête contenant des constantes comme un filtre de prétraitement ou les intégrer dans les mécanismes de résolution propres à chaque algorithme. La seconde extension consiste à s'intéresser à des requêtes non-booléennes, c'est-à-dire à rechercher des tuples de valeurs constituant les réponses à une requête et plus seulement une réponse oui/non. La dernière repose sur la prise en compte d'une ontologie « légère », c'est-à-dire spécifiant une relation de spécialisation sur un ensemble de concepts et un ensemble de relations sous la forme d'un ordre partiel ou d'un préordre (relation réflexive et transitive).

Le chapitre 6 conclut ce mémoire et dessine quelques perspectives à ce travail.

Les travaux concernant les affinements conduisant à l'algorithme `recCheckPlus` et leurs expérimentations ont été publiés à la conférence francophone RFIA [Ben Mohamed *et al.*, 2010b] [1] et aux conférences internationales AIMSA [Ben Mohamed *et al.*, 2010c] et DEXA [Ben Mohamed *et al.*, 2010a]. Les travaux concernant l'algorithme `UNSATCheck` et la comparaison d'algorithmes ont été publiés à la conférence française JIAF [Ben Mohamed *et al.*, 2008] et à DEXA [Ben Mohamed *et al.*, 2010a]. Les travaux sur les algorithmes `breadthCheck` et `recANDcheck`, ainsi que les extensions n'ont pas encore été publiés.

1. Ce papier a reçu le prix IA de l'AFIA.

CHAPITRE

2 Notions de base

Préambule

Dans ce chapitre, nous rappelons des notions classiquement utilisées en bases de données et en bases de connaissances et nous présentons les similitudes et les différences fondamentales entre ces deux approches. Puis nous traduisons ces notions dans un langage plus abstrait, celui d'un fragment de la logique du premier ordre, et nous en présentons une vision sous forme de graphes. Finalement, nous nous intéressons au problème de la déduction apparaissant dans les différents formalismes sous différentes formes, et faisons un état de l'art des algorithmes permettant de le résoudre.

Sommaire

2.1 Notions classiques en bases de données 5
2.2 Notions classiques en bases de connaissances 14
2.3 Reformulation en logique classique 19
2.4 État de l'art algorithmique sur le problème *Déduction* 24

2.1 Notions classiques en bases de données

Dans cette partie, nous rappelons certaines définitions et certains problèmes fondamentaux en bases de données en nous appuyant sur [Abiteboul *et al.*, 1995].

2.1.1 Schéma et instance de bases de données

Les bases de données (BD) peuvent être vues comme des ensembles d'informations stockées dans un dispositif informatique. Elles permettent notamment à l'utilisateur d'organiser et de structurer ses informations, en les stockant efficacement en très grande quantité, tout en facilitant leur manipulation par un système évolué de requêtage.

Films

Titre	Réalisateur	Acteur
Avatar	Cameron	Worthington
Avatar	Cameron	Bana
Avatar	Cameron	Weaver
...	...	...
Troie	Petersen	Pitt
Troie	Petersen	Bana
Troie	Petersen	Kruger

Localisations

Cinéma	Ville
Gaumont	Montpellier
CGR	Lattes
Diagonal	Boissiere
...	...
Royal	Fabregues

Séances

Cinéma	Titre	Horaire
Gaumont	Avatar	19h30
Gaumont	Avatar	21h30
CGR	Avatar	19h
CGR	Troie	20h
Diagonal	Avatar	21h
...	...	...
Royal	Troie	21h

FIGURE 2.1 : la base de données **CINEMA**.

Un exemple simple de BD (inspiré de [Abiteboul *et al.*, 1995]) est illustré par les trois tables de la figure 2.1. Chaque table correspond à *une relation* et est nommée (par exemple *Films*). Chaque nom de colonne correspond à *un attribut* de la relation (par exemple *Titre*, *Réalisateur* et *Acteur* sont des attributs de la relation *Films*). Chaque ligne correspond à *un tuple* composé de valeurs littérales (aussi appelées *constantes*, par exemple des entiers naturels, des chaînes de caractères etc.). Enfin, nous distinguons le *schéma* de la base de données, qui définit la structure de la BD, de l'*instance* de la base de données, qui définit son contenu. On peut rapprocher cette distinction de celle opérée entre les types et les valeurs dans les langages de programmation : prenons par exemple les variables A et B et les suites d'instructions « A : entier, B : booléen » et « A := 5, B := vrai » ; la première correspond

à la définition de la structure des variables A et B alors que la seconde correspond à leur instantiation.

Plus formellement, nous considérons les définitions suivantes :

Définition 2.1 (Schéma de base de données) *Un* schéma de base de données *est une paire* $\mathcal{S} = (\mathcal{R}, \mathbf{dom})$ *où* $\mathcal{R}$ *est un ensemble fini non vide de relations et* **dom** *est un ensemble infini dénombrable de constantes. Chaque relation a une arité (supérieure à 0) définissant le nombre de ses attributs.*

Définition 2.2 (Instance de base de données) *Une* instance de base de données *(appelée plus simplement une base de données)* D *sur un schéma* $\mathcal{S} = (\mathcal{R}, \mathbf{dom})$ *projette chaque relation* R *d'arité* k *de* $\mathcal{R}$ *sur un sous-ensemble fini de* $\mathbf{dom}^k$*, noté* $D(R)$*, soit* $D = \{D(R) | R \in \mathcal{R}\}$

Sur l'exemple de la figure 2.1, le schéma de la base de données **CINEMA** est composé de l'ensemble de relations $\mathcal{R}$ = *{Films,Localisations,Séances}*, où *Localisations* est d'arité 2 et *Films* et *Séances* sont d'arité 3, et de l'ensemble infini de constantes **dom** contenant entre autre les constantes *Avatar*, *Pitt*, etc.

Pour finir, voici quelques notations utilisées par la suite (elles trouveront leur justification en section 2.3). Elles sont exprimées « à la Datalog [1] » (comme d'autres notations par la suite) : nous appelons **var** un ensemble infini de variables ; un *terme* t est une variable ou une constante, soit $t \in \mathbf{var} \cup \mathbf{dom}$; un *tuple de termes d'arité* k est une expression $\vec{t} = t_1, \ldots, t_k$, où chaque t_i est un terme ; un *atome* sur $\mathcal{S} = (\mathcal{R}, \mathbf{dom})$ est une expression $R(\vec{t})$, où $R \in \mathcal{R}$ et $\vec{t}$ est un tuple de termes de $\mathbf{var} \cup \mathbf{dom}$; Nous représentons la base de données comme un ensemble d'atomes composés d'un tuple de constantes : dans cette représentation, les noms des attributs disparaissent ; le $i^{ème}$ argument d'un atome $R(\vec{t})$ est la valeur du $i^{ème}$ attribut de la relation R. Nous avons alors :
CINEMA={{Films(Avatar,Cameron,Worthington),...},
{Localisations(Gaumont,Montpellier),...},{Séances(Gaumont,Avatar,19h30),...}}

2.1.2 Les requêtes

Après avoir stocké des informations, un utilisateur de BD doit pouvoir les retrouver facilement. Ainsi, les systèmes de bases de données fournissent des outils de requêtage évolués permettant d'aller rechercher une information particulière parmi toutes les informations présentes dans la BD. L'exemple 1 fournit des exemples de requêtes exprimées en langue naturelle :

Exemple 1.
(1) Qui est le réalisateur du film Avatar ?
(2) Dans quelle ville se trouve le cinéma Royal ?

1. Datalog est un langage de requêtes basé sur la programmation logique.

(3) Lister les cinémas qui diffusent un film du réalisateur Petersen.
(4) Y a-t-il une séance à 21h d'un film dans lequel joue l'acteur Pitt ?
(5) Lister les acteurs du film Troie du réalisateur Petersen qui ne sont pas acteurs dans le film Avatar du réalisateur Cameron.

Dans la suite, nous nous intéressons à une forme de requête limitée mais extrêmement utilisée, considérée comme les requêtes de base en BD ([Chandra et Merlin, 1977] par exemple) : *les requêtes conjonctives*.

Pour présenter l'intuition des requêtes conjonctives, nous en donnons quelques exemples (tirés de l'exemple 1) en considérant la BD **CINEMA** : les requêtes (1) et (2) concernent l'extraction d'informations à partir d'une seule relation (respectivement *Films* et *Localisations*), alors que les requêtes (3) et (4) impliquent deux relations (*Films* et *Séances*). Nous allons voir que ces requêtes peuvent être décrites en terme d'existence de tuples connectés entre eux par une égalité de valeurs d'attributs communs. De cette notion de connection découle celle de conjonction, d'où le nom de requêtes conjonctives.

Plus formellement, nous avons la définition suivante :

Définition 2.3 (Requête conjonctive) *Soit $\mathcal{S} = (\mathcal{R}, \mathbf{dom})$ un schéma de base de données. Une* requête conjonctive (RC) *sur $\mathcal{S}$ est une expression de la forme :*

$$q = ans(\overrightarrow{u}) \leftarrow R_1(\overrightarrow{u_1}), \ldots, R_n(\overrightarrow{u_n})$$

où $n \geq 1$, $R_1, \ldots, R_n$ sont des relations de $\mathcal{R}$, ans *(pour « answer ») est une relation spéciale n'apparaissant pas dans $\mathcal{R}$, $\overrightarrow{u}$ est un tuple de variables et $\overrightarrow{u_1}, \ldots, \overrightarrow{u_n}$ sont des tuples de termes ; $R_1(\overrightarrow{u_1}), \ldots, R_n(\overrightarrow{u_n})$ sont des atomes (aussi appelés sous-buts) sur $\mathcal{S}$; $ans(\overrightarrow{u})$ est appelée la* tête *de la requête et $R_1(\overrightarrow{u_1}), \ldots, R_n(\overrightarrow{u_n})$ son* corps *; Le symbole « , » correspond à la conjonction ; on note **var**(q) l'ensemble des variables de q et **var**(tête(q)) l'ensemble des variables de la tête de q. Toutes les variables apparaissant dans $\overrightarrow{u}$ doivent apparaître au moins une fois dans $\overrightarrow{u_1}, \ldots, \overrightarrow{u_n}$. Le symbole « _ » est utilisé pour remplacer les variables n'apparaissant qu'une seule fois dans la requête (on parle alors de variable anonyme). Sans perte de généralité, nous supposons que le même atome n'apparaît pas deux fois dans le corps de la requête. Finalement, une* RC *est dite* booléenne *(réponse oui/non) si aucune variable n'apparaît dans sa tête (notée* ans()*).*

Nous pouvons dès lors formaliser les quatre premières requêtes de l'exemple 1 :

Exemple 2.
(1) $q_1 = ans(x) \leftarrow Films(Avatar, x, \_)$
(2) $q_2 = ans(x) \leftarrow Localisations(Royal, x)$
(3) $q_3 = ans(x) \leftarrow Films(w, Petersen, \_), Seances(x, w, \_)$
(4) $q_4 = ans() \leftarrow Films(x, \_, Pitt), Seances(\_, x, 21h)$

Concernant la notion de conjonction, nous remarquons par exemple que la requête q_3 est obtenue par une conjonction de deux atomes, connectés entre eux par la variable « w » ; la requête (4) est quant à elle booléenne.

Intuitivement, une requête est vue comme un outil d'extraction d'information. Si l'on peut trouver des valeurs pour les variables de la requête telles que le corps est « satisfait », alors nous avons une réponse à la requête. Cette notion de « valeurs pour les variables de la requête » est capturée par la notion de *substitution*. Formellement, soit un ensemble V de variables et un ensemble T de termes, une substitution s de V dans T est une application de V dans T. Nous notons $s(E)$ l'application d'une substitution s à une entité E (tuple, requête...). Dans notre cas, nous allons considérer des substitutions de **var**(q) dans **dom**, et de plus nous imposons qu'elles satisfassent les informations contenues dans la BD considérée. Cette forme de substitution particulière définit la sémantique d'une *RC* et conduit à la notion de réponse à une *RC* :

Définition 2.4 (Réponse) *Soit* $q = ans(\overrightarrow{u}) \leftarrow R_1(\overrightarrow{u_1}),\ldots,R_n(\overrightarrow{u_n})$ *une* RC *et* D *une base de données sur un schéma* $\mathcal{S} = (\mathcal{R},$**dom**$)$*. On note q(D) =* $\{s(\overrightarrow{u}) | s$ *est une substitution des variables de q dans* **dom** *et* $s(\overrightarrow{u_i}) \in D(R_i)$ *pour* $i = 1\ldots n\}$ *l'ensemble des réponses à q dans D. Si q est une RC booléenne, la réponse est* oui *si* $q(D) = \{()\}$*, non sinon.*

Par exemple, les réponses aux quatre premières requêtes de l'exemple 1 sont :

Exemple 3.
(1) $q_1(CINEMA) = \{(Cameron)\}$, où chacune des trois substitutions $s_1 = \{x \mapsto Cameron, \_ \mapsto Worthington\}$, $s_2 = \{x \mapsto Cameron, \_ \mapsto Bana\}$ et $s_3 = \{x \mapsto Cameron, \_ \mapsto Weaver\}$ fournit la même réponse.
(2) $q_2(CINEMA) = \{(Fabregues)\}$
(3) $q_3(CINEMA) = \{(CGR),(Royal)\}$
(4) $q_4(CINEMA) = \{()\}$

Cette notion de substitution est connue sous le nom d'homomorphisme de requêtes, lorsqu'elle concerne deux *RC* :

Définition 2.5 (Homomorphisme de *RC***)** *Soient* $q_1 = ans(\overrightarrow{u}) \leftarrow R_1(\overrightarrow{u_1}),\ldots,R_n(\overrightarrow{u_n})$ *et* $q_2 = ans(\overrightarrow{v}) \leftarrow S_1(\overrightarrow{v_1}),\ldots,S_m(\overrightarrow{v_m})$ *deux RC. Un* homomorphisme de q_2 dans q_1 *est une substitution s des variables de* q_2 *par les termes de* q_1 *telle que* $s(\overrightarrow{v}) = \overrightarrow{u}$ *et* $s(corps(q_2)) \subseteq corps(q_1)$.

2.1.3 Problèmes de base

Nous considérons trois problèmes fondamentaux en BD :

1. **Le test d'existence d'une réponse à une requête** : étant données une requête q et une base de données D, il s'agit de déterminer s'il existe une réponse à q dans D.
2. **Le calcul de toutes les réponses à une requête** : étant données une requête q et une base de données D, il s'agit de lister toutes les réponses à q dans D.
3. **Le test d'inclusion de requêtes** : étant données deux requêtes q_1 et q_2, il s'agit de déterminer si q_1 est incluse dans q_2 (noté $q_1 \sqsubseteq q_2$), c'est-à-dire si l'ensemble des réponses à q_1 est inclus dans l'ensemble des réponses à q_2 pour toute base de données.

Les deux premiers problèmes sont naturellement associés au mécanisme de requêtage ; le premier est fondamental pour l'étude de la complexité du requêtage alors que le second est le plus utilisé en pratique. Pour ce qui est du problème d'inclusion de requêtes, il est utilisé, principalement avec les requêtes conjonctives présentées en section 2.1.2, dans des algorithmes pour résoudre de nombreux autres problèmes dont *l'optimisation des mécanismes d'évaluation de requêtes* [Chandra et Merlin, 1977][Aho *et al.*, 1979], *la détection d'indépendance de requêtes de mises à jour de BD* [Levy et Sagiv, 1993] et *la vérification des contraintes d'intégrité* [Gupta *et al.*, 1994]. Au début du $21^{ème}$ siècle, il a connu un regain d'intérêt dû à son étroite relation avec les problèmes de *réponse à des requêtes avec vues* [Levy *et al.*, 1995][Abiteboul et Duschka, 1998][Halevy, 2001], qui sont des problèmes centraux en *intégration et stockage de données* (voir l'étude dans [Ullman, 1997] par exemple). Depuis, il a été montré central pour les problèmes d'*échange de données* [Fagin *et al.*, 2005] et de *web sémantique* [Ortiz *et al.*, 2006] pour ne citer que ces exemples.

Le théorème suivant fait le lien entre homomorphisme de *RC* et inclusion de requêtes conjonctives :

Théorème 2.6 *[Chandra et Merlin, 1977] Soient q_1 et q_2 deux RC. $q_1 \sqsubseteq q_2$ si et seulement s'il existe un homomorphisme de q_2 dans q_1.*

Le problème du test d'inclusion de requêtes conjonctives (noté *IRC*, voir le théorème 2.6) est *NP-complet*[2] [Chandra et Merlin, 1977]. Différents travaux se sont intéressés à la recherche de cas particuliers de *RC* pour lesquels le problème est en théorie facilement soluble (c'est-à-dire qu'il fait alors partie de la classe *P* ou d'une classe de complexité en-dessous). Plusieurs cas particuliers ont été mis en exergue (avec une vision graphe des *RC*) ; il a été montré qu'un algorithme pouvait résoudre le problème d'inclusion de requêtes conjonctives en un temps linéaire dans le cas où la requête contenante (q_2 pour notre définition du problème) est *acyclique* (cette notion correspond à une structure d'hypergraphe acyclique) [Yannakakis, 1981] et dans le cas où chaque relation n'apparaît pas plus de deux fois dans le corps de la requête contenue [Saraiya, 1991]. Plus récemment, Gottlob *et al.* [2002] ont proposé le concept de décomposition en hyper-arbre (qui est une généralisation de l'acyclicité d'hypergraphe) de la requête contenante et le concept associé

2. La classe *NP* correspond à la classe des problèmes Non-déterministes Polynomiaux (elle est au-dessus de la classe *P*, qui contient les problèmes admis facilement solubles). Elle réunit les problèmes de décision (qui admettent une réponse oui/non) pour lesquels on recherche la réponse oui et qui peuvent être résolus sur une machine de Turing non déterministe (l'état suivant du calcul n'est pas « déterminé » par l'état courant) en temps polynomial. De façon équivalente, c'est la classe des problèmes qui admettent un algorithme polynomial pour tester la validité d'une solution du problème (on dit aussi qu'il existe un certificat polynomial). Intuitivement, les problèmes dans NP sont les problèmes de décision qui peuvent être résolus en énumérant l'ensemble des solutions possibles et en les testant à l'aide d'un algorithme polynomial. La classe duale de la classe NP est appelée Co-NP. On dit qu'un problème appartenant à une classe de complexité C (P,NP...) est C-complet s'il est au moins aussi difficile que tous les problèmes dans C. (voir [Papadimitriou, 1994] pour plus de détails sur la théorie de la complexité).

de largeur d'hypergraphe (hypertreewidth) ; dans le cas où q_2 a une largeur d'hypergraphe bornée, le problème d'inclusion de requêtes conjonctives est soluble en un temps polynomial.

Néanmoins tous ces résultats concernent les requêtes conjonctives « positives ». Dans une application réelle, il est souvent nécessaire d'utiliser des requêtes plus expressives, afin de pouvoir exprimer la négation (par exemple la requête **(5)** de l'exemple 1) ou encore des comparaisons arithmétiques comme *Salaire* > *2000€*. Dans ce mémoire nous nous focalisons sur l'introduction de la négation (pour les comparaisons arithmétiques voir [K., 1988][Afrati *et al.*, 2004][Wei et Lausen, 2008] par exemple).

2.1.4 Introduction de la négation et hypothèse du monde clos

Nous nous intéressons à l'introduction d'une forme de négation basique mais néanmoins fondamentale, *la négation atomique*. Cette négation porte uniquement sur un atome : par exemple $\neg sur(A,B)$ (« l'entité A n'est pas sur l'entité B », le symbole $\neg$ exprimant la négation), mais pas sur une forme plus structurée comme $\neg(sur(A,B) \wedge sur(C,D))$ (il est faux que : « A est sur B et C est sur D »). Elle nous permet d'exprimer des connaissances de la forme « cette entité n'a pas cette propriété », « cette relation n'apparaît pas entre ces entités » ...

On distingue généralement deux grandes façons d'interpréter la négation. En effet, si nous posons une question de la forme « $\neg sur(A,B)$? » (« est-ce que l'entité A n'est pas sur l'entité B »), le symbole $\neg$ peut être interprété de différentes manières. Il peut signifier que :

1. La connaissance « $sur(A,B)$ » ne peut pas être prouvée.
2. La connaissance « $\neg sur(A,B)$ » peut être prouvée.

La première affirmation correspond à l'hypothèse du monde clos [Reiter, 1977], qui est communément faite dans les bases de données. Faire l'hypothèse du monde clos correspond à supposer *une connaissance complète du monde*. Ainsi, on se contente d'une description partielle du monde : seules les informations positives (vraies) sont spécifiées. C'est-à-dire que l'absence d'une information est automatiquement traduite par sa négation. Intuitivement, cela signifie que bien que la définition d'une propriété soit limitée à la liste des individus la satisfaisant, il est possible de dire pour tout individu s'il a ou non la propriété. La connaissance $\neg sur(A,B)$ est alors interprétée par « $sur(A,B)$ n'apparaît pas dans la BD », ou plus généralement « $sur(A,B)$ ne peut pas être déduit de la BD ». D'un point de vue bases de données, ce choix est motivé par la simplification de la représentation d'un côté et les performances de l'autre [Bidoit, 1991]. Par exemple, pour définir une propriété, il semble plus naturel et il est plus économe de donner la liste des individus satisfaisant cette propriété plutôt que d'énumérer tous les individus et d'indiquer pour chacun d'eux s'il satisfait ou non la propriété.

Prenons le cas de la propriété « être un cinéma de Montpellier (noté *estCinemaMTP*) ». Supposer une connaissance complète pour cette propriété semble raisonnable ; tous les

cinémas n'apparaissant pas dans la liste des cinémas de Montpellier, c'est-à-dire tous les V tels que *Localisations*(V,Montpellier) n'est pas un atome de la BD **CINEMA**, sont considérés comme ne se situant pas à Montpellier. Ainsi d'un côté la représentation des données est simplifiée puisque seuls les cinémas situés à Montpellier sont représentés, et de l'autre les mécanismes de réponse sont optimisés.

Il est néanmoins à noter que, si cette hypothèse semble naturelle en bases de données, il est impossible en logique classique (plus généralement pour toute logique monotone) d'inférer des informations négatives à partir d'informations positives. Il est par exemple impossible d'inférer $\neg$*estCinemaMTP(A)* si l'information *estCinemaMTP(A)* n'apparaît pas dans la base. Différentes logiques non-monotones ont été proposées pour traduire l'hypothèse du monde clos, comme la logique des défauts ; pour être capable d'inférer des informations négatives à partir d'une description incomplète, on a recours à des règles des défauts de la forme : « s'il est cohérent de supposer qu'une entité ne satisfait pas une propriété, alors inférer que cette entité ne satisfait pas la propriété » (voir [Apt et Bol, 1994] pour plus de détails concernant les logiques non-monotones).

Pour prendre en compte la négation, nous étendons différentes notions dont la précédente définition de requête conjonctive et sa sémantique :

Définition 2.7 (Requête conjonctive avec négation) *Soit $\mathcal{S} = (\mathcal{R}, \mathbf{dom})$ un schéma de base de données. Une requête conjonctive avec négation ($RC^\neg$) sur $\mathcal{S}$ est une expression de la forme :*

$$q = ans(\overrightarrow{u}) \leftarrow P_1(\overrightarrow{u_1}), \ldots, P_n(\overrightarrow{u_n}), \neg N_1(\overrightarrow{v_1}), \ldots, \neg N_m(\overrightarrow{v_m})$$

où $n+m \geq 1$, $P_1, \ldots, P_n, N_1, \ldots, N_m$ sont des relations de $\mathcal{R}$, $P_1(\overrightarrow{u_1}), \ldots, P_n(\overrightarrow{u_n})$ sont des atomes sur $\mathcal{S}$ (aussi appelés littéraux positifs*) et $\neg N_1(\overrightarrow{v_1}), \ldots, \neg N_m(\overrightarrow{v_m})$ sont des négations d'atomes sur $\mathcal{S}$ (aussi appelés* littéraux négatifs*).*

Nous pouvons dès lors formaliser la dernière requête de l'exemple 1 :

Exemple 4.
(5) $q_5 = ans(x) \leftarrow Films(Troie, Petersen, x), \neg Films(Avatar, Cameron, x)$

Concernant la sémantique d'une $RC^\neg$ (sous l'hypothèse du monde clos), nous avons :

Définition 2.8 (Réponse) *Soit q une $RC^\neg$ de la forme donnée dans la définition 2.7 et* D *une base de données sur un schéma $\mathcal{S} = (\mathcal{R}, \mathbf{dom})$. On note $q(D) = \{s(\overrightarrow{u}) | s$ est une substitution des variables de q dans les constantes de D (c'est-à-dire $\mathbf{dom}$), $s(\overrightarrow{u_i}) \in D(P_i)$ pour $i = 1 \ldots n$, et $s(\overrightarrow{v_j}) \notin D(N_j)$ pour $i = 1 \ldots m\}$ l'ensemble des réponses à q dans D.*

L'ajout de la négation amène une plus grande expressivité dans le langage de requêtes à l'utilisateur. Cependant, si les $RC^\neg$ ainsi obtenues sont utilisées sans restriction, il est possible d'exprimer des requêtes qui ont une infinité de « réponses ». Illustrons ce problème

à l'aide de la requête $q = ans(x) \leftarrow \neg Films(Avatar, Cameron, x)$. Nous considérons une base de données D quelconque sur le schéma $\mathcal{S} = (\mathcal{R}, \mathbf{dom})$ de la figure 2.1 ; d'après la définition de réponse vue plus haut, la requête q produit tous les tuples (a) tels que $a \in \mathbf{dom}$ et $Films(Avatar, Cameron, a) \notin D$. Puisque **dom** est un ensemble infini de constantes, la requête q produit un ensemble infini de réponses.

Pour pallier ce problème, une façon de faire consiste à restreindre le domaine de chaque attribut de la BD en le typant. Par exemple, on pourrait imposer que les attributs de la table *Films* soient respectivement de type *Titre*, *Réalisateur* et *Acteur*, chaque type ayant un domaine fini. L'ensemble des réponses à $q = ans(x) \leftarrow \neg Films(Avatar, Cameron, x)$ serait alors fini selon le choix fait sur le domaine du type *Acteur* (par exemple les acteurs apparaissant dans la BD). Néanmoins, cette approche n'est pas entièrement satisfaisante car les réponses à une requête dépendent désormais du domaine des types et peuvent différer d'une BD à l'autre. Prenons par exemple deux BD possédant exactement la même structure (relations et attributs) et contenant les mêmes informations, mais définissant des types d'attribut de domaines différents. Les réponses à la requête q seront différentes selon la BD considérée car ils dépendent du domaine défini pour la BD (on dit que la requête n'est pas indépendante du domaine). Même si les domaines sont finis, l'utilisateur ne connaît généralement pas le contenu exact du domaine de chaque attribut, et ainsi ne pourra pas comprendre la différence de réponses à la même requête pour des BD en apparence identiques. Ceci est un argument en faveur de l'utilisation de requêtes indépendantes du domaine. Plus précisément, on impose que les requêtes soient *safe* : soit q une $RC^{\neg}$ de la forme donnée dans la définition 2.7, elle est dite *safe* si chaque variable apparaissant dans un littéral négatif apparaît aussi dans un littéral positif du corps de q ; la requête $q = ans(x) \leftarrow \neg Films(Avatar, Cameron, x)$ *n'est pas safe* alors que la requête **(5)** est *safe*. Cette dernière ne pose aucun problème, puisque la variable x est « contrainte » par le littéral positif : les réponses font parties des acteurs du film Troie du réalisateur Petersen, soit {Pitt, Bana, Kruger}.

Exemple 5.
(5) $q_5(CINEMA) = \{Pitt, Kruger\}$

Avec la prise en compte de la négation, il est également possible d'exprimer des requêtes absurdes. En effet, il est désormais possible de demander quelque chose et son contraire ; la requête $q = ans(x) \leftarrow Films(Avatar, y, x), \neg Films(Avatar, y, x)$ (« lister tous les acteurs jouant dans le film Avatar et ne jouant pas dans le film Avatar ») en est un exemple, d'où la définition de requête absurde suivante :

Définition 2.9 ($RC^{\neg}$ absurde) *Soit q une $RC^{\neg}$ de la forme donnée dans la définition 2.7. Elle est* absurde *s'il existe $1 \leq i \leq n$ et $1 \leq j \leq m$ tels que $P_i = N_j$ et $u_i = v_j$. Sinon est elle dite* consistante.

Notons que l'ensemble des réponses à une requête absurde sera toujours vide et que la

réponse sera toujours faux si la requête est booléenne. Dans la suite de ce mémoire, toutes les requêtes considérées sont consistantes.

Concernant les trois problèmes de base vus plus haut (existence d'une réponse, calcul des réponses et inclusion de requêtes), la principale différence liée à l'ajout de la négation concerne le problème d'inclusion de requêtes conjonctives. En effet, les problèmes d'existence d'une réponse et de calcul des réponses ne diffèrent que par la sémantique d'une réponse (nous rappelons que la base de données est exclusivement composée d'informations positives sous l'hypothèse du monde clos). Par contre le problème d'inclusion de requêtes conjonctives devient beaucoup plus complexe lorsqu'on considère des $RC^{\neg}$; il est Π_2^p-complet [3] [Farré *et al.*, 2007][Chein et Mugnier, 2009], ce qui signifie intuitivement qu'il serait dans *Co-NP* si on pouvait appeler « gratuitement » un *oracle NP* (un algorithme résolvant un problème de *NP*) autant de fois que nous le voulons, et qu'il fait partie des problèmes les plus difficiles de Π_2^p. Il est « fortement » équivalent à d'autres problèmes importants en intelligence artificielle, dans le sens où il existe une réduction polynomiale naturelle d'un problème à l'autre et réciproquement préservant la structure des objets (ces équivalences seront vues plus en détail dans la section 2.3). À notre connaissance, il n'a été proposé dans la littérature que trois schémas d'algorithmes permettant de le résoudre (une étude détaillée des trois méthodes existantes est présentée en section 2.4). Même si tous les problèmes Π_2^p-complets sont théoriquement équivalents, c'est-à-dire qu'il existe une réduction polynomiale permettant de traduire l'un vers l'autre, certaines réductions ne sont pas immédiates, comme celle du problème d'inclusion de requêtes conjonctives avec négation ($IRC^{\neg}$) vers le problème *QSAT*, aussi connu sous le nom *QBF* pour *Quantified Boolean Formula* (ce point est discuté en section 2.4).

Dans la section suivante, nous présentons l'approche bases de connaissances et ses similarités et différences par rapport à l'approche BD.

2.2 Notions classiques en bases de connaissances

Dans cette partie, nous rappelons la structure classique d'une base de connaissances (BC) et l'illustrons par deux exemples concrets de langages de représentation des connaissances et raisonnements : les logiques de descriptions (LD) et les graphes conceptuels (GC). Nous présentons également les similitudes et les différences fondamentales apparaissant entre les BD et les BC.

2.2.1 La base de connaissances

Dans les systèmes de représentation des connaissances, nous nous intéressons à des bases de connaissances classiquement composées de deux parties distinctes : une ontologie et une base de faits.

3. $\Pi_2^p = (co\text{-}NP)^{NP}$ (voir [Papadimitriou, 1994] pour plus de détails sur la théorie de la complexité).

L'ontologie

Dans [Gruber, 1993], une ontologie est définie comme une spécification explicite d'une conceptualisation. Dans le contexte des bases de connaissances, une conceptualisation signifie un modèle abstrait d'une partie du monde, prenant la forme d'une définition des propriétés de concepts et de relations entre ces concepts. Une spécification explicite signifie que le modèle doit être spécifié dans un langage sans ambiguïté, le rendant manipulable aussi bien par des machines que par des humains. L'ontologie décrit le *vocabulaire* que l'on peut utiliser sous forme de types d'objets, appelés classes ou concepts, et de relations possibles entre objets. En outre, elle décrit des propriétés des concepts et des relations. Les concepts sont généralement organisés en une hiérarchie spécifiant une relation de spécialisation/généralisation, qui peut être donnée (on déclare explicitement qu'un concept est une spécialisation d'un autre) ou calculée (c'est-à-dire *déduite* en prenant en compte les propriétés des concepts). Il peut en être de même pour les relations, bien que ce soit moins souvent le cas.

Il existe différents types d'ontologie, chacun permettant un certain niveau d'expressivité. Ainsi, on peut distinguer au moins trois niveaux d'expressivité :

Niveau 1. *Le vocabulaire* : tous les éléments sont deux-à-deux incomparables. Ceci est le strict minimum.

Exemple 6.
Un ensemble de concepts : $C = \{Homme, Femme, Animal\ldots\}$;
Un ensemble de relations d'arité 2 : $R_2 = \{estPèreDe, estMèreDe\ldots\}$;
Un ensemble de relations d'arité 3 : $R_3 = \{joueAvec, mange\ldots\}$,...

Niveau 2. *L'ontologie légère* (*lightweight ontology* dans la littérature, [Giunchiglia *et al.*, 2006] par exemple) : cette ontologie est organisée en une hiérarchie de spécialisation/généralisation définissant un ordre ou un préordre [4] (voir figure 2.2).

Niveau 3. *Les ontologies lourdes* (*heavyweight ontology* dans la littérature, [Fürst et Trichet, 2006] par exemple) : une ontologie lourde est une ontologie légère enrichie avec plus ou moins d'axiomes utilisés pour fixer l'interprétation sémantique des concepts et des relations.

Notons qu'il est également possible d'inférer de nouvelles connaissances ontologiques à partir de définitions et d'assertions sur les propriétés des concepts et relations. Prenons par exemple l'ontologie de la figure 2.2 : supposons que nous ayons les propriétés « un *Camping-car* est un véhicule dans lequel on habite » et « un objet dans lequel on habite est une *Habitation* » ; d'après ces deux propriétés et le fait que le concept *Objet* est plus général que le concept *Véhicule*, nous en déduisons qu'un *Camping-car* est une spécialisation d'une *Habitation*. La figure 2.3 montre la hiérarchie de concepts obtenue après l'ajout de ce lien de spécialisation.

4. Un *préordre* est une relation réflexive et transitive mais pas forcément antisymétrique. Autrement dit, on peut avoir des éléments équivalents.

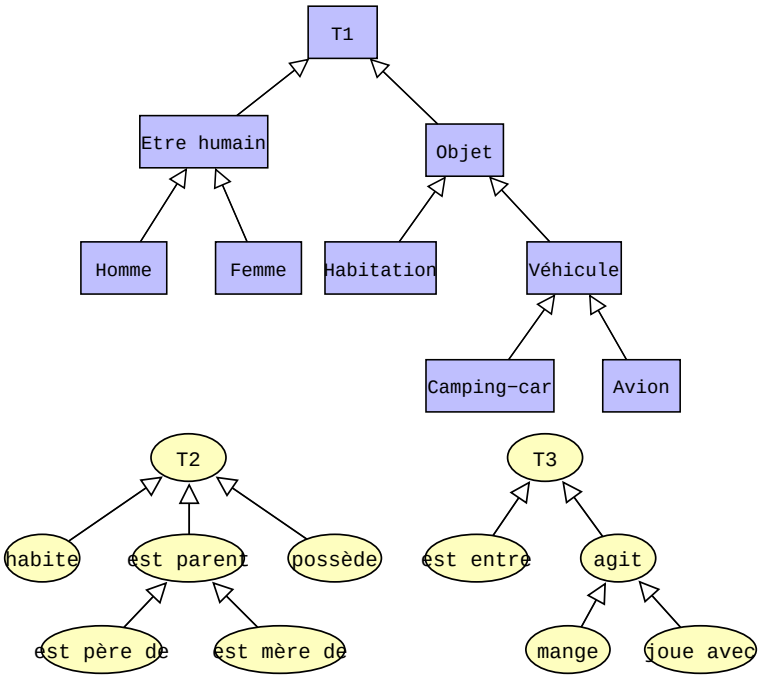


FIGURE 2.2 : un exemple simple d'ontologie organisée en une hiérarchie de spécialisation/-généralisation. Les rectangles et les ovales représentent respectivement les concepts et les relations ; T1, T2 et T3 sont les types universels, ils représentent respectivement « tout objet », « toute relation d'arité 2 » et « toute relation d'arité 3 ».

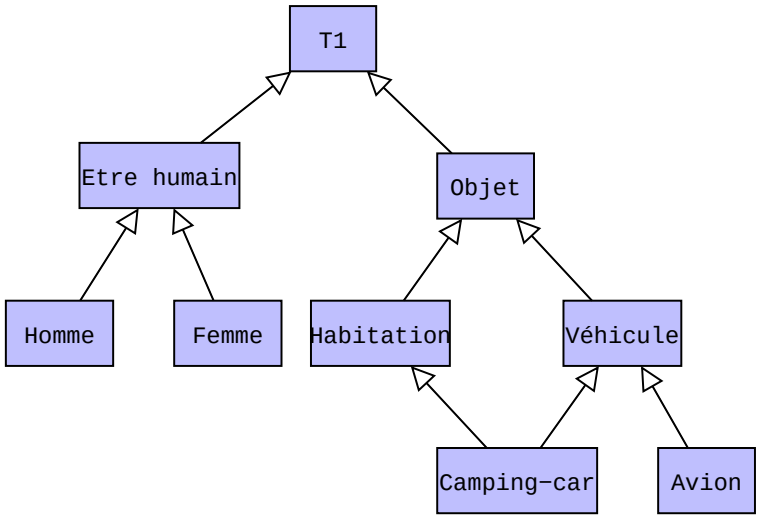


FIGURE 2.3 : la partie raffinée de l'ontologie de la figure 2.2.

Dans cette thèse, nous ne considérons que les niveaux 1 et 2 (ce dernier sera vu au chapitre 5).

La base de faits

La base de faits est constituée d'un ensemble de connaissances appelées observations factuelles ou faits, qui décrivent des objets particuliers ainsi que des relations entre ces objets en utilisant les termes de l'ontologie (par exemple : « l'homme A possède un camping-car » ...). Dans une base de faits nous pouvons exprimer des « faits négatifs » (par exemple « la femme B n'est pas la mère de A » ...), ce qui est une des principales différences avec une base de données.

2.2.2 Les logiques de description et les graphes conceptuels

Nous mentionnons brièvement deux familles de langages utilisant des ontologies : les Logiques de Description (LD) et les Graphes Conceptuels (GC). Les LD [Baader *et al.*, 2003] sont des fragments décidables de la logique du premier ordre. Une base de connaissance en LD est composée de deux parties différentes : la TBox (elle représente une ontologie de niveau 2 ou 3), composante intensionnelle qui définit les connaissances générales, et la ABox (elle représente la base de faits), composante extensionnelle, qui définit les données connues sur des individus. Les GC [Sowa, 1976][Sowa, 1984][Chein et Mugnier, 2009] sont un formalisme de représentation par graphes ayant une traduction en logique du premier ordre. Une base de connaissances en GC est également composée de deux parties différentes : le support (il représente une ontologie de niveau 2, qui pourra être enrichie par l'ajout de règles et de contraintes, ce qui conduira alors à une ontologie de niveau 3), qui décrit les ensembles de concepts et relations servant à étiqueter les sommets des graphes ainsi que leur ordonnancement par une relation de spécialisation/généralisation, et la base de faits, décrite par un ensemble de graphes bipartis étiquetés (une des classes de sommets représente les concepts et l'autre les relations entre ces concepts ou des propriétés sur ces concepts).

Ces deux familles de langages utilisant des ontologies sont issues des réseaux sémantiques (voir [Woods, 1975] par exemple), et ont visé à partir des années 80 des objectifs différents mais complémentaires : les LD se sont concentrées sur les raisonnements par classification dans une ontologie (recherche des éléments plus généraux ou plus spécifiques qu'un élément donné) ; l'idée générale étant d'associer des descriptions complexes à des entités - des concepts par exemple - et de se poser le problème de classifier de nouveaux concepts au sein d'un ensemble de concepts précédemment décrits ou de déterminer si un ensemble de concepts est satisfiable. Les LD ont ainsi développé des types d'axiomes ontologiques pour lesquels ces problèmes sont décidables et peuvent même se résoudre de façon efficace. En revanche, elles ne traitent que des cas limités de requêtes et règles d'inférence. Les GC se sont quant à eux concentrés sur la déduction entre faits,

problème fondamental pour résoudre le problème de calcul des réponses à une requête conjonctive et la prise en compte de règles d'inférence dans les raisonnements. Dans leur version de base, ils permettent l'expression de n'importe quelle requête conjonctive, mais ne disposent que d'expressions limitées d'axiomes ontologiques. Ces dernières années, les objectifs des LD et des GC ont eu tendance à se rejoindre, notamment pour traiter le problème de l'interrogation du web sémantique (voir [Berners-Lee *et al.*, 2001] par exemple) : les LD ont cherché à simplifier leurs axiomes ontologiques pour pouvoir traiter des requêtes conjonctives en limitant la hausse de complexité, tandis que les GC, qui englobent naturellement les requêtes conjonctives, ont cherché à exprimer des axiomes ontologiques plus riches à l'aide des règles d'inférence de leur formalisme.

2.2.3 Les correspondances et différences entre les BD et les BC

Nous présentons certaines similarités et différences entre les BD et les BC.

Similitudes

Comme vu en section 2.1, une BD est construite conformément à un schéma qui définit le vocabulaire utilisable. Ainsi, on peut mettre en parallèle les BD et les BC : le schéma de la BD peut être vu comme un *vocabulaire*, et les données de la BD comme des *faits* positifs. Précisons que le schéma peut être complété par des *contraintes* que l'on peut rapprocher de connaissances ontologiques, mais c'est un aspect que nous n'aborderons pas dans cette thèse. Néanmoins le sens contraire n'est pas possible car la base de faits comporte des faits négatifs.

Différences

Nous citons quelques différences importantes entre les deux approches :

(1). Les BD considèrent l'hypothèse du monde clos alors que les BC considèrent une autre hypothèse : celle du *monde ouvert*. L'hypothèse du monde ouvert suppose une connaissance incomplète du monde. De ce fait, les informations manquantes, ou plus généralement non-déductibles de la base de connaissances, sont simplement inconnues (et non pas fausses comme sous l'hypothèse du monde clos). L'information $\neg sur(A,B)$ est alors interprétée par « $\neg sur(A,B)$ apparaît dans la base de connaissances », ou plus généralement « $\neg sur(A,B)$ peut être déduit de la base de connaissances ». Tout comme l'hypothèse du monde clos, cette hypothèse est intéressante dans de nombreux cas pratiques, notamment celui du Web sémantique. Prenons l'exemple du remplissage d'un formulaire sur Internet : la valeur de certains champs est facultative, cependant si une personne ne remplit pas le champ *adresse* par exemple, cela ne veut pas dire qu'elle est sans-abri, de même si elle ne remplit pas le champ *téléphone*, cela ne veut pas dire qu'elle n'a pas de

numéro de téléphone. Ainsi certaines informations inconnues ne doivent pas automatiquement être considérées comme fausses comme c'est le cas sous l'hypothèse du monde clos.

(2). La notion de réponse à une $RC^{\neg}$ n'est plus aussi naturelle avec l'hypothèse du monde ouvert. Considérons deux requêtes $q_1 = ans(x,y) \leftarrow bleu(x), sur(x,y), \neg bleu(y)$ (« lister les entités telles qu'une entité "bleue" soit sur une entité "non-bleue" ») et $q_2 = ans() \leftarrow bleu(x), sur(x,y), \neg bleu(y)$ (« y a-t-il une entité "bleue" sur une entité "non-bleue" ? ») et une base de connaissances $BC = \{bleu(A), sur(A,B), sur(B,C), \neg bleu(C)\}$ (représentant que A est bleu, A est sur B, B est sur C, et C est non-bleu). Sous l'hypothèse du monde ouvert, nous supposons que la connaissance peut être incomplète ; ainsi puisque la couleur de B n'est pas connue, B peut être soit bleu soit non-bleu. Supposons que B soit bleu : il existe alors une réponse à q_2 (le tuple (B,C)) puisque B qui est bleu est sur C qui est non-bleu ; à l'inverse, supposons maintenant que B soit non-bleu : il existe également une réponse à q_2 (le tuple (A,B)) puisque A qui est bleu est sur B qui est non-bleu. Ainsi il existe une « réponse » à q_2 dans BC, mais il est impossible de la construire ($q_1(BC) = \emptyset$). La notion de réponse à une requête sous l'hypothèse du monde ouvert est discutée plus en détail dans le chapitre 5.

(3). L'objectif premier des BD est le stockage et la récupération des données stockées, alors que celui des BC est l'inférence de nouvelles connaissances à partir de celles déjà connues. Toutefois, le mouvement actuel tend à faire se rencontrer les deux approches, notamment sous la poussée du Web sémantique : les bases de connaissances s'intéressent aux inférences sur de grandes bases de faits (par exemple [Calvanese *et al.*, 2005]) et les bases de données à l'interrogation en monde ouvert (par exemple [Calì *et al.*, 2003]).

2.3 Reformulation en logique classique

La logique, et plus particulièrement la logique du premier ordre, est un formalisme de référence pour la représentation des connaissances et les raisonnements. Dans cette partie, nous nous plaçons dans le cadre du fragment conjonctif existentiel de la logique du 1^{er} ordre muni de la négation atomique, que nous noterons $FOL\{\exists, \wedge, \neg_a\}$ (nous ne considérons pas de symbole de fonction hormis des constantes). Nous donnons « la traduction » de certaines notions déjà exprimées en section 2.1.

2.3.1 Notions de base

Les définitions qui suivent traduisent dans $FOL\{\exists, \wedge, \neg_a\}$ respectivement les notions de schéma de BD, relation de BD, $RC^{\neg}$, $RC^{\neg}$ absurde et homomorphisme de RC.

Définition 2.10 (Langage logique, prédicat) *Un* langage logique *est une paire $(\mathcal{P},\mathcal{I})$, où $\mathcal{P}$ est un ensemble fini de prédicats et $\mathcal{I}$ un ensemble infini dénombrable de constantes. Les termes sur $(\mathcal{P},\mathcal{I})$ sont soient des constantes de $\mathcal{I}$ soient des variables. Un atome sur $(\mathcal{P},\mathcal{I})$ est de la forme $r(t_1,\ldots,t_k)$, où $r \in \mathcal{P}$ et $t_1,\ldots,t_k$ est un k-tuple de termes (pour tout $j = 1\ldots k$, t_j est un terme sur $(\mathcal{P},\mathcal{I})$). Un* littéral *est un atome (littéral positif) ou la négation d'un atome (littéral négatif).*

Définition 2.11 (Formule et sous-formule, fermeture existentielle) *Une* formule *f de $FOL\{\exists,\wedge,\neg_a\}$ est une conjonction de littéraux positifs ou négatifs* fermée existentiellement *(c'est-à-dire que chaque variable est quantifiée existentiellement). On la verra également comme un ensemble de littéraux. Une* sous-formule *f' de f est alors un ensemble de littéraux tel que $f' \subseteq f$ (l'ensemble des littéraux de f' est inclus dans l'ensemble des littéraux de f).*

Définition 2.12 (Formule inconsistante (ou insatisfiable)) *Une formule de $FOL\{\exists,\wedge,\neg_a\}$ est* inconsistante *si elle contient deux littéraux $r(t_1,\ldots,t_n)$ et $\neg r(t_1,\ldots,t_n)$. Autrement elle est* consistante.

Définition 2.13 (Homomorphisme) *Soient deux formules de $FOL\{\exists,\wedge,\neg_a\}$ f et g, un homomorphisme h de f dans g est une substitution des termes de f dans les termes de g (une constante ayant pour image elle-même) telle que $h(f) \subseteq g$. Nous indiquons l'existence d'un homomorphisme de f dans g par $f \succeq g$.*

Un schéma de BD $\mathcal{S} = \{\mathcal{R}, \mathbf{dom}\}$ est traduit en un langage logique par la bijection naturelle suivante : des relations de $\mathcal{R}$ vers les prédicats de $\mathcal{P} \cup \{ans_i\}$ (pour tout entier $i \geq 1$) et des constantes de **dom** vers les constantes de $\mathcal{I}$. Une $RC^\neg$ q est traduite en une formule f de $FOL\{\exists,\wedge,\neg_a\}$ par la bijection naturelle suivante : des variables de q vers les variables de f (chaque variable x produit une variable x, et chaque variable anonyme produit une *nouvelle* variable) des constantes de q vers les constantes de f, des littéraux positifs de q vers les littéraux positifs de f (avec de $ans(x_1,\ldots,x_n)$ vers $ans_n(x_1,\ldots,x_n)$) et des littéraux négatifs de q vers les littéraux négatifs de f. Ainsi il y a bijection de l'ensemble des $RC^\neg$ sur un schéma $\mathcal{S} = \{\mathcal{R}, \mathbf{dom}\}$ vers l'ensemble des formules de $FOL\{\exists,\wedge,\neg_a\}$ sur un langage logique $(\mathcal{P},\mathcal{I})$.

Notons que nous pouvons considérer des formules fermées existentiellement grâce à la traduction de l'atome $ans(x_1,\ldots,x_n)$ par un littéral positif $ans_n(x_1,\ldots,x_n)$, alors qu'en général une requête q est traduite en une formule logique *ouverte* avec les variables correspondant à $x_1,\ldots,x_n$ *libres* (c'est-à-dire non quantifiées). Néanmoins cette traduction est seulement valable pour traiter des problèmes prenant en entrée des requêtes (par exemple le problème d'inclusion de requêtes), car nous projetterons le littéral $ans_n(x_1,\ldots,x_n)$ de Q_2 sur le littéral $ans_n(y_1,\ldots,y_n)$ de Q_1.

Nous donnons les formules associées aux requêtes de l'exemple 1 :

Exemple 7.
(1) $f_1 = \exists x \exists y\ (ans_1(x) \wedge Films(Avatar, x, y))$
(2) $f_2 = \exists x\ (ans_1(x) \wedge Localisations(Royal, x))$
(3) $f_3 = \exists w \exists x \exists y \exists z\ (ans_1(x) \wedge Films(w, Petersen, y) \wedge Seances(x, w, z))$
(4) $f_4 = \exists x \exists y \exists z\ (Films(x, y, Pitt) \wedge Seances(z, x, 21h))$
(5) $f_5 = \exists x\ (ans_1(x) \wedge Films(Troie, Petersen, x) \wedge \neg Films(Avatar, Cameron, x))$

Dans la suite, nous utilisons la notation suivante pour les termes : des lettres minuscules pour représenter les variables et des lettres majuscules pour représenter les constantes.

Problèmes équivalents au problème $IRC^\neg$

Le problème $IRC^\neg$ est équivalent à deux problèmes fondamentaux en logique du premier ordre : le problème de déduction de formules et le problème d'implication de clauses.

Le problème de déduction est un problème de base en logique classique : « étant données deux formules f et g, f se déduit-elle de g (noté $g \vdash f$) ? » (c'est-à-dire intuitivement est-ce que les informations contenues dans f se trouvent dans g ?). Par la suite, nous considérons la restriction de ce problème à des formules de $FOL\{\exists, \wedge, \neg_a\}$, et nous l'appellerons simplement *Déduction*. Ce problème est équivalent au problème $IRC^\neg$ en bases de données, comme le montrent les réductions polynomiales naturelles vues en 2.3.1 ; nous manipulons les mêmes objets sous des formes différentes, et nous posons la même question. Nous avons alors la propriété suivante : soient q_1 et q_2 deux $RC^\neg$, et g et f les formules de $FOL\{\exists, \wedge, \neg_a\}$ qui leur sont associées, $q_1 \sqsubseteq q_2$ si et seulement si $g \vdash f$ (ceci peut se prouver par exemple en s'appuyant sur le théorème 2.19 en section 2.4.1).

Le problème *Déduction* est également équivalent à un autre problème fondamental en programmation logique inductive, le problème d'implication de clauses (sans fonction, comme dans [Gottlob, 1987]) : « étant données deux clauses c_1 et c_2, c_1 implique-t-elle c_2, c'est-à-dire c_2 se déduit-elle de c_1 ? » L'équivalence est immédiate : si nous prenons la négation des formules de $FOL\{\exists, \wedge, \neg_a\}$, nous obtenons directement des clauses dans un autre fragment de la logique du premier ordre, le fragment universel disjonctif, noté $FOL\{\forall, \vee, \neg_a\}$. L'équivalence provient de ce que $g \vdash f$ si et seulement si $\neg f \vdash \neg g$.

Exemple 8. Soient $f = \exists x \exists y\ ans(x, y) \wedge p(x) \wedge s(x, y) \wedge \neg p(y)$ et $g = \exists t \exists u \exists v \exists w\ ans_2(t, u) \wedge p(t) \wedge s(t, u) \wedge s(u, v) \wedge s(v, w) \wedge \neg p(w)$ deux formules de $FOL\{\exists, \wedge, \neg_a\}$. Les clauses c_1 et c_2, associées respectivement à f et g, sont obtenues en prenant la négation de chaque formule : nous obtenons $c_1 = \forall x \forall y\ (\neg ans_2(x, y) \vee \neg p(x) \vee \neg s(x, y) \vee p(y))$ et $c_2 = \forall t \forall u \forall v \forall w\ \neg ans_2(t, u) \vee \neg p(t) \vee \neg s(t, u) \vee \neg s(u, v) \vee \neg s(v, w) \vee p(w)$.

Nous n'avons pas défini la notion de réponse à une $RC^\neg$ en présence du monde ouvert, car elle ne va pas de soi (voir la discussion en section 2.2.3). On verra au chapitre 5 que le problème du calcul des réponses à une $RC^\neg$ non-booléenne n'est plus aussi naturel que sous l'hypothèse du monde clos.

2.3.2 Vision graphes

Notre vision « graphes » des formules de $FOL\{\exists, \wedge, \neg_a\}$ s'inspire des graphes conceptuels, modèle qui a été introduit par Sowa [1976], en particulier pour représenter les schémas conceptuels utilisés en BD, et développé en 1984 [Sowa, 1984], avec notamment des applications dans les domaines de l'intelligence artificielle et des sciences cognitives. L'une des directions de recherche actuelles développe les graphes conceptuels comme un formalisme de représentation de connaissances et de raisonnements ; une des particularités de ce modèle est de permettre de représenter des connaissances sous forme graphique, plus précisément sous forme de graphes. Pour plus de détails, nous dirigeons le lecteur vers [Chein et Mugnier, 2009], qui propose une synthèse des résultats théoriques et algorithmiques obtenus sur les graphes conceptuels. Dans cette thèse, nous avons choisi de représenter les formules de $FOL\{\exists, \wedge, \neg_a\}$ comme des graphes conceptuels *simplifiés* (nous considérons un *vocabulaire*, et les termes ne sont pas typés), ce qui permet une meilleure lisibilité (pour de petits graphes) mais surtout de s'intéresser à leur structure et de manipuler des notions de graphe qui n'ont pas d'équivalent simple en logique (comme celle de « sous-graphe » vue plus loin). Plus précisément, une formule f est représentée par un graphe F biparti, non orienté et étiqueté, appelé *graphe polarisé* (cette appellation a été introduite par Kerdiles dans sa thèse sur les graphes conceptuels [Kerdiles, 2001]).

Définition 2.14 (Graphe polarisé [Chein et Mugnier, 2009]) *Nous considérons un vocabulaire $\mathcal{V} = (\mathcal{R}, \mathcal{C})$ où $\mathcal{R}$ est un ensemble fini de relations de n'importe quelle arité et $\mathcal{C}$ un ensemble de constantes. Un* graphe polarisé *sur $\mathcal{V}$ est un 4-tuple $G = (T, R, E, l)$ satisfaisant les conditions suivantes :*

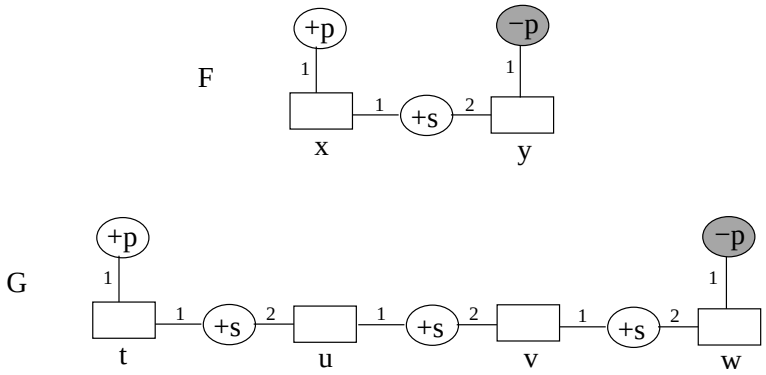
1. *(T, R, E) est un multi-graphe biparti, non orienté et étiqueté. T est l'ensemble des* sommets termes*, R l'ensemble des* sommets relations *et E la famille des* arêtes *(plusieurs arêtes pouvant avoir la même extrémité, nous parlons de multi-graphe et pas simplement de graphe).*
2. *l est la fonction d'étiquetage des sommets et des arêtes de G qui satisfait :*
   - **(a)** *Un sommet terme $t \in T$ peut être étiqueté par $l(t) \in \mathcal{C}$ (t est appelé sommet constante) ou ne pas être étiqueté (t est appelé sommet variable).*
   - **(b)** *Un sommet $s \in R$ est étiqueté par $l(s) = +r$ (s est appelé sommet relation positif) ou $l(s) = -r$ (s est appelé sommet relation négatif), où $r \in \mathcal{R}$ et $+$ (resp. $-$) est le signe de r. Le degré de s (c'est-à-dire le nombre d'arêtes incidentes à s) est égal à l'arité de r. Les arêtes incidentes à s sont totalement ordonnées, ce qu'on représente en les étiquetant de 1 à* arité*(r). Une arête étiquetée i entre un sommet relation s et un sommet terme t est notée (s, i, t).*

Un graphe polarisé G est dit sous *forme normale* si pour chaque constante $C \in \mathcal{C}$, il existe au plus un sommet terme t tel que $l(t) = C$ dans G. Dans la suite, les graphes polarisés sont supposés être sous forme normale et ne pas présenter de redondance de sommets

relations (c'est-à-dire deux sommets relations de même étiquette et ayant les mêmes voisins dans le même ordre).

Une formule f de $FOL\{\exists, \wedge, \neg_a\}$ sur un langage logique $(\mathcal{P},\mathcal{J})$ est traduite en un graphe polarisé F sur un vocabulaire $\mathcal{V} = (\mathcal{R}, \mathcal{C})$ par les bijections naturelles suivantes : des variables de f vers les sommets variables de F (chaque variable produit un nouveau sommet variable), des constantes de f vers les sommets constantes de F (chaque constante A produit un sommet constante d'étiquette A), des littéraux positifs (resp. négatifs) de f vers les sommets relations positifs (resp. négatifs) de F (le « signe » et le prédicat d'un littéral produit l'étiquette du sommet relation). Pour chaque terme t_i d'un littéral l, il y a une arête (s, i, t), où s est le sommet relation associé à l et t le sommet terme associé à t_i (cf. figure 2.4). Ainsi, il y a bijection de l'ensemble des formules de $FOL\{\exists, \wedge, \neg_a\}$ sur un langage logique $(\mathcal{P},\mathcal{J})$ vers l'ensemble des graphes polarisés sous forme normale et sans sommets termes isolés sur un vocabulaire $\mathcal{V} = (\mathcal{R}, \mathcal{C})$.

Notations. On note une formule de $FOL\{\exists, \wedge, \neg_a\}$ par une lettre minuscule et son graphe polarisé associé par la majuscule de la même lettre. On note $+r(t_1, \ldots, t_k)$ (resp. $-r(t_1, \ldots, t_k)$) un sommet relation étiqueté $+r$ (resp. $-r$) dont la liste de voisins est $t_1, \ldots, t_k$. Des sommets relations $+r(t_1, \ldots, t_k)$ et $-r(t'_1, \ldots, t'_k)$ ayant même relation mais des signes différents sont dits *opposés*. Des sommets relations opposés $+r(t_1, \ldots, t_k)$ et $-r(t_1, \ldots, t_k)$ ayant même voisinage sont dits *contradictoires*. $\sim r$ désigne une étiquette de sommet relation de signe quelconque et $\overline{\sim r}$ désigne l'étiquette opposée.



$$f = \exists x \exists y\ (p(x) \wedge s(x, y) \wedge \neg p(y))$$
$$g = \exists t \exists u \exists v \exists w\ (p(t) \wedge s(t, u) \wedge s(u, v) \wedge s(v, w) \wedge \neg p(w))$$

FIGURE 2.4 : les graphes polarisés associés à f et g.

Un graphe polarisé est dit *consistant* s'il ne possède pas deux sommets relations contradictoires (ce qui correspond exactement à la notion de consistance de la formule associée).

Une notion fondamentale dans cette étude est celle d'homomorphisme. Un *homomorphisme* h de F dans G est une application des sommets de F dans les sommets de

G, qui respecte la bipartition (un sommet terme - resp. relation - a pour image un sommet terme - resp. relation), respecte les arêtes et leur numérotation (si $(r,i,t) \in F$ alors $(h(r),i,h(t)) \in G$), conserve les étiquettes des sommets relations (un sommet relation et son image ont même étiquette) et peut « instancier » les étiquettes des sommets termes (si un sommet terme est étiqueté par une constante, son image a la même étiquette, sinon son image peut avoir une étiquette quelconque).

Définition 2.15 (Homomorphisme) *Soient $F = (T_F, R_F, E_F, l_F)$ et $G = (T_G, R_G, E_G, l_G)$ deux graphes polarisés. Un homomorphisme h de F dans G est une application des sommets de F dans les sommets de G telle que :*

1. $\forall (r,i,t) \in F, (h(r),i,h(t)) \in G$;
2. $\forall r \in R_F, l_F(r) = l_G(h(r))$;
3. $\forall t \in T_F$, *si t est un sommet constante alors* $l_F(t) = l_G(h(t))$.

Dans la suite, lorsqu'on s'intéresse à la recherche d'un homomorphisme de F dans G, on appelle F le graphe *source* et G le graphe *cible*. Nous indiquons l'existence d'un homomorphisme de F dans G par $F \succeq G$.

Il est à noter que la notion d'homomorphisme sur les graphes polarisés correspond exactement à celle vue sur les formules : $F \succeq G$ si et seulement si $f \succeq g$. Dans la suite, nous utilisons de manière équivalente les notations $g \vdash f$ et $G \vdash F$ pour parler de la déduction de la formule associée à F à partir de celle associée à G.

2.4 État de l'art algorithmique sur le problème *Déduction*

Dans cette partie, nous nous focalisons sur le problème *Déduction* dans $FOL\{\exists, \wedge, \neg_a\}$ et proposons une synthèse des trois différentes approches algorithmiques pour le résoudre que nous connaissons. Il est à noter que les seuls schémas d'algorithme trouvés sur ce problème ou sur un problème « directement » équivalent (c'est-à-dire avec des réductions simples) sont en BD pour le problème $IRC^{\neg}$.

2.4.1 Impact de l'ajout de la négation sur le problème *Déduction*

Pour des graphes polarisés uniquement positifs F et G, l'homomorphisme est correct et complet :

Théorème 2.16 *[Sowa, 1984][Chein et Mugnier, 1992]*[5] *Soient deux graphes polarisés positifs F et G, $G \vdash F$* si et seulement s'il *existe un homomorphisme de F dans G.*

5. Notons qu'un résultat similaire pour l'inclusion de requêtes conjonctives était déjà présent dans [Chandra et Merlin, 1977].

Par contre, dès que l'on considère la négation atomique, un seul sens de la propriété reste vrai :

Propriété 2.1 *Soient deux graphes polarisés F et G,* s'il existe *un homomorphisme de F dans G alors $G \vdash F$.*

La réciproque n'est plus vraie comme le montre l'exemple 9.

Exemple 9. Considérons les formules f et g de la figure 2.4. Il n'y a pas d'homomorphisme de F dans G alors que $g \vdash f$: il suffit de « compléter » g par rapport au prédicat p pour s'en persuader. On obtient alors la formule g' (équivalente à g) suivante : $g' = (g \wedge p(u) \wedge p(v)) \vee (g \wedge \neg p(u) \wedge p(v)) \vee (g \wedge p(u) \wedge \neg p(v)) \vee (g \wedge \neg p(u) \wedge \neg p(v))$. Chacune des quatre conjonctions de g' correspond à une façon de compléter g par rapport au prédicat p. On peut vérifier qu'il existe un homomorphisme de F dans chacun des graphes qui leur sont associés (l'image de F par chacun de ces homomorphismes est en pointillé sur la figure 2.5). f se déduit donc de g'.

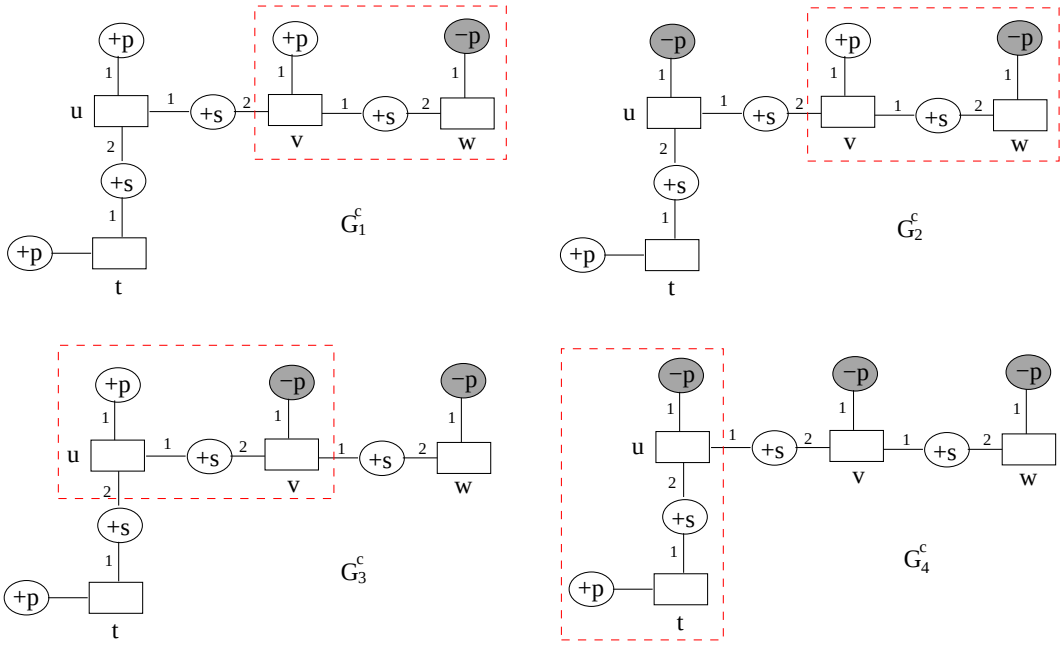


FIGURE 2.5 : les quatre graphes représentant les quatre conjonctions de g'.

La prise en compte de la négation atomique fait ainsi augmenter la complexité du problème *Déduction*. En effet sans négation, un test d'homomorphisme (problème *NP-complet*) suffit à savoir si une formule se déduit d'une autre. Dès que la négation est prise en compte, un seul test ne suffit pas comme le montre le cas très simple de l'exemple 9

(quatre tests sont nécessaires). Plus généralement, dans le pire des cas il est possible d'effectuer un nombre exponentiel de fois le test d'homomorphisme. La complexité passe de *NP-complet* à Π_2^p *-complet*[6] avec la prise en compte de la négation [Farré *et al.*, 2007][Chein et Mugnier, 2009].

Nous donnons les définitions concernant la notion de « compléter » et le théorème de [Leclère et Mugnier, 2007] qui fait le lien avec le problème *Déduction* :

Définition 2.17 (Graphe complet) *Un graphe G consistant est dit* complet*, noté G^c, par rapport à un ensemble de relations $\mathcal{R}$ si pour chaque relation r de $\mathcal{R}$ d'arité k et chaque k-tuple de termes (non nécessairement distincts) $t_1, \ldots, t_k$ de G, G contient soit $+r(t_1, \ldots, t_k)$ soit $-r(t_1, \ldots, t_k)$.*

Définition 2.18 (Complétion, Littéral de complétion) *Une* complétion *G' d'un graphe G consistant est un graphe obtenu de G par ajouts successifs de nouveaux sommets relations non-redondants (étiquetés par des relations apparaissant déjà dans G et portant sur des sommets termes déjà dans G), appelés* littéraux de complétion*, aussi longtemps qu'aucune inconsistance n'apparaît ($G \subseteq G'$). Chaque ajout représente une* étape de complétion*. Une complétion de G est dite* totale *si c'est un graphe complet par rapport à l'ensemble des relations apparaissant dans G, sinon elle est dite* partielle.

Théorème 2.19 *[Leclère et Mugnier, 2007] Soient deux graphes polarisés F et G (avec G consistant), $G \vdash F$* si et seulement si *quelque soit G^c, une complétion totale de G par rapport à l'ensemble des relations apparaissant dans G, il existe un homomorphisme de F dans G^c.*

Dans la suite, nous situons les travaux de [Ullman, 1997] et [Wei et Lausen, 2003] dans le cadre des graphes polarisés, même si à l'origine ils ne sont pas formulés de cette façon : ils s'intéressent au problème $IRC^\neg$ (« $q_1 \sqsubseteq q_2$? »), et considèrent la partie positive de q_1 comme une BD (soit D). Ainsi D représente une complétion totale dans le sens où toutes les informations manquantes peuvent être représentées par des sommets relations négatifs ; l'ajout ou le retrait d'une information positive permet de passer d'une complétion totale à une autre. Pour faire ce changement de formalisme, nous associons deux graphes polarisés F et G à deux $RC^\neg$ q_2 et q_1 et le test « $q_1 \sqsubseteq q_2$? » devient « $G \vdash F$? ».

2.4.2 L'approche de Ullman

Comme le suggère l'exemple 9, une des façons de résoudre le problème *Déduction* consiste à générer l'ensemble des complétions totales que l'on peut obtenir à partir de G en utilisant les relations apparaissant dans G, puis à tester s'il existe un homomorphisme de F dans chacun de ces graphes. La figure 2.6 illustre cette méthode, qui correspond à l'approche introduite par Ullman [1997] (même si, comme expliqué plus haut, elle ne manipule pas explicitement des complétions au sens de la définition 2.18).

6. $\Pi_2^p = (co\text{-}NP)^{NP}$

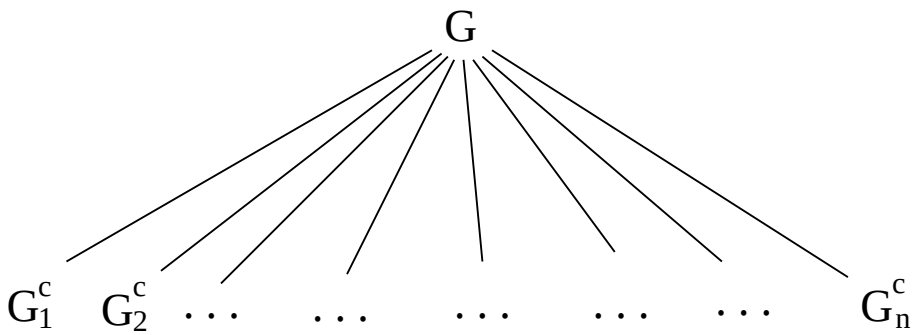


FIGURE 2.6 : la méthode de Ullman.

Néanmoins, cette approche brutale entraîne une complexité exorbitante : l'algorithme est en $\mathcal{O}(2^{(n_G)^k \times |\mathcal{P}_G|} \times hom(F, G^c))$, où n_G est le nombre de sommets termes dans G, k est l'arité maximum d'un sommet relation de G, $\mathcal{P}_G$ est l'ensemble des relations apparaissant dans G et $hom(F, G^c)$ est la complexité du test d'existence d'un homomorphisme de F dans G^c (sa résolution par un algorithme brutal est en $\mathcal{O}(n_G^{n_F})$, où n_F est le nombre de sommets termes de F).

Des améliorations sont proposées dans [Wei et Lausen, 2003] et [Leclère et Mugnier, 2007]. Elles concernent les deux points suivants :

1. Des conditions nécessaires pour l'inclusion sont proposées ; elles peuvent être utilisées pour essayer de détecter un échec avant de générer les complétions ;
2. Les complétions peuvent être construites et vérifiées incrémentalement (c'est-à-dire par étape).

2.4.3 L'approche de Wei et Lausen

Les auteurs de [Wei et Lausen, 2003] proposent un nouvel algorithme pour le problème $IRC^\neg$: celui-ci exploite les homomorphismes de la partie positive de F (composée de tous les sommets relations positifs de F) dans la partie positive de G. Il est affirmé dans [Wei et Lausen, 2003] que dans le pire des cas le nombre de complétions totales générées n'est pas pire que [Ullman, 1997] mais nous réfutons cette affirmation au chapitre 4. Dans cette approche, les requêtes sont supposées être toutes *safe*.

Condition nécessaire pour la déduction

Une condition nécessaire (mais non suffisante) est proposée : si $G \vdash F$ alors il doit exister un homomorphisme, noté h, de la partie positive de F (composée de tous les sommets relations positifs de F), notée F^p, dans G^p ; de plus, cet homomorphisme ne doit pas contredire les sommets relations négatifs de F : pour chaque sommet relation $\neg r(t_1, \ldots, t_k)$

dans F, G ne doit pas contenir $+r(h(t_1),\ldots,h(t_k))$. Cette propriété peut être utilisée comme un filtre : s'il n'y a pas un tel homomorphisme de F^p dans G^p, alors $G \not\vdash F$.

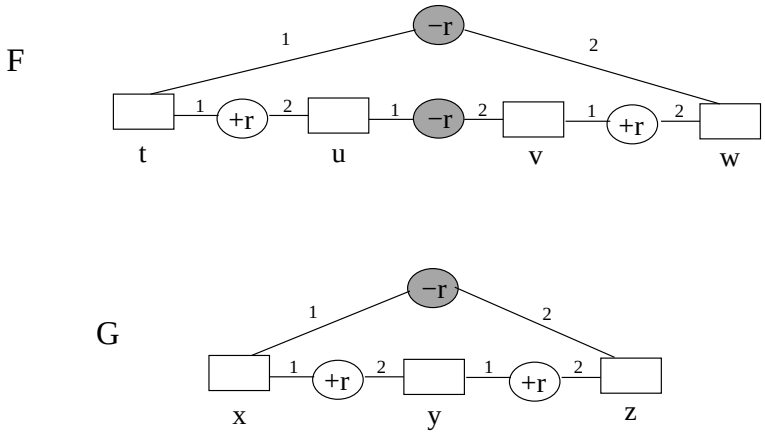
Construction incrémentale des complétions

Le but de la construction incrémentale des complétions est de ne pas avoir à parcourir l'ensemble des complétions totales pour répondre si oui ou non F se déduit de G ; ainsi on cherche un ensemble de complétions partielles couvrant l'ensemble des complétions totales. Cette construction est directement liée à la condition nécessaire pour la déduction suivante : on a $G \vdash F$ si et seulement si (1) il existe un homomorphisme h de F^p dans G^p, et (2) pour chaque sommet relation négatif $-r(t_1,\ldots,t_k)$ de F, on a $G' \vdash F$, où G' est la complétion de G obtenue par l'ajout de $+r(h(t_1),\ldots,h(t_k))$.

L'algorithme proposé explore l'espace de recherche menant de G à ses complétions totales par ce qui « ressemble » à un parcours en largeur [7]. Nous employons le terme « ressembler » car nous montrons au chapitre 4 qu'une complétion peut être parcourue plusieurs fois. On peut voir cet espace de recherche (cf. [Leclère et Mugnier, 2007]) comme partiellement ordonné par la relation d'inclusion « sous-graphe de » (notée $\subseteq$), G formant la racine de l'arbre de recherche. Tous les homomorphismes $h_1,\ldots,h_n$ de F^p dans G^p sont calculés, et une branche est construite pour chacun d'eux à partir de la racine. Pour chaque branche h_i et pour chaque sommet relation négatif $-r(t_1,\ldots,t_k)$ de F, on construit un nœud $N_{i,j}$ à partir de h_i, correspondant à une complétion obtenue de G par l'ajout de $+r(h_i(t_1),\ldots,h_i(t_k))$ (cf. figure 2.8). L'idée est que toute complétion de G contient soit $-r(h_i(t_1),\ldots,h_i(t_k))$, soit $+r(h_i(t_1),\ldots,h_i(t_k))$. Dans le premier cas, l'homomorphisme h_i peut être étendu. Dans le deuxième cas, il reste à prouver que F se déduit de $G \cup \{+r(h_i(t_1),\ldots,h_i(t_k))\}$. Chaque nœud $N_{i,j}$ dont la complétion associée est inconsistante (le dernier sommet relation ajouté est contradictoire avec un sommet relation négatif de G) est marqué comme « contenu » (il « étend » l'homomorphisme h_i) et chaque nœud $N_{i,j}$ dont la complétion associée est redondante (le dernier sommet relation ajouté est égal à un sommet relation positif de G) est marqué comme « terminal » (il contredit l'homomorphisme h_i). S'il existe une branche (partant de la racine) valide, c'est-à-dire que pour toutes complétions G' associées aux nœuds se trouvant dans le sous-arbre de racine h_i, alors $G \vdash F$. S'il existe un nœud $N_{i,j}$ tel qu'au moins un nœud de chaque branche partant de $N_{i,j}$ est marqué « terminal » (aucun homomorphisme ne vérifie la condition nécessaire susmentionnée), alors $G \not\vdash F$. Sinon, les nœuds non marqués sont étendus en prenant en compte les *nouveaux* homomorphismes et le processus recommence.

Néanmoins cet algorithme semble poser deux problèmes principaux, l'un concernant la notion de *nouveau* homomorphisme et l'autre la détection des complétions partielles identiques ; ils sont discutés au chapitre 4.

7. Un algorithme de parcours en largeur d'un graphe fonctionne de la manière suivante : (1) Choisir un sommet de départ et le mettre dans une file. (2) Tant que la file n'est pas vide, itérer : (a) Retirer le sommet du début de la file pour l'examiner. (b) Mettre tous ses voisins non-examinés dans la file (à la fin).



$f = \exists t \exists u \exists v \exists w \; r(t,u), r(v,w), \neg r(t,w), \neg r(u,v)$
$g = \exists x \exists y \exists z \; r(x,y), r(y,z), \neg r(x,z)$

FIGURE 2.7 : les graphes associés à f et g.

Exemple 10. Nous reprenons l'exemple décrit dans [Wei et Lausen, 2003] en le formalisant sous forme de graphes polarisés (cf. figure 2.7). Nous allons conclure que $G \nvdash F$.

Les sommets relations positifs de G forment la racine de l'arbre construit. Il y a quatre homomorphismes $h_1, \ldots, h_4$ de F^p dans G^p (table 2.1, étape 1). Une branche h_i partant de la racine est construite pour chacun d'eux. Chaque branche h_i est composée de deux nœuds, chacun étant associé à la complétion obtenue de G par l'ajout de l'image d'un sommet relation négatif de F par l'homomorphisme h_i (cf. figure 2.8). Puis chaque nœud dont la complétion associée est inconsistante est marqué comme « contenu » (ici $+r(x,z)$, il « étend » l'homomorphisme h_1) et chaque nœud dont la complétion associée est redondante est marqué comme « terminal » ($+r(x,y)$ et $+r(y,z)$, ils contredisent respectivement h_2 et h_3). Sur la figure 2.8, un nœud contenu est représenté par un ovale et un nœud terminal par un rectangle. Les nœuds non marqués sont alors étendus : ici $+r(y,y)$ qui conduit à un échec pour h_1, on en conclut donc que $G \nvdash F$; intuitivement, il n'y a pas d'homomorphisme de F dans la complétion totale G^c obtenue de G en ajoutant un seul sommet relation positif $+r(y,y)$ et les autres sommets relations en négatif ($-r(x,x), -r(z,z)\ldots$) ; G^c est un contre-exemple.

2.4.4 L'approche de Leclère et Mugnier

Une première propriété de [Leclère et Mugnier, 2007] est que les relations n'apparaissant pas à la fois dans des sommets relations positifs et négatifs dans F et dans G ne sont pas utiles au calcul des complétions. On peut donc restreindre l'ensemble de relations considéré à un ensemble appelé le vocabulaire de complétion :

	Étape 1		Étape 2
h_1	$t \mapsto x, u \mapsto y, v \mapsto y, w \mapsto z$	h_5	$t \mapsto y, u \mapsto y, v \mapsto y, w \mapsto$ y
h_2	$t \mapsto x, u \mapsto y, v \mapsto x, w \mapsto y$	h_6	$t \mapsto x, u \mapsto y, v \mapsto y, w \mapsto$ y
h_3	$t \mapsto y, u \mapsto z, v \mapsto y, w \mapsto z$	h_7	$t \mapsto y, u \mapsto y, v \mapsto x, w \mapsto$ y
h_4	$t \mapsto y, u \mapsto z, v \mapsto x, w \mapsto y$	h_8	$t \mapsto y, u \mapsto y, v \mapsto y, w \mapsto$ z
		h_9	$t \mapsto y, u \mapsto z, v \mapsto y, w \mapsto$ y

TABLE 2.1 : les homomorphismes de l'exemple 10.

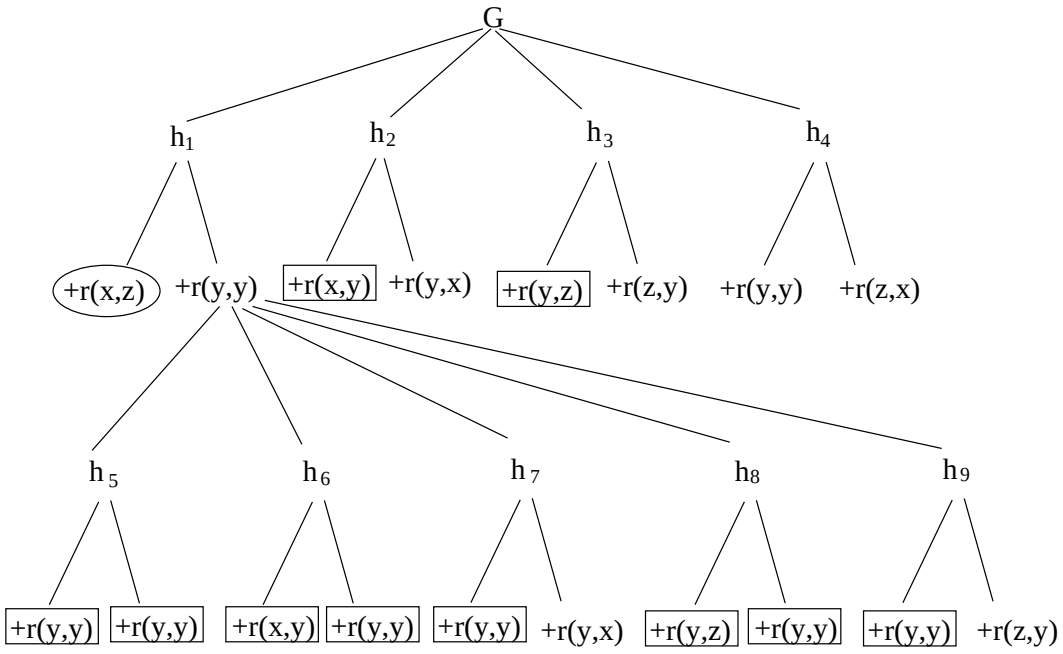


FIGURE 2.8 : l'arbre généré pour l'exemple 10.

Définition 2.20 *[Vocabulaire de complétion] Soient deux graphes polarisés F et G, un* symbole de complétion *de F et G est une relation apparaissant dans des sommets relations de signe opposé, à la fois dans F et dans G. Le* vocabulaire de complétion *de F et G, noté $\mathcal{V}_c$, est composé de l'ensemble des symboles de complétion de F et G.*

Condition nécessaire pour l'inclusion

Dans [Leclère et Mugnier, 2007], le premier niveau d'amélioration consiste à identifier des « sous-graphes » de F pour lesquels il existe nécessairement un homomorphisme dans G quand $G \vdash F$.

Définition 2.21 (Sous-Graphe [8]) *Un sous-graphe $F' = (T', R', E', l')$ de $F = (T, R, E, l)$, noté $F' \subseteq F$, est un graphe polarisé tel que $T' = T$, $R' \subseteq R$, E' est la restriction de E à R', et l' est la restriction de l à (T', R', E').*

Un exemple de sous-graphe ayant cette propriété est celui des sous-graphes *purs* :

Définition 2.22 (Graphe polarisé pur) *Un graphe polarisé* pur *ne contient pas de sommets relations opposés (c'est-à-dire toute relation n'y apparaît que sous une forme, positive ou négative).*

De tels sous-graphes sont utilisés pour mettre en place un filtrage :

Propriété 2.2 *Soient F et G deux graphes polarisés. S'il n'y a pas d'homomorphisme d'un sous-graphe pur de F dans G alors $G \not\vdash F$.*

De plus, il est imposé qu'un homomorphisme d'un sous-graphe de F dans G soit « compatible » avec un homomorphisme de F dans une complétion totale de G (il est à noter que cette notion était implicitement présente dans [Wei et Lausen, 2003], cf. condition nécessaire). D'où la définition suivante :

Définition 2.23 *[Frontière, Homomorphisme compatible] Soient deux graphes polarisés F et G et F' un sous-graphe de F. Les sommets relations de $F \setminus F'$ sont appelés* sommets relations frontières *(de F' par rapport à F). Un homomorphisme h de F' dans G est dit* compatible *par rapport à F si :*

- *pour chaque sommet relation $\sim r(t_1, \ldots, t_k)$ de $F \setminus F'$, il n'existe pas de sommet relation de signe opposé $\overline{\sim r}(h(t_1), \ldots, h(t_k))$ dans G.*
- *pour chaque paire de sommets relations de signe opposé $\sim r(c_1, \ldots, c_k)$ et $\overline{\sim r}(d_1, \ldots, d_k)$ de $F \setminus F'$, $(h(c_1), \ldots, h(c_k)) \neq (h(d_1), \ldots, h(d_k))$.* [9]

9. Cette condition n'était pas présente dans [Leclère et Mugnier, 2007] (ce qui ne remet pas en cause les résultats de ce papier, cette condition étant forcément satisfaite si F' est un sous-graphe pur maximal pour l'inclusion), mais elle est nécessaire pour assurer que h constitue un homomorphisme de F dans une complétion de G quel que soit F'.

Il est à noter que la notion de sous-graphe pur généralise la notion de partie positive de F proposée par [Wei et Lausen, 2003].

Une notion plus forte est celle de sous-graphe ne possédant pas de *littéraux échangeables*, c'est-à-dire de sommets relations $\sim r(c_1,\ldots,c_k)$ et $\overline{\sim r}(d_1,\ldots,d_k)$ tels qu'il existe deux complétions totales de G, G_1 et G_2, et deux homomorphismes h_1 et h_2, respectivement de F dans G_1 et de F dans G_2 avec $(h_1(c_1),\ldots,h_1(c_k)) = (h_2(d_1),\ldots,h_2(d_k))$. On a ainsi :

Théorème 2.24 *[Leclère et Mugnier, 2007] Si $G \vdash F$ alors pour tout sous-graphe F' de F sans littéraux échangeables, il existe un homomorphisme compatible de F' dans G par rapport à F.*

Néanmoins, il y a une grande différence de complexité entre la reconnaissance d'un sous-graphe pur et celle d'un sous-graphe sans littéraux échangeables. En effet, le problème de reconnaissance d'un sous-graphe sans littéraux échangeables est *NP-complet* (algorithme exponentiel en pratique) alors qu'un test polynomial suffit pour un sous-graphe pur. Ainsi, toutes les heuristiques présentes dans cette thèse sont basées sur les signes des sommets relations, bien qu'il serait possible de les affiner mais au prix d'un prétraitement beaucoup plus coûteux.

Nous énumérons les différents sous-graphes purs de F mentionnés dans [Leclère et Mugnier, 2007] et utilisés par la suite :

- Sous-graphes purs maximaux : nous appelons F^{Max} un sous-graphe pur de F de cardinalité maximale et F^{max} un sous-graphe pur maximal pour l'inclusion.
- Sous-graphe positif (resp. négatif) : nous appelons F^+ (resp. F^-) le sous-graphe pur positif (resp. négatif) de F obtenu par suppression de tous les sommets relations négatifs (resp. positifs) de F.

Construction incrémentale des complétions

Dans [Leclère et Mugnier, 2007], une exploration incrémentale de l'espace de recherche menant de G à ses complétions totales est effectuée. Cet espace de recherche est partiellement ordonné par la relation d'inclusion « sous-graphe de » (notée $\subseteq$). Au lieu de construire et tester toutes les complétions totales de G, on recherche un ensemble de complétions partielles couvrant ces complétions totales, c'est-à-dire la question de savoir s'il existe un homomorphisme de F dans chaque complétion totale de G devient : « Existe-t-il un ensemble de complétions partielles $\{G_1,\ldots,G_n\}$ tel que (1) il existe un homomorphisme de F dans chaque G_i pour $i = 1\ldots n$; (2) chaque complétion totale G^c de G est couverte par un G_i, c'est-à-dire $\forall G^c\ \exists G_i \mid G_i \subseteq G^c$? » L'espace de recherche est exploré sous la forme d'une arborescence binaire de racine G. Les fils d'un nœud sont respectivement obtenus en ajoutant au graphe associé à ce nœud (soit G') un sommet relation sous sa forme positive et sous sa forme négative (chacun des deux nouveaux graphes est donc obtenu par une étape de complétion à partir de G'). Après chaque étape de complétion, un test d'homomorphisme de F dans la complétion courante est effectué ; si le résultat est positif, cette

complétion est l'un des G_i recherchés, sinon l'exploration continue. La figure 2.9 donne un aperçu de cette méthode sur le cas très simple de l'exemple 9 ($\mathcal{V}_c = \{p\}$) . On crée deux

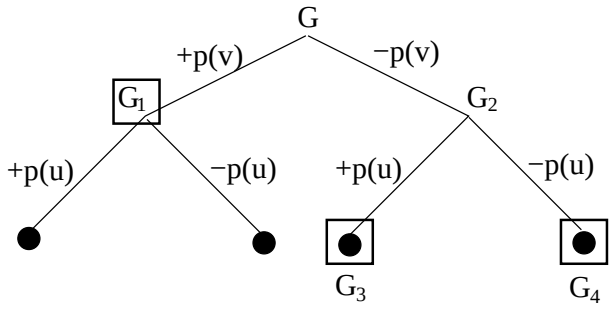


FIGURE 2.9 : exemple d'arbre de recherche de l'exemple 9. Chaque point noir représente un G^c et chaque carré un G_i.

nouveaux graphes G_1 et G_2, en ajoutant respectivement $+p(v)$ et $-p(v)$ à G. Il y a un homomorphisme de F dans G_1, on arrête donc de compléter G_1. Il n'y a pas d'homomorphisme de F dans G_2 : on crée deux nouveaux graphes G_3 et G_4, en ajoutant respectivement $+p(u)$ et $-p(u)$ à G_2. Il y a un homomorphisme de F dans G_3 et de F dans G_4. Finalement, l'ensemble prouvant que F se déduit de G est $\{G_1, G_3, G_4\}$ (alors qu'il existe quatre complétions totales de G par rapport à p).

Après une étape de filtrage, notamment basée sur les sous-graphes purs (s'il n'y a pas d'homomorphisme compatible d'un sous-graphe pur de F dans G, on conclut immédiatement que $G \not\vdash F$), l'algorithme finalement proposé est l'algorithme `recCheck` (cf. algorithme 1).

Algorithme 1: **recCheck**(G)

Données: un graphe polarisé consistant G
Accès: un graphe polarisé F, le vocabulaire de complétion $\mathcal{V}_c$ de F et G
Résultat: vrai si F se déduit de G, faux sinon
début
  si *il y a un homomorphisme de F dans G* **alors**
    retourner *vrai* ;
  si *G est complet par rapport à* $\mathcal{V}_c$ **alors**
    retourner *faux* ;
  Choisir $r \in \mathcal{V}$ et $t_1, \ldots, t_k$ dans G tel que $+r(t_1, \ldots, t_k) \notin G$ et $-r(t_1, \ldots, t_k) \notin G$;
  Soit G' obtenu de G en ajoutant $+r(t_1, \ldots, t_k)$;
  Soit G'' obtenu de G en ajoutant $-r(t_1, \ldots, t_k)$;
  retourner $recCheck(G')$ *ET* $recCheck(G'')$;
fin

CHAPITRE

3 Méthodologie expérimentale

Préambule

Afin de pouvoir comparer différentes heuristiques de l'algorithme `recCheck` *ainsi que différents algorithmes de résolution du problème* Déduction*, nous avons besoin de disposer de jeux de données et d'une méthodologie d'expérimentation. Nous décrivons dans ce chapitre la démarche expérimentale que nous avons menée : après avoir choisi les paramètres d'une instance du problème* Déduction*, nous proposons un algorithme de génération aléatoire de graphes polarisés (sans sommets termes isolés, équivalents à des formules de* $FOL\{\exists, \wedge, \neg_a\}$*) ; nous discutons ensuite des instances dites « difficiles », ainsi que du phénomène de seuil (aussi connu sous le nom de* transition de phase*) associé. Finalement, nous analysons l'influence de chaque paramètre sur la difficulté de résolution d'une instance et localisons des pics de difficulté « raisonnables », c'est-à-dire des valeurs de paramètres pour lesquelles les instances générées sont difficiles tout en restant généralement solubles en un temps « raisonnable » (ne dépassant pas une valeur prédéfinie). Ces valeurs de paramètres nous serviront dans la suite afin de tester expérimentalement nos propositions de raffinements et de comparer différents algorithmes (voir chapitre 4).*

Sommaire

3.1 Les jeux de données 36
3.2 Paramètres de génération 36
3.3 Algorithme de génération aléatoire 38
3.4 Instances difficiles et phénomène de seuil 39
3.5 Méthodologie de test 41
3.6 Influence des différents paramètres sur la difficulté 41

3.1 Les jeux de données

Afin de pouvoir mesurer les performances d'un mécanisme de résolution pour le comparer à d'autres, il est utile d'avoir accès à des jeux de tests standards appelés *benchmarks*. Dans un premier temps nous avons donc essayé de récupérer un benchmark pour le problème *Déduction*. Force est de constater que jusqu'à présent nous n'en avons identifié aucun adapté à ce problème. Nous nous sommes alors tournés vers la génération d'instances aléatoires, qui est notamment utilisée pour mesurer la difficulté théorique de résolution d'un problème, et assure la possibilité de construire une infinité d'instances. À notre connaissance, il n'existe actuellement aucun générateur aléatoire d'instances du problème *Déduction* (c'est-à-dire d'une paire de graphes polarisés ou de formules ou requêtes équivalentes). La priorité était donc de construire un « bon » générateur d'instances aléatoires. Par « bon » nous entendons que la génération ne doit pas être biaisée, c'est-à-dire que toutes les instances doivent avoir la même probabilité d'apparition ; on parle alors de génération équiprobable. Dans un premier temps, nous définissons les paramètres de génération d'un graphe polarisé (deux graphes polarisés formant une instance du problème *Déduction*) et proposons un algorithme de génération aléatoire dont nous pensons qu'il vérifie cette propriété (ceci restant toutefois à prouver).

3.2 Paramètres de génération

Les paramètres de génération choisis pour un graphe polarisé G sont les suivants :

- nbT : le nombre de sommets termes de G ;
- c : le pourcentage de sommets constantes dans G : c'est-à-dire le pourcentage de sommets constantes dans G par rapport à l'ensemble des sommets termes de G ;
- nbE : le nombre de relations différentes dans G ;
- k : l'arité de ces relations ;
- d : la densité par relation : c'est-à-dire, pour toute relation r, le rapport entre le nombre de sommets relations étiquetés $\sim r$ de G et le nombre de sommets relations étiquetés $\sim r$ ou $\overline{\sim r}$ d'un graphe complet G^c de G par rapport à r ;
- neg : le pourcentage de négation par relation : c'est-à-dire, pour toute relation r, le pourcentage de sommets relations négatifs étiquetés $-r$ dans G par rapport à l'ensemble des sommets relations étiquetés $\sim r$ de G.

Nous nous sommes posés la question de définir la densité et le pourcentage de négation de manière globale ou relative à chaque relation. Nous avons étudié les deux possibilités et nous nous sommes rendus compte que les résultats expérimentaux obtenus pour

des instances générées avec une densité globale (resp. un pourcentage de négation global) n'étaient pas reproductibles, c'est-à-dire que deux résultats expérimentaux basés sur les mêmes valeurs de paramètres pouvaient être totalement différents. Ceci est dû au fait qu'une densité globale (resp. un pourcentage de négation global) n'assure pas une répartition équitable dans le graphe. Ainsi pour une densité globale amenant la création d'un graphe $F_1^{exemple}$ composé de 12 sommets relations et construit à partir de 3 relation différentes (r, s et t), on peut par exemple avoir 2 sommets relations étiquetés r, 4 sommets relations étiquetés s et 6 sommets relations étiquetés t (cf. figure 3.1), ou toute autre distribution. De même pour le pourcentage de négation : prenons le graphe $F_1^{exemple}$ construit et un pourcentage de négation de 50%, il est possible que les 2 sommets relations étiquetés r et les 4 sommets relations étiquetés s soient négatifs et que les 6 sommets relations étiquetés t soient positifs. Comme nous l'avons vu au chapitre 2, le vocabulaire de complétion de deux graphes F et G est composé de toutes les relations apparaissant à la fois dans des sommets relations positifs et négatifs à la fois dans F et dans G. Dans $F_1^{exemple}$ il n'y a aucune relation satisfaisant cette condition : quelque soit le graphe G associé, le vocabulaire de $F_1^{exemple}$ et G sera vide, il n'y aura donc aucune complétion. Pour pallier ce problème, nous choisissons une densité et un pourcentage de négation *par* relation. En reprenant la construction du graphe, nous obtenons désormais le graphe $F_2^{exemple}$ composé d'exactement 4 sommets relations pour chaque relation avec 2 de ces sommets relations positifs et les 2 autres négatifs. Pour des mêmes valeurs de paramètres, nous assurons donc une forme de similarité entre les graphes générés, ce qui nous donne la possibilité de reproduire nos expérimentations.

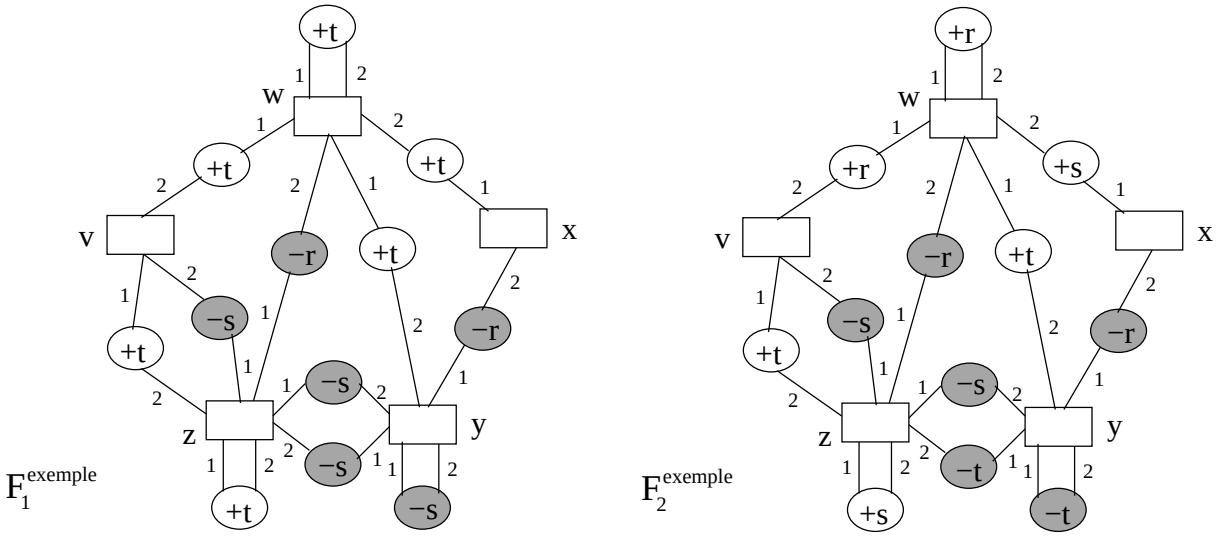


FIGURE 3.1 : deux graphes polarisés $F_1^{exemple}$ et $F_2^{exemple}$.

3.3 Algorithme de génération aléatoire

En nous basant sur les paramètres de génération choisis, nous proposons un algorithme de génération d'instances aléatoires du problème *Déduction*, que nous appelons `générerGPaléat` (cf. algorithme 2).

Algorithme 2: **générerGPaléat**(nbT,c,nbE,k,d,neg)

Données: le nombre de sommets termes nbT, le pourcentage de sommets constantes c, le nombre de relations différentes nbE, l'arité des relations k, la densité par relation d et le pourcentage de négation par relation neg.
Résultat: un graphe polarisé aléatoire G.
début
  Construire un graphe G à nbT sommets termes dont $nbT * c$ sommets constantes (sans sommets relations) ;
(1) **pour** $i = 1 \dots nbE$ **faire**
(2) **pour** $j = 1 \dots d * nbT^k$ **faire**
    $t_1, \dots, t_k \leftarrow$ choisir aléatoirement un k-tuple de termes de G tel que $+r_i(t_1, \dots, t_k) \notin G$ et $-r_i(t_1, \dots, t_k) \notin G$;
    Ajouter $+r_i(t_1, \dots, t_k)$ à G ;
(3) **pour** $i = 1 \dots neg * d * nbT^k$ **faire**
    Choisir aléatoirement dans G un sommet relation $+r_i(t_1, \dots, t_k)$;
    Inverser le signe de $+r_i(t_1, \dots, t_k)$;
  retourner G ;
fin

L'algorithme `générerGPaléat` construit un graphe polarisé aléatoire standard de la manière suivante : il génère un graphe G à nbT sommets termes dont $nbT * c$ sommets termes sont des sommets constantes. Pour chaque relation r_i (avec $i = 1 \dots nbE$), il ajoute *aléatoirement* $d * nbT^k$ sommets relations étiquetés $+r$ à G et inverse *aléatoirement* le signe de $d * nbT^k * neg$ d'entre eux. Il termine car le nombre de relations différentes est fini. Il est en $\mathcal{O}(2 * nbT^k * |nbE|)$: il y aura $|nbE|$ tours de la première boucle (1), nbT^k tours de la seconde boucle (2) et au plus nbT^k tours de la troisième boucle (3).

Notons que lors du tirage aléatoire d'un k-tuple de G, nous assurons que le k-tuple choisi sera différent. Soit *listeTuples* la liste des k-tuples de G. Lors du $n^{ième}$ tirage aléatoire, on choisit un nombre N tel que $0 \leq N \leq |listeTuples| - n$. On inscrit le nombre choisi dans une liste de nombres déjà vus qui est triée par ordre croissant de la manière suivante : on parcourt la liste des nombres déjà vus et on incrémente N de 1 pour chaque nombre inférieur ou égal à N. Si à l'indice i on trouve un nombre supérieur à N, on s'arrête et on insère N à l'indice i, sinon on insère N à la fin de la liste. Le k-tuple $t_1, \dots, t_k$ choisi sera celui se trouvant à l'indice N de la liste des k-tuples de G (notons qu'il vérifiera automati-

quement la condition « $+r_i(t_1,\ldots,t_k) \notin G$ et $-r_i(t_1,\ldots,t_k) \notin G$ » puisqu'il n'aura jamais été choisi auparavant). Ce procédé intuitif permet d'assurer que l'on ne visitera pas plusieurs fois le même k-tuple. Nous faisons de même pour le choix d'un sommet relation dans G.

Dans le cas où l'on souhaite se restreindre à des graphes polarisés possédant une ou plusieurs caractéristiques particulières (par exemple sans sommet terme isolé, connexe, safe...), nous relançons l'algorithme jusqu'à ce que la ou les propriétés soient satisfaites pour le graphe généré. Ainsi nous n'introduisons aucun biais lors de la génération aléatoire.

Notons finalement que le nombre de sommets relations d'un graphe F est donné par la formule suivante : $nbSomR = d * nbT_F^k * nbE$.

3.4 Instances difficiles et phénomène de seuil

Le principe général des expérimentations sur données aléatoires consiste à générer un jeu de test en fixant certains paramètres et en en faisant évoluer un (ou plusieurs) autre(s). Pour chacune des valeurs du paramètre variable, on génère un ensemble d'instances et on lance la résolution de ces instances. On effectue alors des mesures sur les résultats obtenus que l'on présente sur des courbes prenant en abscisse le paramètre variable et en ordonnée la mesure calculée. Pour obtenir une instance du problème `Déduction`, nous générons deux graphes polarisés aléatoires avec l'algorithme `générerGPaléat`, respectivement le source, F, et le cible, G, et cherchons à savoir si F se déduit de G. Nous appelons *nbTS* (resp. *DS*) et *nbTC* (resp. *DC*) le nombre de sommets termes (resp. la densité par relation) respectivement du source et du cible. Les autres paramètres seront identiques pour le source et le cible pour les raisons suivantes : si une constante (resp. une relation, un sommet relation positif ou négatif) apparaît dans le source mais pas dans le cible alors il n'y aura trivialement pas déduction, de même si l'arité des sommets relations du source et du cible est différente ; si une constante (resp. une relation, un sommet relation positif ou négatif) apparaît dans le cible mais pas dans le source alors elle ne joue aucun rôle lors de la déduction et peut être supprimé du cible sans incidence sur la résolution du problème.

Une notion importante est celle de la mesure de la difficulté des instances du problème à résoudre. En effet, si l'on souhaite comparer différentes techniques de résolution, il semble préférable de le faire sur des instances « difficiles » du problème plus à même de discriminer les heuristiques. Intuitivement une instance est dite *difficile* si sa résolution nécessite un temps important quelque soit la méthode utilisée. Si de nombreux travaux d'identification d'un tel critère existent sur les problèmes *NP-complets* autour de la notion de transition de phase ([Sais, 2008] par exemple), très peu à notre connaissance ont été réalisés sur les problèmes Π_2^P-$complets$. La notion de transition de phase est utilisée dans différents domaines. Généralement, elle représente une transformation du système étudié provoquée par la variation d'un paramètre extérieur particulier. Cette transition a lieu lorsque le paramètre atteint une valeur seuil. Par exemple en physique, pour un métal

ferromagnétique (aimant...) une transition de phase apparaît à une température critique, le point de Curie. Au-dessus de cette température, le métal ferromagnétique n'a aucune magnétisation. Si le métal ferromagnétique est refroidi, il devient brusquement magnétisé à partir de la température de Curie. On appelle ce phénomène une transition de phase (passage de l'état paramagnétique à l'état ferromagnétique à une valeur seuil de température). Dans notre cas le système étudié est la déduction ; la figure 3.2 donne un aperçu d'une transition de phase de la phase déduction à la phase non-déduction.

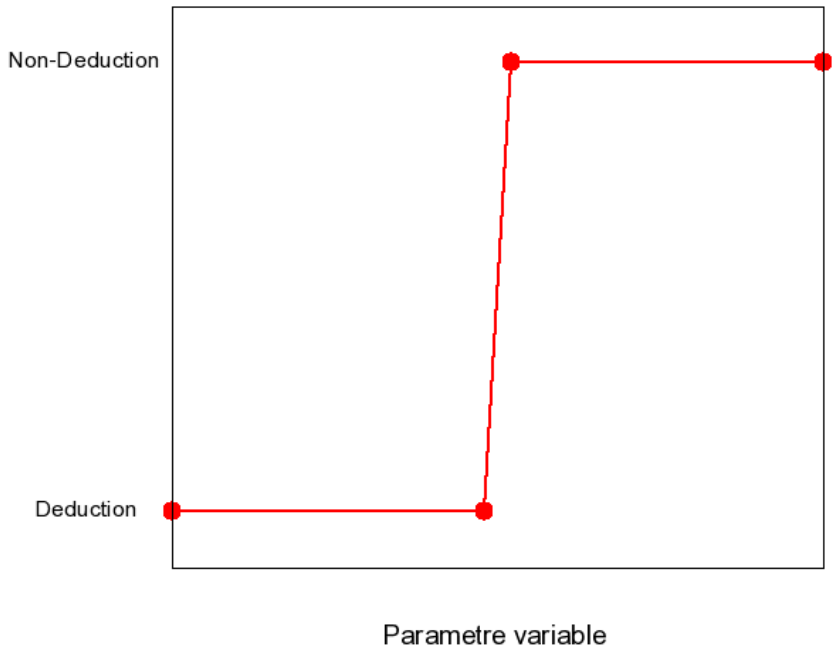


FIGURE 3.2 : illustration d'une transition de phase de la phase déduction à la phase non-déduction.

L'existence d'une transition de phase est prouvée théoriquement et observée expérimentalement pour divers problèmes *NP-complets* tels SAT, CSP... Par exemple, pour le problème *3-SAT* il existe une valeur critique pour un paramètre donné en-dessous de laquelle la quasi-totalité des formules sont satisfiables et au-delà de laquelle la quasi-totalité des formules sont insatisfiables [Sais, 2008]. Ce brusque changement est une transition de la phase *SAT* à la phase *UNSAT*.

À notre connaissance, il n'existe aucun résultat théorique sur l'existence d'une transition de phase pour un problème Π_2^P-$complet$. Néanmoins à la fin des années 90, Gent et Walsh [1999] prouvent de manière empirique que le comportement de la transition de phase pour le problème $QSAT$ se rapproche de celui constaté pour des problèmes *NP-complets* tels que SAT ([Sais, 2008] par exemple) ou CSP ([Tsang, 1993] par exemple).

Ils conjecturent qu'un comportement similaire apparaîtra également pour des problèmes *PSPACE-complets* tels que la planification de tâches ou les jeux combinatoires.

3.5 Méthodologie de test

Pour essayer de caractériser l'influence des paramètres sur la difficulté du problème *Déduction* et la transition de phase, nous avons dans un premier temps procédé par essais successifs avec l'algorithme `recCheck` raffiné (voir le chapitre 4 pour les raffinements), en fixant un timeout à 5 minutes. Pour chaque expérimentation, les mesures effectuées sont les suivantes :

- le temps de calcul de la résolution (en millisecondes) ;
- la taille de l'arbre de recherche exploré ;
- le nombre de tests d'homomorphisme ;
- le pourcentage de déduction ;
- le pourcentage de timeouts.

Pour chaque valeur du paramètre variable, nous exécutons 1000 instances du problème *Déduction* et retenons la moyenne de l'ensemble des résultats obtenus. Notons qu'en cas de timeout, les résultats (temps de calcul, taille de l'arbre de recherche exploré...) de l'instance l'ayant provoqué sont ceux obtenus au moment du timeout (seul le champ correspondant à la déduction reste vide et l'instance ne sera pas utilisée pour calculer le pourcentage de déduction). Ainsi moins il y aura de timeouts plus les résultats seront précis.

Finalement, nos expérimentations ont été réalisées sous environnement Linux, sur un serveur Sun Fire X4100 AMD Opteron 252, équipé d'un processeur 2.6 GHz Dual-Core et de 4Go de mémoire vive.

3.6 Influence des différents paramètres sur la difficulté

La difficulté d'une instance dépend des valeurs des paramètres choisies pour la génération des graphes. Dans cette partie, nous déterminons empiriquement l'influence des paramètres sur la difficulté de résolution. Notons que, pour une instance I, plus l'espace de recherche exploré afin de résoudre I est grand plus le nombre d'homomorphismes calculés est grand (puisqu'au moins un test d'homomorphisme est effectué à chaque nœud de l'espace de recherche), et donc plus l'instance sera difficile. Intuitivement, on s'attend à ce qu'en moyenne plus la taille de l'espace de recherche menant de G à ses complétions totales est grande (c'est-à-dire plus le nombre de complétions totales de G est grand), plus l'arbre de recherche exploré a de chances d'être grand. Rappelons qu'un graphe G a $2^{(n_G)^k \times |\mathcal{V}_G|}$ complétions totales possibles, ce qui est équivalent avec nos paramètres à $2^{(nbT_G)^k \times nbE}$ (en supposant que la taille du vocabulaire de complétion associé soit égale à nbE). Ainsi en augmentant la valeur des paramètres nbT_G, k et nbE, le nombre de com-

plétions totales possibles va croître (de façon exponentielle), ce qui nous laisse supposer que la difficulté de l'instance augmentera également.

D'un autre côté, les paramètres *c* et *neg* semblent indépendants des autres paramètres : pour ce qui est du pourcentage de négation, on s'attend à ce que les instances les plus difficiles (pour toute valeur des autres paramètres) soient caractérisées par une valeur de 50%. En effet dans le cas contraire, par exemple supposons une difficulté maximum pour *neg* = 40%, il suffirait d'inverser les signes des sommets relations de F et G afin de se retrouver avec *neg* = 60% qui serait par hypothèse plus facile à résoudre (cf. figure 3.3, nous obtenons F_2 après inversion des signes des sommets relations de F_1).

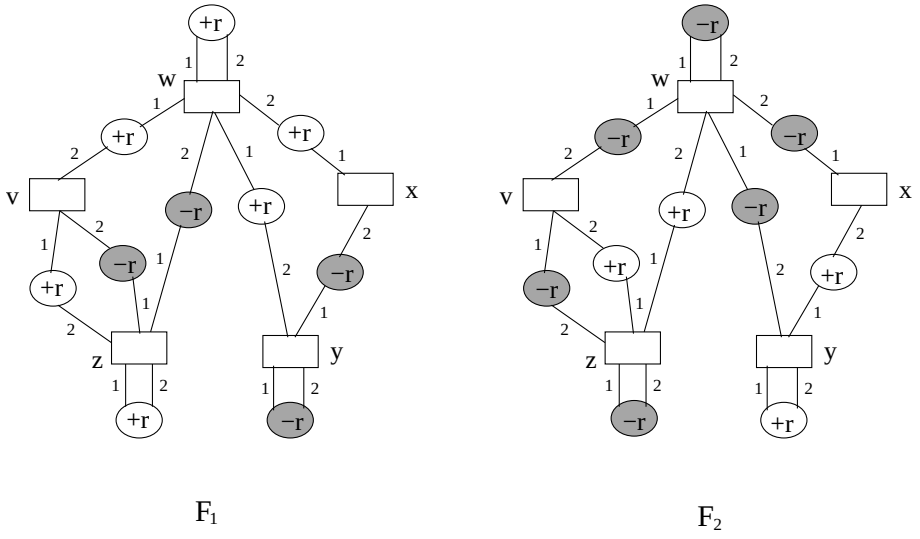


FIGURE 3.3 : illustration de l'inversion des signes d'un graphe.

Pour le pourcentage de sommets constantes, on s'attend à ce que les instances les plus difficiles soient caractérisées par une valeur de 0%, car l'ajout de sommets constantes dans le graphe source influence directement la résolution de l'instance : un sommet constante étiqueté *a* dans le graphe source sera automatiquement « fixé » au sommet constante étiqueté *a* dans le graphe cible s'il existe, sinon on pourra directement conclure qu'il n'y a pas déduction.

Finalement, on s'attend à ce que le choix des valeurs caractérisant les instances difficiles pour les paramètres de densité du source et de densité du cible soit dépendant des valeurs des autres paramètres. Prenons par exemple la densité du graphe cible : selon le nombre de sommets termes du source et du cible, le nombre de relations différentes et la densité du graphe source par exemple, la valeur de la densité du cible pour laquelle les instances ne seront pas triviales sera totalement différente. Considérons par exemple un graphe source F composé de 5 sommets variables et de 6 sommets relations de relation r d'arité 2 dont 3 positifs et 3 négatifs, soit une densité de 24%. Supposons que nous ayons

un graphe cible G composé de 5 sommets variables et un autre graphe cible G' composé de 25 sommets variables, avec pour les deux les valeurs suivantes : $neg = 50\%$, $nbE = 1$ et $k = 2$. Si nous choisissons une densité cible de 32%, le graphe G sera composé de 8 sommets relations de relation r d'arité 2 (4 positifs et 4 négatifs) alors que le graphe G' sera composé de 200 sommets relations de relation r d'arité 2 (100 positifs et 100 négatifs). Le problème $G \vdash F$ sera difficilement solvable alors que le problème $G' \vdash F$ sera trivialement solvable (il y aura généralement un homomorphisme de F dans G').

Dans la suite, nous commençons par caractériser l'influence des densités des graphes source et cible, puis nous caractérisons l'influence des paramètres dits « indépendants », soit c et neg, et enfin l'influence des paramètres nbT_G, k et nbE.

3.6.1 Influence des densités des graphes source et cible

Nous avons commencé par déterminer l'influence des densités des graphes source et cible sur la difficulté du problème car ces paramètres sont primordiaux dans le processus de résolution. En effet, si nous fixons tous les autres paramètres pour F et G et que nous considérons un graphe source très dense et un graphe cible peu dense, l'instance est *triviale*, nous concluons directement à un échec (nous pouvons par exemple utiliser la propriété 2.2 sur les sous-graphes purs : comme dans ce cas un sous-graphe pur F' de F est grand, il n'y a généralement pas d'homomorphisme compatible de F' dans G). De même si nous considérons un graphe source peu dense et un graphe cible très dense, nous pouvons conclure à un succès sans compléter G (il y a généralement un homomorphisme de F dans G). Nous avons alors étudié l'influence des densités des graphes source et cible sur la difficulté en fixant $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 1$, $k = 2$ et $neg = 50\%$. Dans la suite, pour $neg = 50\%$, nous considérons toujours les densités pour lesquelles le nombre de sommets relations par relation est pair, afin d'avoir le même nombre de sommets relations positifs et négatifs par relation. Les figures 3.4, 3.5 et 3.6 illustrent les résultats obtenus, respectivement en mesurant le temps de calcul, la taille de l'arbre de recherche exploré et le pourcentage de déduction pour les valeurs de densité du source $DS = 9.375\%$, 12.5%, 15.625%, 18.75%, 21.875%, 25% et 28.125%, et en faisant varier en abscisse la densité du graphe cible. Notons que le pourcentage de timeouts est proche de 0% pour toute valeur de DS et DC.

Intuitivement, on s'attend à ce que le temps de résolution soit directement lié à la taille de l'arbre de recherche exploré puisqu'au moins un test d'homomorphisme est effectué à chaque nœud de l'arbre de recherche. Cette intuition est confirmée par les figures 3.4 et 3.5 : les zones de difficulté pour les mesures du temps et de la taille de l'arbre de recherche sont corrélées (par exemple pour $DS = 18.75\%$, les trois pics de difficulté - $DC = 31.25\%$, 34.375% et 37.5% - se retrouvent sur les deux figures). Par ailleurs la difficulté est observée maximale pour $DS = 15.625\%$. Néanmoins quelques valeurs ne sont pas directement corrélées (par exemple pour $DS = 15.625\%$, le temps de résolution pour $DC = 18.75\%$ est inférieur à celui pour $DC = 21.875\%$ - respectivement 2189ms et 2531ms - alors que la taille

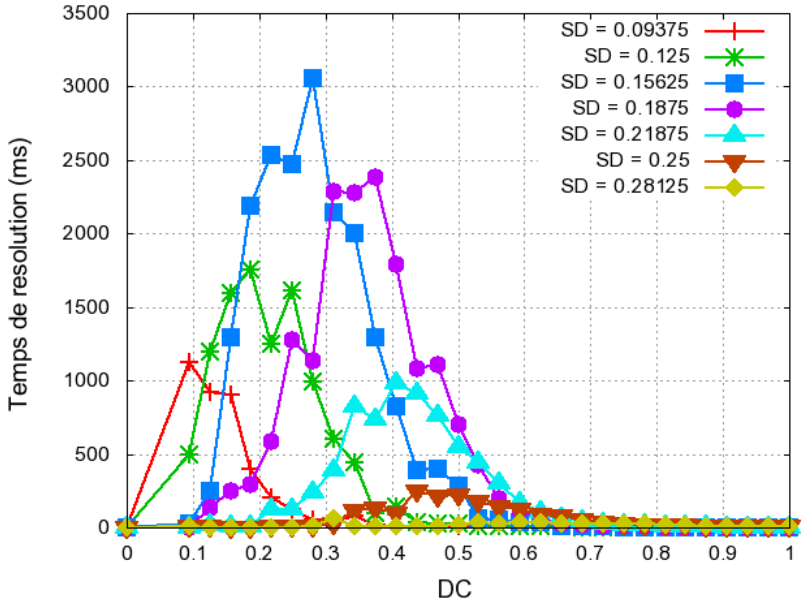


FIGURE 3.4 : influence des densités sur le temps de calcul.

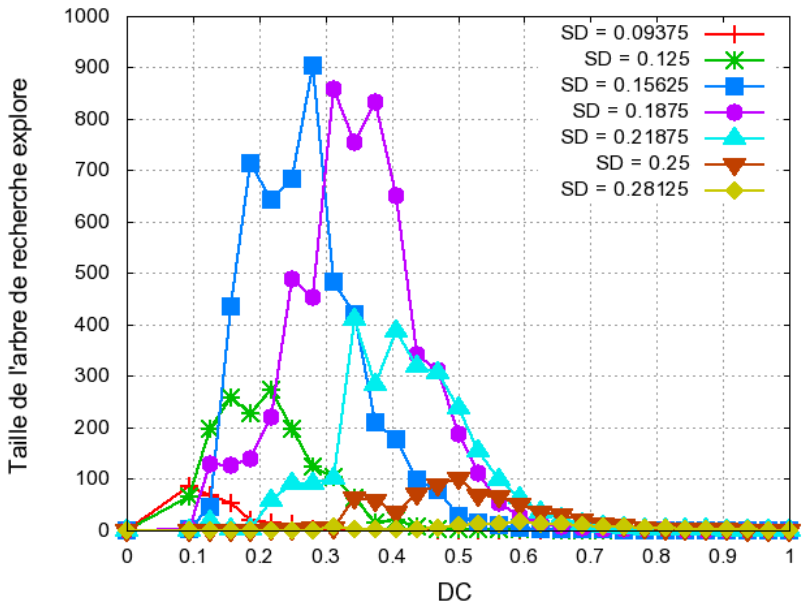


FIGURE 3.5 : influence des densités sur la taille de l'arbre de recherche.

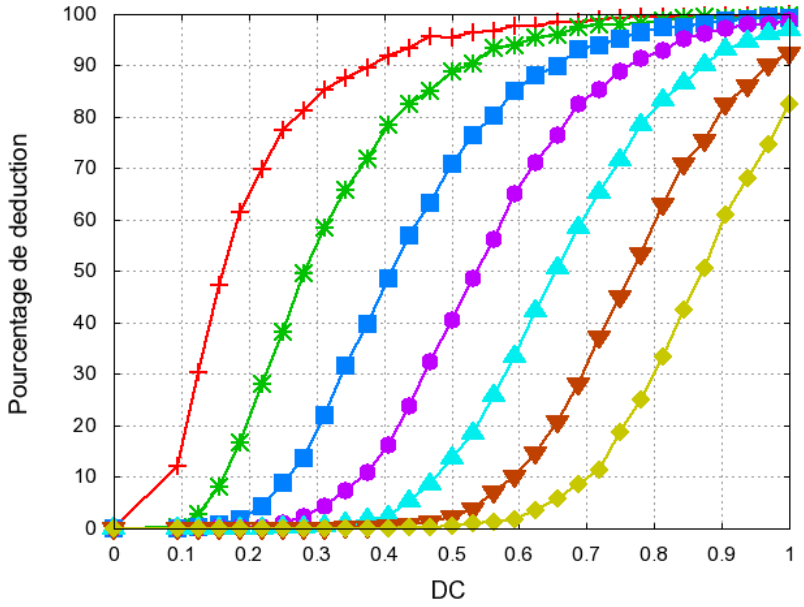


FIGURE 3.6 : influence des densités sur le pourcentage de déduction.

de l'arbre de recherche pour DC = 18.75% est supérieure à celle pour DC = 21.875% - respectivement 714 et 643 nœuds). Cela est dû d'une part au fait que le temps d'exécution du test d'homomorphisme est variable d'une instance à l'autre (car il prend en compte la structure des graphes et elle peut être très différente d'une instance à l'autre) et d'autre part à la présence de quelques instances très difficiles. Ce phénomène d'instances exceptionnellement difficiles a déjà été montré pour des problèmes *NP-complets* (voir [Hogg et Williams, 1994] et [Smith et Grant, 1994] par exemple).

Nous observons également que pour une valeur v^{DS} de DS, les instances les plus difficiles apparaissent généralement pour une valeur v^{DC} de DC telle que $v^{DC} \simeq 1.8 * v^{DS}$, soit pour un graphe cible composé d'environ 1.8 fois plus de sommets relations que le source. Cette distribution peut s'expliquer par le fait qu'elle permet d'obtenir des instances qui vont nécessiter un grand nombre de complétions explorées ; ces instances sont donc généralement non triviales.

On s'aperçoit sur la figure 3.6 que pour une valeur de DS, il existe un certain intervalle de valeurs de DC tel qu'au-dessous de cet intervalle la quasi-totalité des instances ne présentent pas de déduction alors qu'au-delà de cet intervalle la quasi-totalité des instances présentent une déduction ; cet intervalle représente une transition de phase de la phase non-déductible à la phase déductible. Pour le problème *NP-complet SAT* par exemple, il a été montré que les instances les plus difficiles étaient corrélées à un pourcentage de satisfiabilité d'environ 50%. Dans notre cas, nous observons que les instances les plus coûteuses à résoudre (en temps et en espace, cf. les figures 3.4 et 3.5) se trouvent toutes au

début de la transition de phase, soit pour un pourcentage de déduction quasiment toujours inférieur à 20%. La figure 3.7 montre ce phénomène pour $DS = 15.625\%$, en mesurant respectivement le temps de calcul (courbe rouge) et le pourcentage de déduction (courbe verte). La valeur de DC amenant le plus haut pic de difficulté est $DC = 28.125\%$ (cf. flèche (1)) et la valeur du pourcentage de déduction pour $DC = 28.125\%$ est d'environ 14% (cf. flèche (2)). Ce phénomène se répète pour chaque valeur de DS et sa caractérisation reste un problème ouvert.

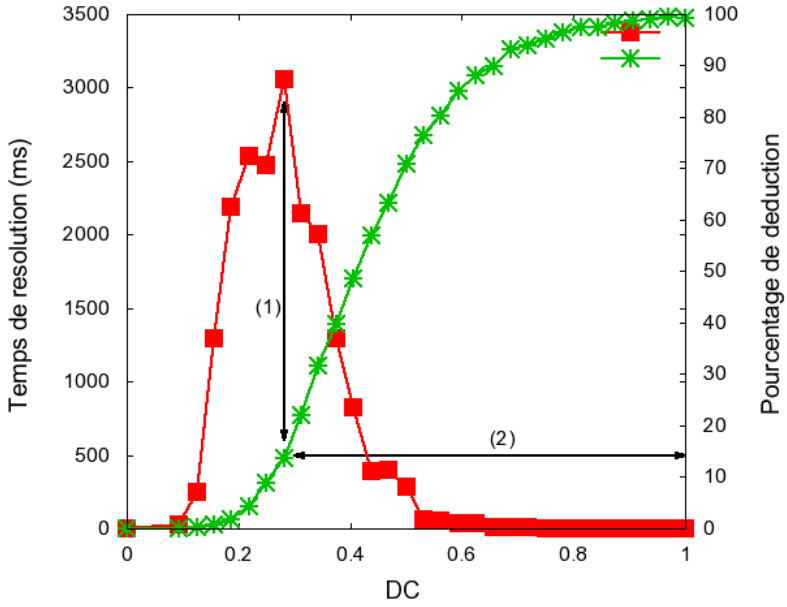


FIGURE 3.7 : influence des densités sur le pourcentage de déduction.

3.6.2 Influence du pourcentage de négation

Les figures 3.8 et 3.9 illustrent les résultats obtenus pour $nbTS = 5$, $nbTC = 10$, $k = 2$, $nbE = 1$ et $DS = 40\%$, soit pour un graphe source composé d'exactement 10 sommets relations, respectivement en mesurant le temps de calcul et la taille de l'arbre de recherche exploré. Chaque courbe représente le résultat obtenu pour une valeur du pourcentage de négation, soit $neg = 10$, 20, 30, 40 ou 50% (notons que par symétrie nous obtenons directement les valeurs 60%, 70%...). Nous faisons varier en abscisse la densité du graphe cible et nous montrons que pour la quasi-totalité des valeurs de DC, plus on se rapproche d'un pourcentage de négation de 50%, plus la résolution de l'instance est difficile. Les figures 3.10 et 3.11 montrent les résultats sous une autre forme, la courbe étant obtenue en faisant varier neg en abscisse et en choisissant pour chaque point la valeur de DC amenant le plus haut pic de difficulté (voir la table 3.1 pour plus de précisions), respectivement en mesurant le

Pourcentage de négation	10	20	30	40	50	60	70	80	90
DC (%)	34	32	44	46	50	46	44	34	34

TABLE 3.1 : valeur de *DC* choisie pour chaque valeur du pourcentage de négation.

temps de calcul et la taille de l'arbre de recherche exploré. Dans la suite nous garderons cette représentation qui est plus lisible.

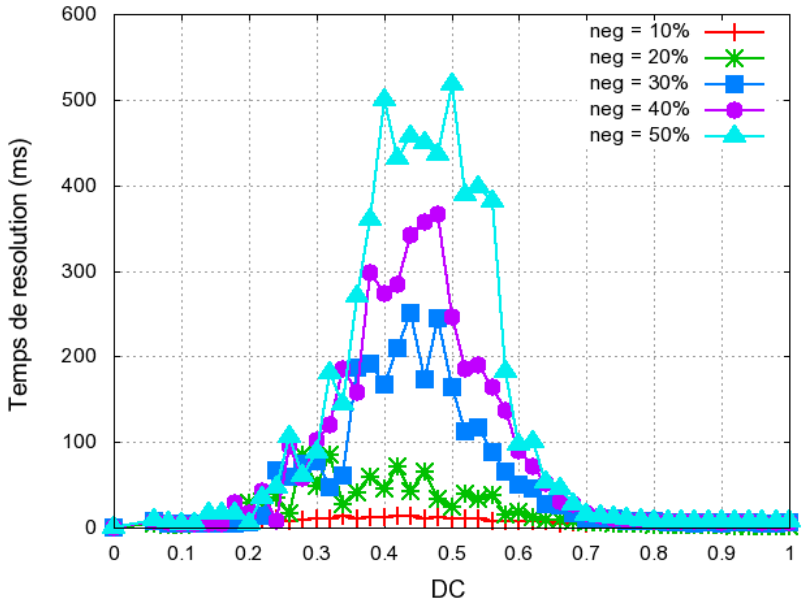


FIGURE 3.8 : influence du pourcentage de négation sur le temps de calcul.

Les figures 3.10 et 3.11 confirment notre intuition : les instances les plus difficiles sont obtenues pour un pourcentage de négation de 50% ; au plus haut pic de difficulté elles sont en moyenne résolus en 517ms pour un arbre de recherche de taille 569.

Dans la suite, nous choisirons toujours $neg = 50\%$.

3.6.3 Influence du pourcentage de sommets constantes

Intuitivement, on s'attend à ce que les instances les plus difficiles soient caractérisées par un pourcentage de sommets constantes de 0%, car les sommets constantes jouent un rôle particulier : puisqu'un sommet constante t étiqueté A dans F' apparaît une seule fois dans G (soit t' étiqueté A), il limite les images possibles de t au seul sommet t'. Ainsi les sommets constantes contraignent les images des sommets termes de F et augmentent la probabilité de trivialité de l'instance (il n'y aura plus aussi facilement d'homomorphisme compatible d'un sous-graphe pur de F dans G par exemple).

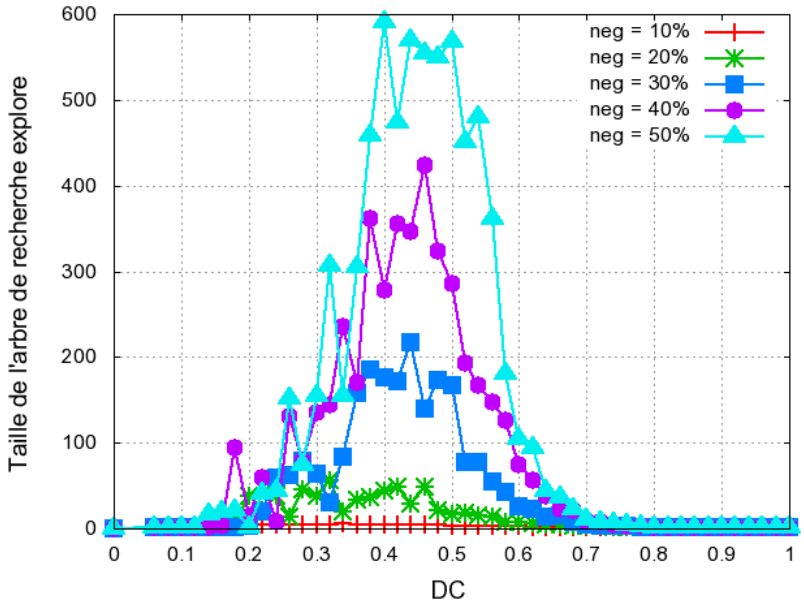


FIGURE 3.9 : influence du pourcentage de négation sur la taille de l'arbre de recherche exploré.

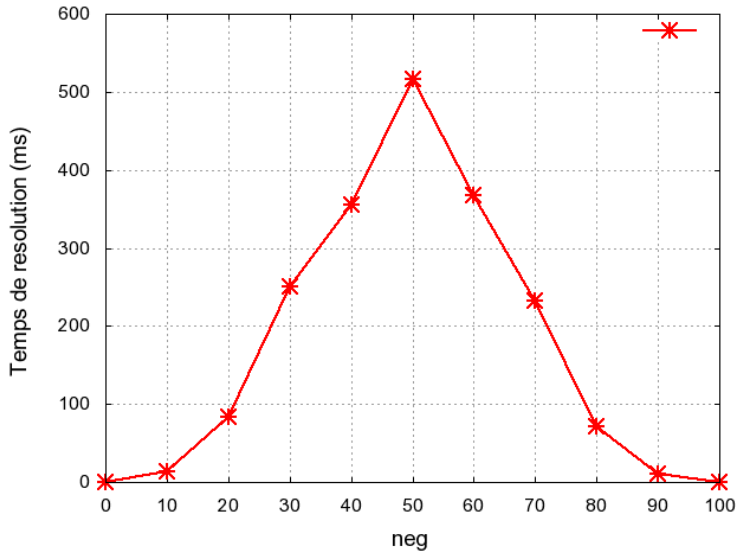


FIGURE 3.10 : influence du pourcentage de négation sur la difficulté de résolution (temps de calcul).

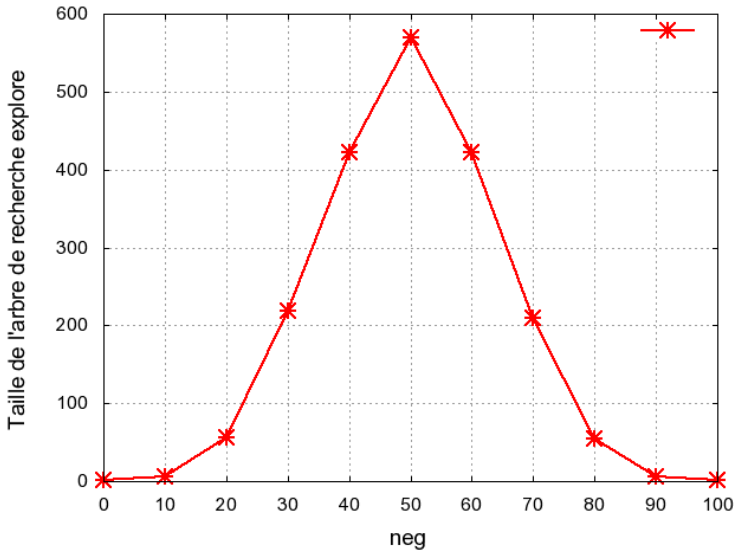


FIGURE 3.11 : influence du pourcentage de négation sur la difficulté de résolution (taille de l'arbre de recherche exploré).

Les figures 3.12 et 3.13 montrent les résultats obtenus pour $nbTS = 10$, $nbTC = 10$, $k = 2$, $nbE = 1$ et $DS = 8\%$, soit pour un graphe source composé d'exactement 10 sommets relations, respectivement en mesurant le temps de calcul et la taille de l'arbre de recherche exploré. Chaque courbe représente le résultat obtenu en faisant varier c en abscisse, soit $c = 0$, 20, 40, 60, 80 ou 100%, et en choisissant pour chaque point la valeur de DC amenant le plus haut pic de difficulté, respectivement en mesurant le temps de calcul et la taille de l'arbre de recherche exploré.

Notre intuition est à nouveau confirmée : plus le pourcentage de sommets constantes est petit, plus la résolution de l'instance est difficile. Plus précisément, les instances les plus difficiles sont obtenues pour un pourcentage de sommets constantes de 0% ; elles sont en moyenne résolus en 13568ms pour un arbre de recherche de taille 106. L'importante différence entre cette expérimentation est la précédente vient du nombre de sommets termes des graphes source et cible. Ici le nombre de sommets termes du source et du cible est plus important, ce qui rend le test d'homomorphisme beaucoup plus complexe (chaque sommet variable du source a $nbTC$ images possibles dans le cible - c'est-à-dire tous les sommets termes du cible). De plus, nous constatons que la difficulté décroît rapidement dès que nous ajoutons des sommets constantes.

Dans la suite, nous choisirons $c = 0\%$ pour nos tests.

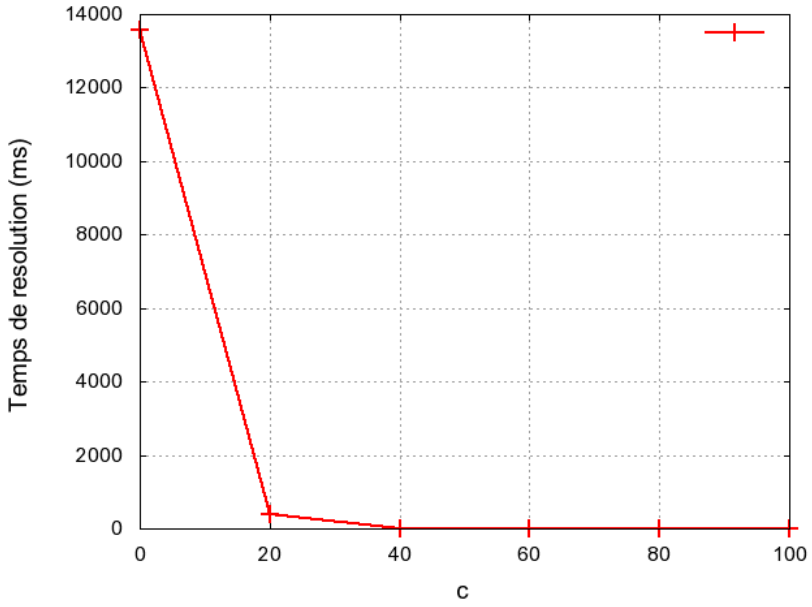


FIGURE 3.12 : influence du pourcentage de sommets constantes sur le temps de calcul.

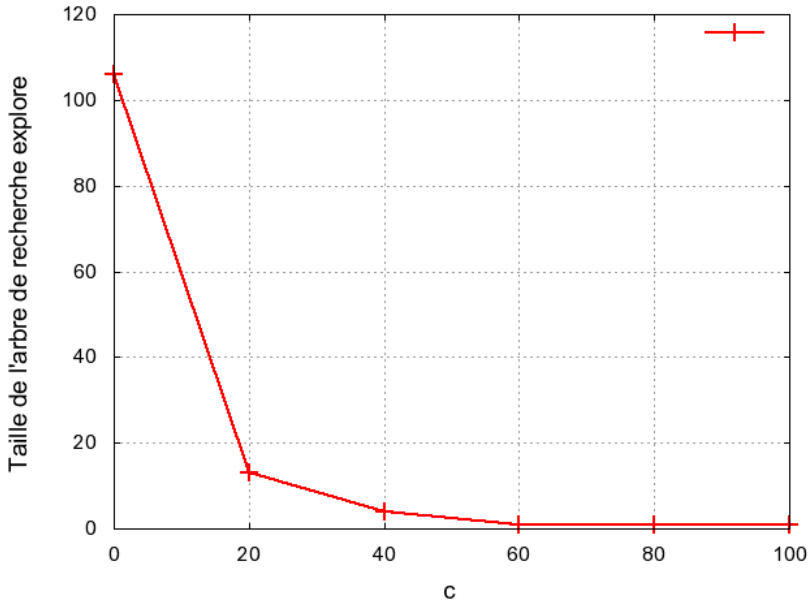


FIGURE 3.13 : influence du pourcentage de sommets constantes sur la taille de l'arbre de recherche exploré.

NbTC	5	10	15	20	25
DC (%)	32	20	14	13	11

TABLE 3.2 : valeur de *DC* choisie pour chaque valeur du nombre de sommets termes du graphe cible.

3.6.4 Influence du nombre de sommets termes du graphe cible

Le nombre de sommets termes du graphe cible G étant un facteur du nombre de complétions totales de G, on s'attend à ce que son augmentation accroisse la difficulté de résolution des instances. Les figures 3.14 et 3.15 confirment notre intuition : les résultats sont obtenus pour $nbTS = 5$, $k = 2$, $nbE = 1$, $DS = 24\%$ (soit un graphe source composé de 6 sommets relations) et $DC = x$, où x correspond (pour chaque point) à la valeur de DC telle que la difficulté de résolution de l'instance soit maximale (voir table 3.2 pour plus de détails), respectivement en mesurant le temps de calcul, la taille de l'arbre de recherche exploré et le pourcentage de timeouts en faisant varier en abscisse le nombre de sommets termes du graphe cible. On voit que l'augmentation du nombre de sommets termes du graphe cible est corrélée avec un accroissement de la taille de l'arbre de recherche exploré et du temps d'exécution.

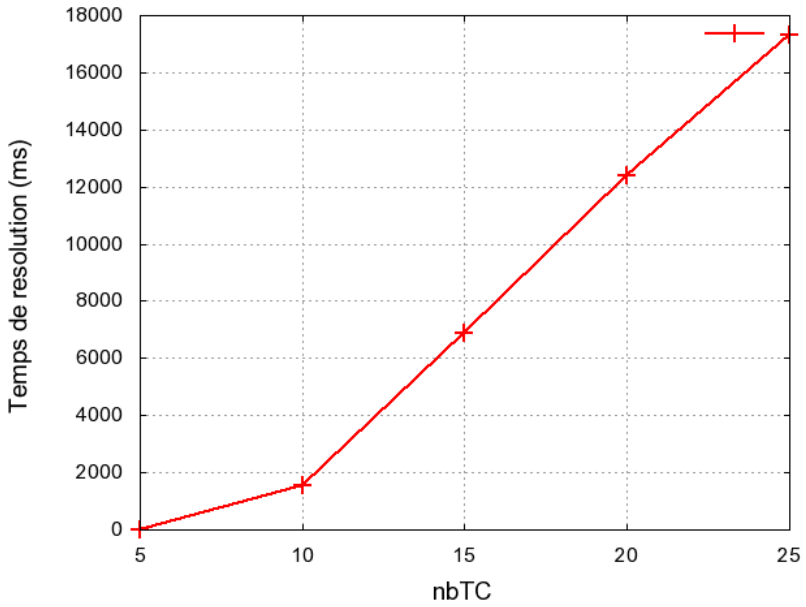


FIGURE 3.14 : influence du nombre de sommets termes du graphe cible sur le temps de calcul.

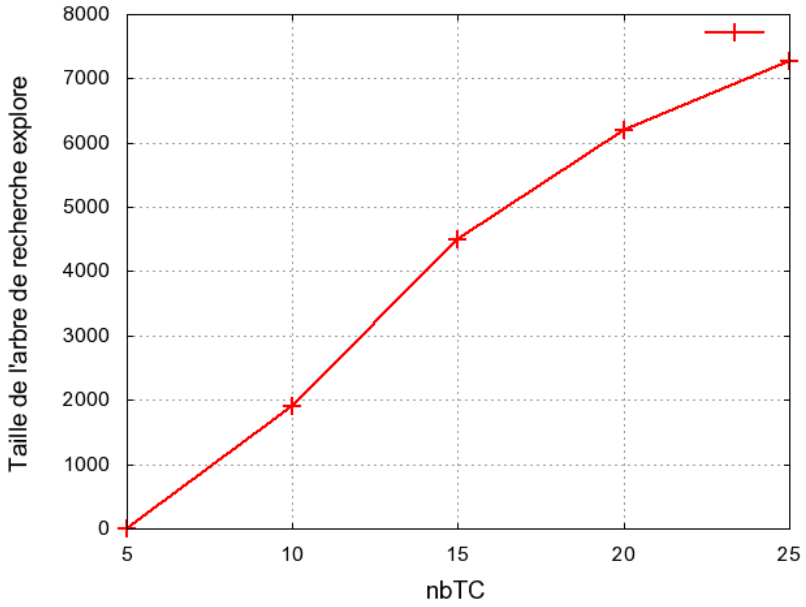


FIGURE 3.15 : influence du nombre de sommets termes du graphe cible sur la taille de l'arbre de recherche exploré.

La figure 3.16 montre les résultats obtenus en mesurant le pourcentage de timeouts. On s'aperçoit que ce pourcentage augmente très rapidement, avec plus de 10% de timeouts pour $nbTC = 15$. Comme nous l'avons précisé en introduction de cette section, moins il y a de timeouts et plus les résultats sont précis. Ainsi dans la suite, nous nous intéresserons à des valeurs de paramètres tels que le pourcentage de timeouts soit inférieur à 1% : nous choisirons $nbTS = nbTC$.

3.6.5 Influence de l'arité des relations

Comme le nombre de sommets termes du graphe cible, l'arité des relations est un facteur du nombre de complétions totales de G. On s'attend donc à ce que son augmentation accroisse la difficulté de résolution des instances. Notons que la densité du graphe considéré dépend de l'arité des relations. En effet, prenons deux graphes F_1 et F_2 composés de 8 sommets termes et respectivement de sommets relations d'arité 2 et 3 (avec $nbE = 1$). Si nous choisissons pour F_1 et F_2 une densité égale à 0.1875, F_1 sera composé de 12 sommets relations d'arité 2 ($d * nbT^k_{F_1} * nbE$ soit $0.1875 * 8^2 * 1$) alors que F_2 sera composé de 96 sommets relations d'arité 3 ($0.1875 * 8^3 * 1$). Pour que F_2 soit composé de 12 sommets relations, il faudrait qu'il ait une densité de 2.34375% (avec la formule $d = \frac{nbSomR}{nbT^k_{F_2} * nbE}$, soit $d = \frac{12}{8^3 * 1}$). Néanmoins nous nous sommes aperçus que le nombre de sommets relations n'était pas

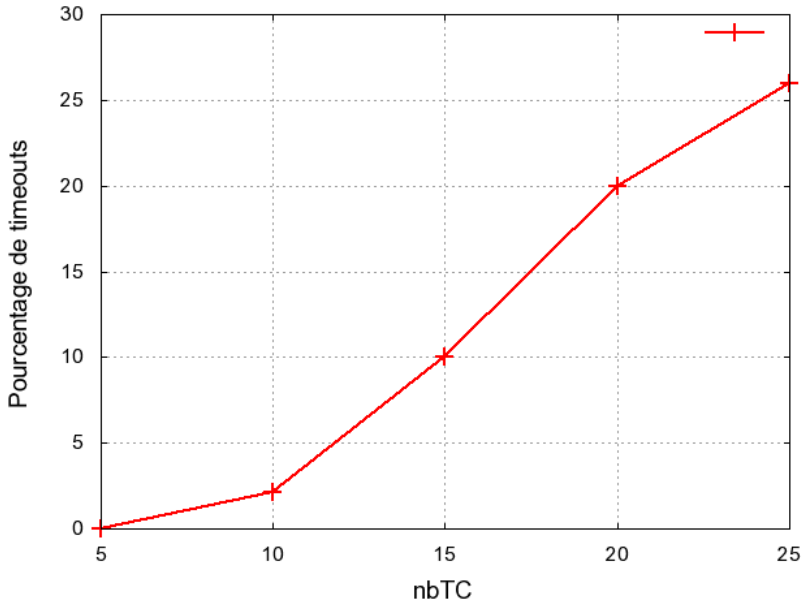


FIGURE 3.16 : influence du nombre de sommets termes du graphe cible sur le pourcentage de timeouts.

la bonne notion pour caractériser une densité commune entre les différentes valeurs de k. En effet si un graphe F est composé de n sommets relations d'arité k, le nombre d'arêtes dans F est égal à $n \times k$. Notons que chaque arête amène une contrainte supplémentaire dans le test d'homomorphisme, donc plus le graphe source a d'arêtes plus il contraint la recherche d'homomorphisme. Ainsi si nous avons choisi une densité de 18.75% pour F_1 (il sera alors composé de 12 sommets relations soit $12 \times 2 = 24$ arêtes), nous choisirons une densité de 15.625% pour F_2 (il sera alors composé de 8 sommets relations soit $8 \times 3 = 24$ arêtes) et ainsi de suite.

Les figures 3.17 et 3.18 confirment notre intuition : les résultats sont obtenus pour $nbTS = 8$, $nbTC = 8$, $nbE = 1$, DS = x et DC = y, où x correspond (pour chaque point) à la valeur de DS telle que le graphe source est composé de 24 arêtes et y correspond à la valeur de DC telle que la difficulté de résolution de l'instance soit maximale, respectivement en mesurant le temps de calcul, la taille de l'arbre de recherche exploré et le pourcentage de timeouts en faisant varier en abscisse l'arité des relations. Notons que pour cette expérimentation nous avons baissé le timeout à 1 minute car le temps d'exécution était trop long.

Nous apercevons qu'il y a un grand écart entre $k = 2$ (résolution en 2381ms - dont 0.26% de timeouts - pour un arbre de recherche de taille 833) et $k = 3$ (résolution en 36533ms - dont 60% de timeouts - pour un arbre de recherche de taille 9011). Ceci s'explique par l'important écart entre le nombre de complétions totales possibles dans chaque cas (pour $k =$

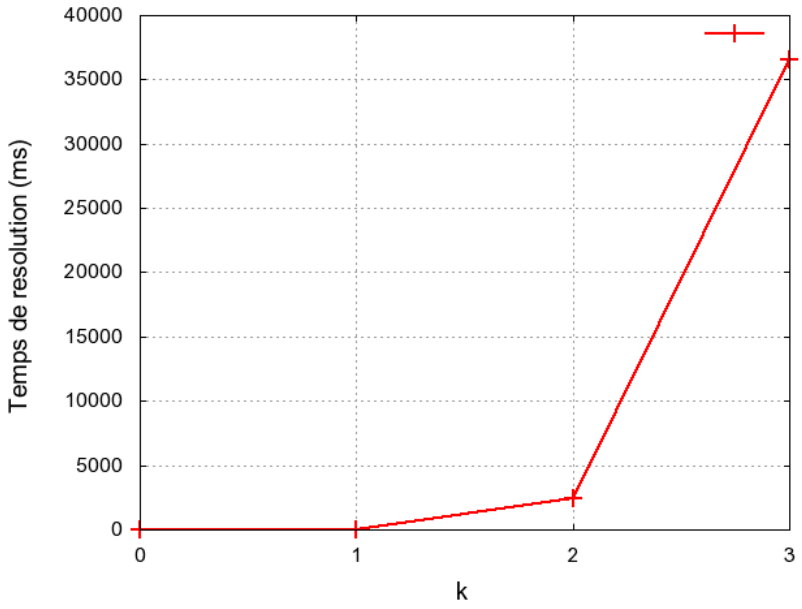


FIGURE 3.17 : influence de l'arité des relations sur le temps de calcul.

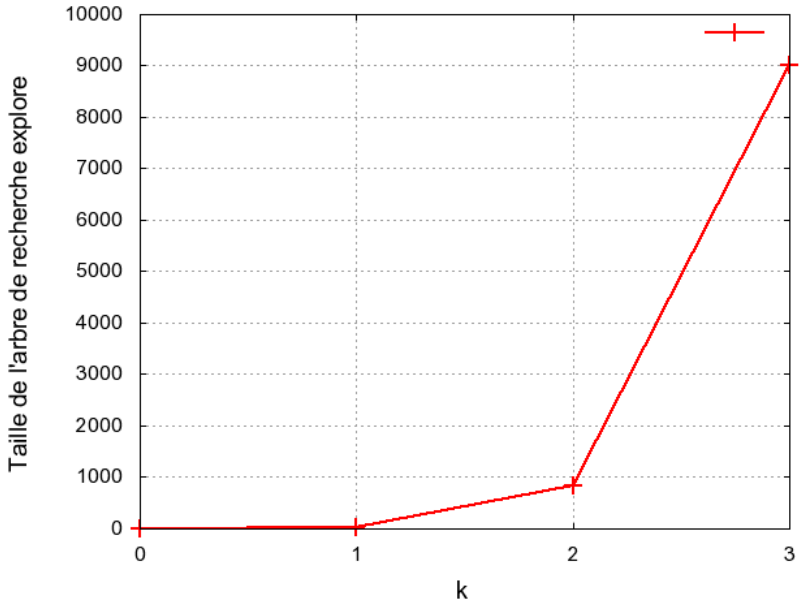


FIGURE 3.18 : influence de l'arité des relations sur la taille de l'arbre de recherche exploré.

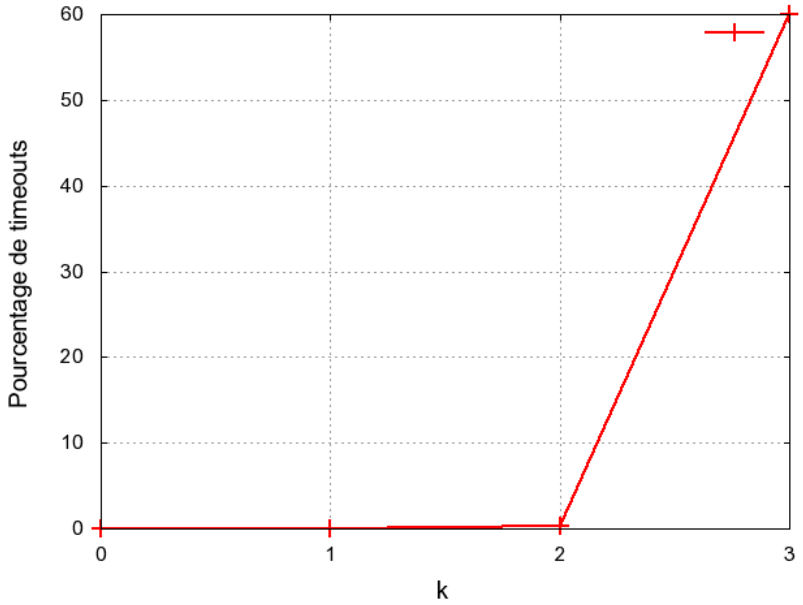


FIGURE 3.19 : influence de l'arité des relations sur le pourcentage de timeouts.

2 il y a 2^{64} - 2^{8^2*1} - complétions totales possibles et pour $k = 3$ il y a 2^{512} - 2^{8^3*1} - complétions totales possibles). Dans la suite, afin de minimiser l'influence des timeouts sur les résultats, nous choisissons $k = 2$.

3.6.6 Influence du nombre de relations

On s'attend à ce que l'ajout de nouvelles relations dans les graphes amène une augmentation du temps de calcul et de la taille de l'arbre de recherche exploré puisque le nombre de relations est un facteur du nombre de complétions totales de G. Mais d'un autre côté l'ajout de relations contraint le nombre d'images possibles des sommets relations du graphe source dans le cible. On peut donc s'attendre à ce que la difficulté de résolution augmente jusqu'à une valeur seuil du paramètre *nbE*, puis diminue après cette valeur seuil (lorsque le graphe source sera très contraint). Notons que nous avons opté pour une densité *par* relation comme paramètre, ce qui induit que *DS* et *DC* dépendent de *nbE*. En effet, prenons deux graphes F_1 et F_2 composés de 8 sommets termes et de sommets relations de relation respectivement r et r ou s (avec $k = 2$). Si nous choisissons pour F_1 et F_2 une densité égale à 0.1875, F_1 sera composé de 12 sommets relations (tous étiquetés $\sim r$) alors que F_2 sera composé de 24 sommets relations (12 étiquetés $\sim r$ et 12 étiquetés $\sim s$). Pour que F_2 soit composé de 12 sommets relations, il faudrait qu'il ait une densité de 9.375% soit $d = \frac{12}{8^2*2}$. Notons également que pour $neg = 50\%$, nous forçons la densité d'un graphe construit à partir de n relations à ne pas être inférieure à la densité assurant l'exis-

tence, pour chaque relation r, d'au moins 2 sommets relations respectivement étiquetés $+r$ et $-r$. Nous avons alors : $nbSomR_{min} = nbE * 2$ (au minimum 2 sommets relations pour chaque relation). Ainsi le nombre de sommets relations minimal augmentera avec le nombre de relations. Prenons par exemple deux graphes F_1 et F_2 composés de 8 sommets termes et de sommets relations étiquetés respectivement par 5 et par 10 relations différentes (avec $k = 2$). Les nombres de sommets relations minimaux respectivement pour F_1 et F_2 seront $nbSomR_{min1} = 10$ et $nbSomR_{min2} = 20$. Nous imposons également que le nombre de sommets relations par relation soit pair, afin qu'il y ait autant de sommets relations positifs et négatifs dans le graphe. Ainsi pour un graphe F construit à partir de x relations, les densités de F étudiés sont telles que le nombre de sommets relations de F soit un multiple du nombre minimal de sommets relations de F. Prenons par exemple un graphe F et supposons que $nbT = 8$, $k = 2$ et $nbE = 3$. On a $nbSomR_{min} = 6$. Ainsi les densités $d = 3.125\%$ ou $d = 6.25\%$ par exemple sont possibles (il y aura respectivement 6 et 12 sommets relations dans F, soit respectivement 2 et 4 sommets relations par relation) alors que la densité $d = 4.6875\%$ par exemple est impossible (il y aura 9 sommets relations dans F, soit 3 sommets relations par relation, ce qui contredit la parité par relation).

Nos intuitions sont validées par les expérimentations : prenons les figures 3.20 et 3.21 qui ont été obtenues pour $nbTS = nbTC = 8$, $k = 2$, $DS =$ x et $DC =$ y, où x et y correspondent (pour chaque point) à la valeur de DS et de DC telle que la difficulté de résolution de l'instance soit maximale, respectivement en mesurant le temps de calcul et la taille de l'arbre de recherche exploré en faisant varier en abscisse le nombre de relations. Nous observons que la difficulté augmente jusqu'à une valeur seuil du nombre de relations (2 ici, avec un temps de résolution de 13308ms et une taille d'arbre de recherche égale à 6501) et qu'à partir de cette valeur, elle diminue pratiquement continuellement. Cette diminution s'explique à partir de $nbE = 7$ puisque le nombre de sommets relations minimal ne cesse de croître (voir la table 3.3 pour plus de détails). Nous observons une augmentation de la difficulté pour $nbE = 5$, qui peut être dû à une distribution du graphe source (qui est composé de 10 sommets relations avec deux sommets relations par étiquette) « difficile », c'est-à-dire qu'elle assure la construction de nombreuses complétions.

Protocole d'expérimentation

Au vu des résultats expérimentaux obtenus, nous effectuons les choix suivants pour les expérimentations réalisées au chapitre 4 (sauf mention du contraire) :

- $nbTS = nbTC = 8$;
- $c = 0\%$;
- $nbE = 2$;
- $k = 2$;
- $DS = 9.375\%$;
- DC est le paramètre variable ;
- $neg = 50\%$;

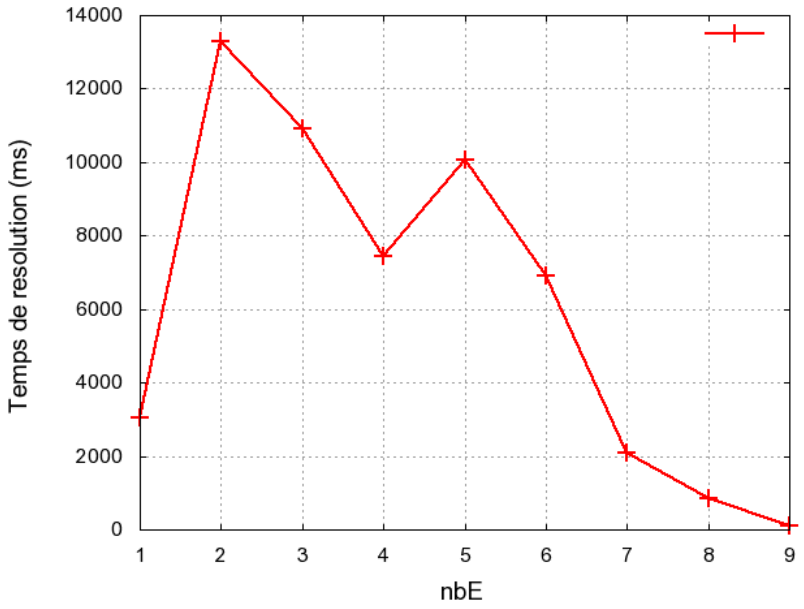


FIGURE 3.20 : influence du nombre de relations sur le temps de calcul.

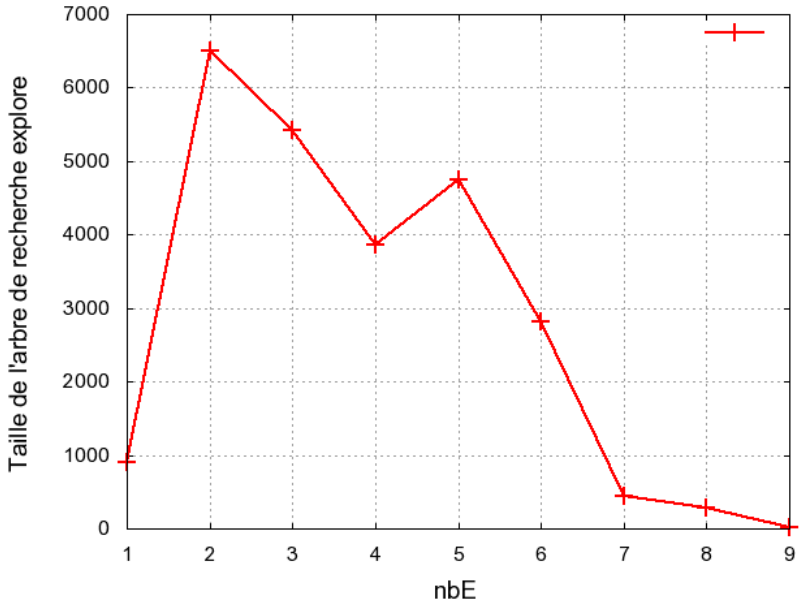


FIGURE 3.21 : influence du nombre de relations sur la taille de l'arbre de recherche exploré.

nbE	$nbSomR_{min}$	DS	nbSomR
1	2	15.625	10
2	4	9.375	12
3	6	6.25	12
4	8	3.125	8
5	10	3.125	10
6	12	3.125	12
7	14	3.125	14
8	16	3.1258	16
9	18	3.125	18

TABLE 3.3 : influence du nombre de relations sur la densité du graphe source.

– Les mesures effectués seront : le temps d'exécution, la taille de l'arbre de recherche exploré et le nombre de tests d'homomorphisme (seulement dans le cas où ce nombre diffère du nombre de nœuds de l'arbre de recherche).

CHAPITRE

4 Affinements et nouveaux algorithmes

Préambule

Dans ce chapitre, nous proposons trois raffinements à l'algorithme `recCheck` *et nous les analysons expérimentalement. Nous présentons alors un nouvel algorithme raffiné. Puis nous analysons la propriété de Wei et Lausen [2003] et mettons en évidence la présence d'une erreur dans l'algorithme qu'ils proposent. Nous proposons alors deux nouveaux algorithmes, tous deux basés sur la propriété fondamentale du second schéma d'algorithme : ils corrigent l'erreur détectée mais utilisent des parcours différents de l'espace de recherche. Puis nous proposons un nouvel algorithme basé sur une approche différente de celles connues à ce jour, qui conduit à tester l'insatisfiabilité d'une forme clausale propositionnelle, ce qui permet d'utiliser un solveur SAT. Finalement, nous comparons expérimentalement les quatre algorithmes entre eux.*

Sommaire

4.1 Raffinements de l'algorithme `recCheck` **59**
4.2 Analyse et reformulation de l'algorithme de Wei et Lausen **69**
4.3 Algorithme UNSATCheck **84**
4.4 Comparaison des algorithmes **92**

4.1 Raffinements de l'algorithme `recCheck`

Comme mentionné dans le chapitre 2, section 2.4.4, l'espace de recherche menant de G à ses complétions totales est partiellement ordonné par la relation d'inclusion « sous-

graphe de ». L'algorithme `recCheck` l'explore par un parcours en profondeur, qui définit une arborescence binaire de racine G (cf. figure 2.9 du chapitre 2, section 2.4.4).

Dans cette partie, nous proposons et testons expérimentalement trois affinements à l'algorithme `recCheck` visant à élaguer l'arbre de recherche. Les deux premiers concernent la façon d'explorer l'arbre de recherche alors que le troisième porte sur la notion de filtrage. Plus précisément, ils concernent les aspects suivants :

1. le choix du prochain littéral de complétion ;
2. le choix du fils à explorer en priorité ;
3. la mise en place d'un filtrage à chaque nœud de l'arbre de recherche.

Nous les indiquons dans un premier temps sur l'algorithme `recCheck` avant de les développer en détail et de proposer en section 4.1.4 l'algorithme les prenant en compte.

Algorithme 3: **recCheck**(G)

Données: un graphe polarisé consistant G
Accès: un graphe polarisé F, le vocabulaire de complétion $\mathcal{V}_c$ de F et G
Résultat: vrai si F se déduit de G, faux sinon
début
  si *il y a un homomorphisme de F dans G* **alors**
    retourner *vrai* ;
  si *G est complet par rapport à $\mathcal{V}_c$* **alors**
    retourner *faux* ;
(3) /* *Mise en place d'un filtrage dynamique* */
(1) Choix du littéral Choisir $r \in \mathcal{V}$ et $t_1, \ldots, t_k$ dans G tel que $+r(t_1, \ldots, t_k) \notin G$ et $-r(t_1, \ldots, t_k) \notin G$;
  Soit G' obtenu de G en ajoutant $+r(t_1, \ldots, t_k)$;
  Soit G'' obtenu de G en ajoutant $-r(t_1, \ldots, t_k)$;
(2) Fils prioritaire **retourner** ***recCheck***(G') *ET* ***recCheck***(G'') *;*
fin

4.1.1 Choix du prochain littéral de complétion

L'exploration de l'espace de recherche se faisant à l'aide d'un parcours en profondeur, le choix du prochain sommet relation à ajouter, c'est-à-dire $\sim r(t_1, \ldots, t_k)$ dans `recCheck` (point 1), est primordial. Ceci est illustré par la figure 4.1 : en choisissant $\sim r(e, b)$ comme premier littéral de complétion, trois nœuds suffisent à prouver que $G \vdash F$. En effet, il y a un homomorphisme de F dans G' ($h' = \{w \mapsto e, x \mapsto b, y \mapsto e, z \mapsto b\}$) et un homomorphisme de F dans G'' ($h'' = \{w \mapsto f, x \mapsto e, y \mapsto b, z \mapsto c\}$). Dans le pire des cas (c'est-à-dire la construction de l'ensemble des complétions totales de G), plus de 2 milliards

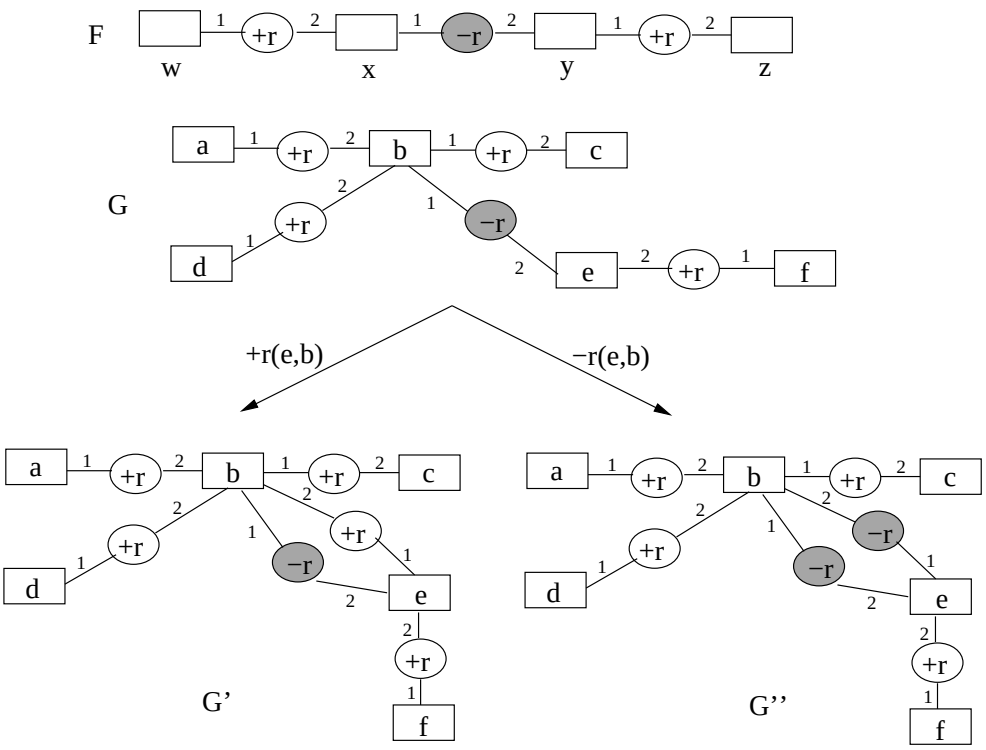


FIGURE 4.1 : un bon choix de littéral de complétion : $\sim r(e,b)$.

$(2^{(6^2-5)} = 2^{31})$ de complétions totales auraient été construites, soit le double de nœuds parcourus $(2^{31} + (2^{31} - 1))$.

Plus précisément, nous choisissons une relation $r \in \mathcal{R}$ d'arité k, ainsi qu'une liste de k sommets termes (non nécessairement distincts) $t_1, \ldots, t_k$ de G tels que $+r(t_1, \ldots, t_k) \notin G$ et $-r(t_1, \ldots, t_k) \notin G$. Les fils de G (c'est-à-dire G' et G'') sont obtenus en lui ajoutant respectivement $+r(t_1, \ldots, t_k)$ et $-r(t_1, \ldots, t_k)$. Une façon brutale de procéder consiste à faire un choix aléatoire de r et de $t_1, \ldots, t_k$.

Notre proposition est de guider ce choix par un homomorphisme compatible, soit h, d'un F^{max} (cf. chapitre 2, section 2.4.4) dans la complétion courante, soit G. Plus précisément, les sommets relations frontières de F par rapport au F^{max} choisi peuvent être divisés en deux catégories. Dans la première, nous avons les sommets relations frontières $\sim r(e_1, \ldots, e_n)$ tels que $\sim r(h(e_1), \ldots, h(e_n)) \in G$, qui peuvent être utilisés pour *étendre* h ; si tous les sommets relations frontières appartiennent à cette catégorie, h peut être étendu à un homomorphisme de F dans G. Le choix du littéral de complétion est basé sur un sommet relation frontière $\sim r(e_1, \ldots, e_n)$ appartenant à la seconde catégorie : r est sa relation et $t_1, \ldots, t_n = h(e_1), \ldots, h(e_n)$ ses voisins termes (il est à noter que ni $\sim r(h(e_1), \ldots, h(e_n))$ ni $\overline{\sim r}(h(e_1), \ldots, h(e_n))$ n'appartiennent à G, puisque respectivement $\sim r(e_1, \ldots, e_n)$ appartient à la seconde catégorie et h est compatible). Nous avons alors les définitions suivantes :

Définition 4.1 (Sommet relation extensif, manquant) *Soient F et G deux graphes polarisés et h un homomorphisme compatible de F' (un sous-graphe de F) dans G. Soit $\sim r(t_1,\ldots,t_k)$ un sommet relation de $F \setminus F'$. Il est dit* extensif *à G par rapport à h s'il existe $\sim r(h(t_1),\ldots,h(t_k))$ dans G, sinon il est dit* manquant *à G par rapport à h.*

Intuitivement, l'idée est de donner la priorité aux sommets relations potentiellement capables de transformer l'homomorphisme compatible h en un homomorphisme de F dans une complétion partielle de G, soit G'. Si c'est le cas, toutes les complétions qui incluent G' seront éliminées. Le sous-algorithme `choixLittéralCompletion` (algorithme 4) s'appuie sur ce guidage. Cet algorithme prend en entrée un graphe polarisé G et retourne un littéral qui servira à compléter G. Il a en accès F et F', un sous-graphe pur de F maximal pour l'inclusion, que l'on appelle par la suite *sous-graphe de guidage*. Il recherche un homomorphisme compatible de F' dans G (par la fonction *recHomComp(F,F',G)*), soit h, que l'on appelle par la suite *homomorphisme compatible de guidage*. S'il n'y en a pas, il retourne un échec. Sinon, il choisit un sommet relation $\sim r(t_1,\ldots,t_k) \in F \setminus F'$ manquant à G par rapport à h et retourne le littéral de complétion $\sim r(h(t_1),\ldots,h(t_k))$. Notons que comme ni $\sim r(h(t_1),\ldots,h(t_k))$ ni $\overline{\sim r}(h(t_1),\ldots,h(t_k))$ n'appartiennent à G, on peut retourner au choix l'un ou l'autre. Le signe de ce littéral de complétion sera utilisée par une autre heuristique (voir section 4.1.2).

Algorithme 4: **choixLittéralCompletion**(G)

Données: G un graphe polarisé consistant
Accès: F, F' un sous-graphe pur de F maximal pour l'inclusion
Résultat: un littéral de complétion de G s'il existe, échec sinon
début
  $h \leftarrow$ **recHomComp(F,F',G)** ;
  si h = *échec* **alors retourner** *échec* ;
  Choisir un sommet relation frontière $\sim r(t_1,\ldots,t_k) \in F \setminus F'$ manquant à G par rapport à h ;
  retourner $\sim r(h(t_1),\ldots,h(t_k))$;
fin

La figure 4.2 montre les résultats obtenus avec les choix suivants :
- *choix aléatoire* ;
- *choix aléatoire + filtrage* : choix aléatoire et F^+ comme filtre (c'est-à-dire qu'à chaque étape de complétion, un homomorphisme compatible de F^+ dans G est recherché : s'il n'en existe pas, on retourne un échec) ;
- *choix guidé par un homomorphisme compatible* : F^+ est utilisé à la fois comme filtre et comme guide.

Notons que le choix guidé produit un filtre implicite : en effet, quand un homomorphisme compatible de F^+ dans une complétion partielle de G (soit G') est recherché, un

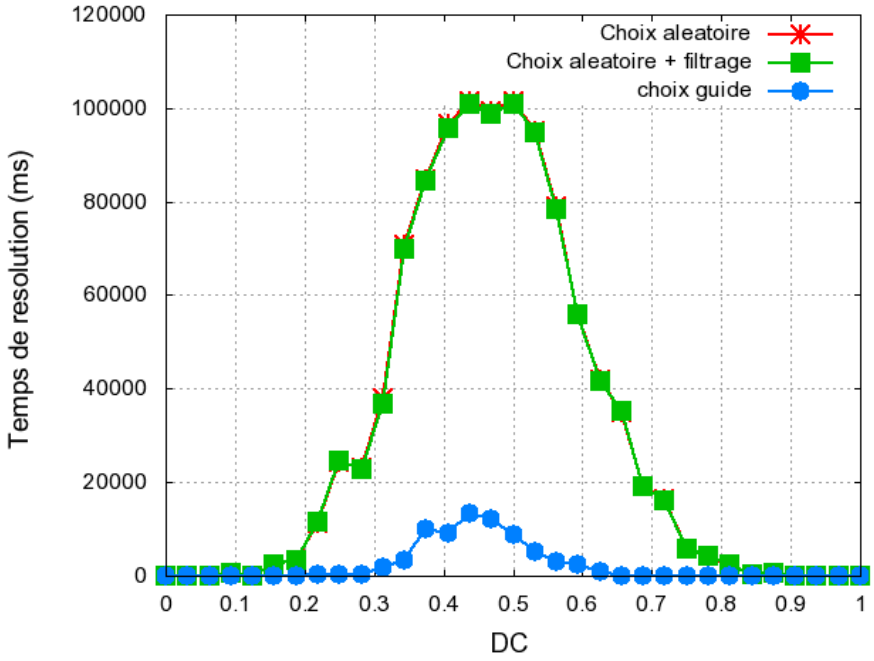


FIGURE 4.2 : influence du choix de complétion : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$.

échec est retourné s'il n'en existe pas (puisque $G' \not\vdash F$). Afin de discriminer uniquement les heuristiques de choix, nous considérons aussi un choix aléatoire avec un filtre identique au choix guidé.

Comme on pouvait s'y attendre, le choix guidé est toujours meilleur que le choix aléatoire (avec ou sans filtre) : par exemple pour $DC = 43.75\%$, le choix guidé est pratiquement 8 fois meilleur que le choix aléatoire (environ 13s contre 1min41). Notons que le temps de calcul est quasiment identique pour le choix aléatoire avec ou sans filtre. Ceci s'explique par le fait qu'avec le filtrage en moyenne moins de nœuds sont parcourus, mais parallèlement des tests d'homomorphisme compatible doivent être effectués (un à chaque nœud). Ici, la différence entre le temps gagné à ne pas parcourir certains nœuds et le temps perdu à rechercher des homomorphismes est quasiment nulle.

Dans la suite, nous considérons le choix guidé par un homomorphisme compatible.

4.1.2 Choix du fils à parcourir en priorité

Cet affinement est basé sur l'observation que l'ajout d'un littéral de complétion à G a des incidences sur l'homomorphisme de guidage de G. Soit h l'homomorphisme de guidage de G et $\sim r(e_1,\ldots,e_n)$ le sommet relation de $F \setminus F'$ définissant le littéral de

complétion retourné par l'algorithme `choixLitteralCompletion`. Si nous ajoutons $\sim r(h(e_1),\ldots,h(e_n))$ à G, soit un sommet relation de même signe que $\sim r(e_1,\ldots,e_n)$, alors cet ajout *étendra* h, c'est-à-dire qu'il conservera h comme homomorphisme compatible de F' dans $G \cup \{\sim r(h(e_1),\ldots,h(e_n))\}$. Si l'ajout est $\overline{\sim r}(h(e_1),\ldots,h(e_n))$, alors il *contredira* h, c'est-à-dire que h ne sera plus un homomorphisme compatible de F' dans $G \cup \{\overline{\sim r}(h(e_1),\ldots,h(e_n))\}$. Plus précisément, nous appelons *h-extension* (resp. *h-contradiction*) la complétion obtenue de G en ajoutant $\sim r(h(e_1),\ldots,h(e_n))$ (resp. $\overline{\sim r}(h(e_1),\ldots,h(e_n))$). L'exemple 11 illustre ces deux notions. Intuitivement, en contredisant l'homomorphisme compatible de guidage on supprime pour le nœud fils un des homomorphismes compatibles de son père, ce qui revient à rechercher un échec plus rapidement.

Exemple 11. Considérons la figure 2.4 et $F^+ = \{+p(x), +s(x,y)\}$ comme sous-graphe de guidage. Le seul sommet relation frontière de F^+ par rapport à F est $-p(y)$. $h = \{(x,t),(y,u)\}$ est le seul homomorphisme compatible de F^+ dans G. La h-extension G_1 (resp. h-contradiction G_2) est obtenue en ajoutant $-p(u)$ (resp. $+p(u)$).

La figure 4.3 montre les résultats obtenus avec les choix suivants :
- *Extension* : parcours des *h-extensions* en priorité ;
- *Contradiction* : parcours des *h-contradictions* en priorité.

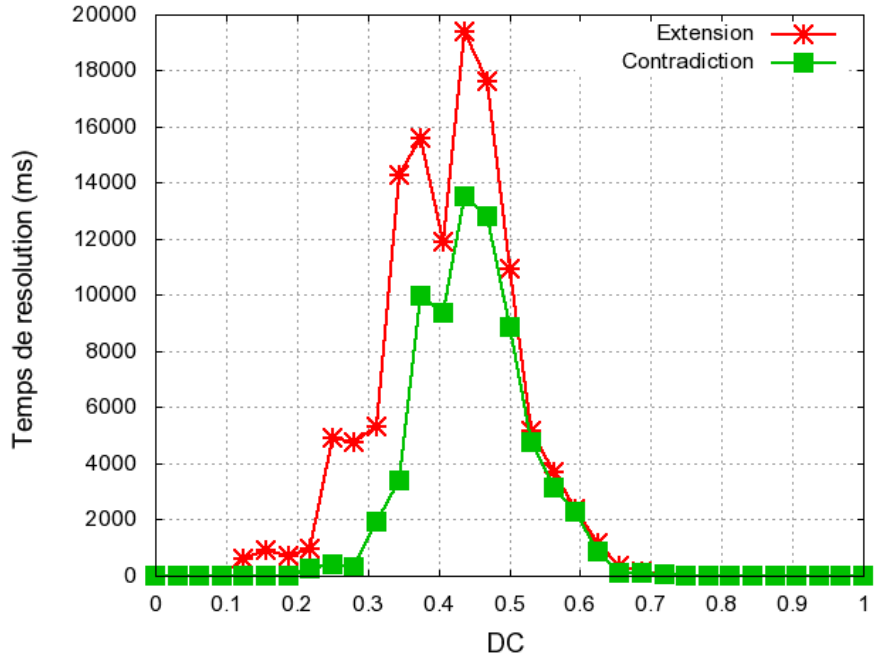


FIGURE 4.3 : influence de l'ordre d'exploration : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$.

Nous observons qu'il est toujours meilleur d'explorer en priorité les *h-contradictions*. Dans la suite, nous donnons la priorité au parcours des *h-contradictions*.

Lors de l'exploration d'une complétion correspondant à une *h-extension*, nous proposons également deux améliorations :

1. Nous ne calculons pas un nouvel homomorphisme compatible de guidage pour une *h-extension* G ; h restant compatible pour G, il devient l'homomorphisme compatible de guidage de G. (cf. exemple 11 : l'homomorphisme compatible de guidage de G devient celui de G_1). L'algorithme 5 prend en compte cette amélioration en ajoutant comme paramètre de l'algorithme `choixLittéralCompletion` un homomorphisme compatible de F' dans G.
2. Nous pouvons dans certains cas détecter un homomorphisme de F dans une *h-extension* sans le calculer : lors du calcul d'un homomorphisme compatible h d'un sous-graphe de F, soit F', dans la complétion courante, soit G', nous associons à h le nombre de sommets relations manquants à G' afin que h soit un homomorphisme de F dans G'. Si le nombre associé à h est égal à 1, alors nous ne parcourons pas la *h-extension*, soit G'', car le prochain ajout du littéral de complétion à G' assurera que h sera un homomorphisme de F dans G'' (cf. exemple 11 : nous ne parcourons pas G_1). L'algorithme 8 (proposé à la fin de cette section) prend en compte cette amélioration.

Algorithme 5: **choixLittéralCompletionPlus**(G, h)

Données: G un graphe polarisé consistant, h un homomorphisme compatible de F' dans G ou $\emptyset$
Accès: F, F' un sous-graphe pur de F maximal pour l'inclusion
Résultat: un littéral de complétion de G s'il existe et l'homomorphisme compatible de guidage associé, échec sinon

```
début
    si h = ∅ alors
        h ← recHomComp(F,F',G) ;
        si h = échec alors retourner échec ;
    Choisir un sommet relation ~r(t_1,...,t_k) ∈ F \ F' tel que ~r(h(t_1),...,h(t_k)) ∉ G et
    ~r̄(h(t_1),...,h(t_k)) ∉ G ;
    retourner ~r(h(t_1),...,h(t_k)), h ;
fin
```

4.1.3 Filtrage dynamique

Le dernier affinement consiste à exécuter un filtrage à chaque nœud de l'arbre de recherche. Une fois de plus, le but est de détecter plus rapidement un échec. Plus précisé-

ment, nous considérons un ensemble de F^{max} et testons pour chaque élément de cet ensemble, soit F', s'il existe un homomorphisme compatible de F' dans la complétion courante de G. Si le test échoue pour un F', nous concluons que $G \not\vdash F$.

Le sous-algorithme `filtrageDynamique` (algorithme 6) s'appuie sur ce filtrage. Cet algorithme prend en entrée un graphe polarisé G et retourne le succès ou l'échec du filtrage. Il a en accès un ensemble de F^{max} $E = \{F''_1, \ldots, F''_n\}$ où F''_i est appelé *sous-graphe de filtrage*. Pour chaque F''_i, on recherche un homomorphisme compatible de F''_i dans G. S'il n'y en a pas, on retourne l'échec du filtrage.

Algorithme 6: **filtrageDynamique**(G)

Données: un graphe polarisé consistant G
Accès: un ensemble de F^{max} $E = \{F''_1, \ldots, F''_n\}$ où F''_i est un sous-graphe de filtrage
Résultat: succès ou échec du filtrage
début
  pour chaque $F'' \in E$ **faire**
    si *il n'existe pas d'homomorphisme compatible de* F'' *dans* G **alors retourner** *échec* ;
  retourner *succès* ;
fin

Notons que, pour un sous-graphe de filtrage F' et une complétion partielle G' de père G, si nous avons calculé un homomorphisme compatible de F' dans G, soit h', il n'est pas nécessaire de relancer le test d'homomorphisme compatible de F' dans G' dans le cas où le dernier sommet relation ajouté à G' étend h' (c'est-à-dire que G' est une h'*-extension*). L'algorithme 7 prend en compte cette amélioration en gardant en mémoire les homomorphismes compatibles calculés précédemment.

Les figures 4.4, 4.5 et 4.6 montrent, pour des mesures différentes, les résultats obtenus avec les configurations suivantes :

- *Sans filtrage dynamique* : F^{Max} comme sous-graphe de guidage et aucun sous-graphe de filtrage ;
- *Avec filtrage dynamique* : F^{Max} comme sous-graphe de guidage et tous les autres F^{max} comme sous-graphes de filtrage.

La prise en compte du filtrage dynamique permet de diminuer la taille de l'arbre de recherche exploré (figure 4.4), ce qui n'est pas surprenant puisque le filtrage dynamique ajoute des contraintes pouvant provoquer un arrêt à chaque nœud de l'arbre de recherche. Le temps d'exécution est à peu près le même pour les deux configurations (figure 4.5) alors qu'avec le filtrage dynamique beaucoup plus de tests d'homomorphisme sont effectués (figure 4.6, en considérant la somme des tests d'homomorphisme et d'homomorphisme compatible). La table 4.1 donne les résultats précis pour $DC = 43.75\%$. Néanmoins, notre actuel algorithme de test d'homomorphisme doit pouvoir être amélioré en utilisant par

Algorithme 7: **filtrageDynamiquePlus**(G)

Données: un graphe polarisé consistant G
Accès: un ensemble de couples $E = \{(F''_1, h''_1), \ldots, (F''_n, h''_n)\}$ où F''_i est un sous-graphe de filtrage et h''_i un homomorphisme compatible de F''_i dans le père de G (vide si G est la racine)
Résultat: succès ou échec du filtrage
début
  pour chaque $(F'', h'') \in E$ **faire**
    si h'' *est vide* ***ou*** G *est une* h''-contradiction **alors**
      $h'' \leftarrow$ **recHomComp(E,F",G)** ;
      si $h'' =$ *échec* **alors retourner** *échec* ;
  retourner *succès* ;
fin

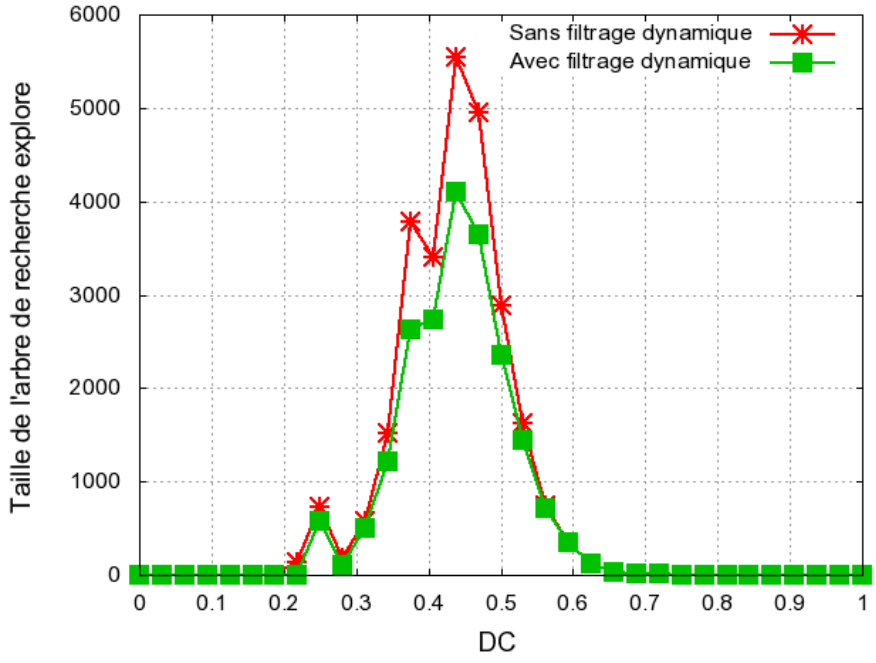


FIGURE 4.4 : influence du filtrage dynamique sur la taille de l'arbre de recherche exploré : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$.

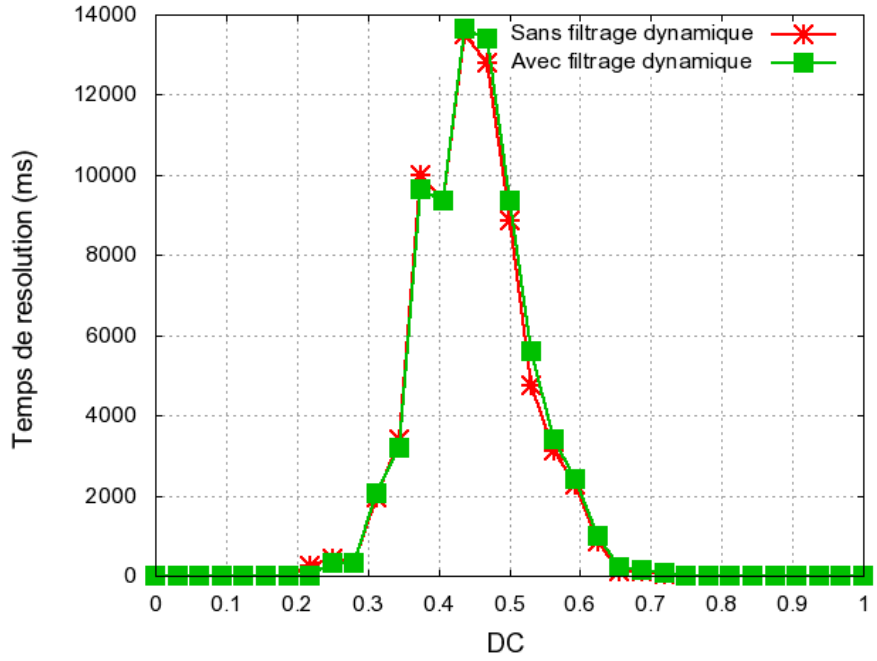


FIGURE 4.5 : influence du filtrage dynamique sur le temps d'exécution : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$.

Configuration	Temps (ms)	Taille de l'arbre	Tests d'hom.
Sans filtrage dynamique	13495	5541	7874
Avec filtrage dynamique	13662	4098	13602

TABLE 4.1 : influence du filtrage dynamique : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $DC = 43.75\%$, $neg = 50\%$.

exemple un solveur CSP ou le solveur Django [1] ; ainsi nos résultats montrent que la prise en compte du filtrage dynamique est le meilleur choix.

4.1.4 L'algorithme `recCheckPlus`

L'algorithme finalement obtenu, qui prend en compte les trois affinements et les améliorations secondaires discutées, est appelé `recCheckPlus` (cf. algorithme 8). Il introduit deux sous-algorithmes : **filtrageDynamiquePlus** et **choixLitteralCompletionPlus** ; il est initialement appelé avec $(G, \emptyset)$. Le second paramètre est utilisé pour mémoriser l'homo-

1. http ://tao.lri.fr/tiki-index.php ?page=Django

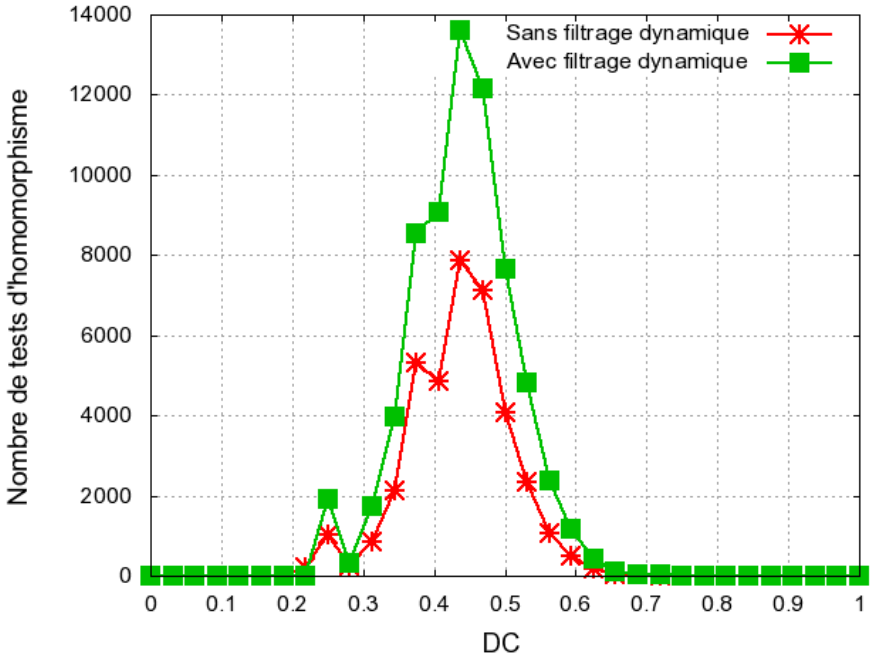


FIGURE 4.6 : influence du filtrage dynamique sur le nombre de tests d'homomorphisme effectués : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$.

morphisme compatible trouvé pour le père du nœud courant, soit h, pour le cas où ce nœud est une h-*extension* (soit G'' dans l'algorithme) ; autrement, l'homomorphisme compatible h a été contredit et on doit en rechercher un nouveau, ce qui est réalisé dans le sous-algorithme **choixLitteralCompletionPlus** (algorithme 5).

4.2 Analyse et reformulation de l'algorithme de Wei et Lausen

Dans cette section, nous étudions le travail de Wei et Lausen [2003] et reformulons dans notre cadre d'étude les propriétés sur lesquelles ils fondent la recherche d'inclusion de requêtes conjonctives avec négation. Dans un deuxième temps, nous analysons l'algorithme qu'ils proposent dans l'annexe de cet article et mettons en évidence une erreur qui le rend incorrect [2]. Nous proposons alors deux nouveaux algorithmes, `breadthCheck` et `recAND-check`, tous deux basés sur les propriétés de Wei et Lausen [2003], qui corrigent l'erreur détectée mais utilisent des parcours différents de l'espace de recherche. Notons que ces

2. La preuve de correction d'un algorithme apporte l'assurance que si l'algorithme termine en donnant une proposition de solution, alors celle-ci est correcte, elle est effectivement une solution au problème posé.

Algorithme 8: **recCheckPlus**(G,h)

Données: un graphe polarisé consistant G, un homomorphisme compatible h du sous-graphe de guidage dans le père de G (vide pour la racine)
Accès: un graphe polarisé F, un sous-graphe de guidage pur maximal pour l'inclusion F' de F, le vocabulaire de complétion $\mathcal{V}$ de F et G, un ensemble de couples $E = \{(F''_1, h''_1), \ldots, (F''_n, h''_n)\}$
Résultat: vrai si F se déduit de G, faux sinon
début
  si *il y a un homomorphisme de F dans G* **alors retourner** *vrai* ;
  si *G complet par rapport à* $\mathcal{V}$ **alors retourner** *faux* ;
(3) **si filtrageDynamiquePlus(G)** = *échec* **alors retourner** *faux* ;
(1) $l, h \leftarrow$ **choixLitteralCompletionPlus(G,h)** ;
  si l = *échec* **alors retourner** *faux* ;
  Soit G' obtenu de G en ajoutant $\bar{l}$;
  Soit G'' obtenu de G en ajoutant l ;
  Soit n le nombre de sommets relations manquants à G afin que h soit un homomorphisme de F dans G ;
(2) **retourner recCheckPlus**$(G', \emptyset)$ *ET* $(n = 1$ *OU* **recCheckPlus**$(G'', h))$;
fin

résultats ont été obtenus à la fin de la durée de thèse et sont les moins développés (notamment ceux qui touchent à l'algorithme `recANDcheck`).

4.2.1 Propriétés fondamentales

Dans la suite, nous situons les travaux de [Wei et Lausen, 2003] dans le cadre des graphes polarisés, même si à l'origine ils ne sont pas formulés de cette façon : ils s'intéressent au problème $IRC^{\neg}$ (« $q_1 \sqsubseteq q_2$? »). Pour faire ce changement de formalisme, nous associons deux graphes polarisés F et G à deux $RC^{\neg}$ q_2 et q_1 et le test « $q_1 \sqsubseteq q_2$? » devient « $G \vdash F$? ». Dans [Wei et Lausen, 2003], toutes les requêtes sont supposées *safe*. Néanmoins nous pouvons considérer des graphes polarisés respectant ou non cette propriété sans incidence sur la résolution du problème.

L'algorithme proposé dans [Wei et Lausen, 2003] repose sur les deux théorèmes suivants (rappelons que F^p représente la partie positive de F, c'est-à-dire tous les sommets relations positifs de F) :

Théorème 4.2 *[Wei et Lausen, 2003] Soient F et G deux graphes polarisés. Supposons que $G^p \vdash F^p$, et soient $h_1, \ldots, h_n$ les homomorphismes de F^p dans G^p. Si, pour tout homomorphisme h_i $(i = 1 \ldots n)$, il existe au moins un sommet relation négatif $-r(t_1, \ldots, t_k)$ dans F tel que $+r(h_i(t_1), \ldots, h_i(t_k))$ appartient à G, alors $G \nvdash F$.*

Théorème 4.3 *[Wei et Lausen, 2003] Soient F et G deux graphes polarisés. On a $G \vdash F$ si et seulement si (1) il existe un homomorphisme h de F^p dans G^p et (2) pour chaque sommet relation négatif $-r(t_1,\ldots,t_k)$ de F, on a $G' \vdash F$, où G' est la complétion de G obtenue par l'ajout de $+r(h(t_1),\ldots,h(t_k))$.*

Notons tout d'abord que le théorème 4.2 peut être reformulé en terme d'homomorphisme compatible. Nous obtenons alors la propriété suivante, qui est une spécialisation de la propriété 2.2 (tirée de [Leclère et Mugnier, 2007]) introduite au chapitre 2, section 2.4.4 :

Propriété 4.1 *Soient F et G deux graphes polarisés. S'il n'y a pas d'homomorphisme compatible de F^p dans G alors $G \not\vdash F$.*

Nous observons également que nous retrouvons implicitement, dans le théorème 4.3, les notions d'homomorphisme compatible, de sommet relation manquant et de h-contradiction. En effet, l'homomorphisme de la première condition doit obligatoirement être un homomorphisme compatible de F^p dans G, sinon il conduira à la construction d'au moins une complétion G' redondante et le processus sera alors infini. Soit h un homomorphisme incompatible de F^p dans G, alors il existe $-r(t_1,\ldots,t_k)$ dans F et $+r(h(t_1),\ldots,h(t_k))$ dans G. D'après la condition 2 du théorème 4.3, on va construire la complétion G' de G obtenue par l'ajout de $+r(h(t_1),\ldots,h(t_k))$. Or comme $+r(h(t_1),\ldots,h(t_k))$ appartient déjà à G, G' sera redondante par rapport à G (cela revient à considérer G en boucle). D'autre part, la condition 2 prend en compte tous les sommet relations négatifs de F alors qu'il suffit de choisir les sommets relations manquants à G par rapport à l'homomorphisme h considéré. En effet les sommets relations extensifs à G par rapport à h ne sont pas utiles : soit $-r(t_1,\ldots,t_k)$ un tel sommet relation extensif, alors par définition $-r(h(t_1),\ldots,h(t_k))$ appartient à G. D'après la condition 2 du théorème 4.3, on va construire la complétion G' de G obtenue par l'ajout de $+r(h(t_1),\ldots,h(t_k))$. Or comme $-r(h(t_1),\ldots,h(t_k))$ appartient déjà à G, G' sera inconsistante, puisque la complétion G' obtenue sera inconsistante (on aura alors automatiquement $G' \vdash F$). Enfin, la notion de h-contradiction est immédiate. Nous obtenons alors le théorème de Wei et Lausen [2003] reformulé suivant :

Théorème 4.4 *[Wei et Lausen, 2003] **reformulé**. Soient F et G deux graphes polarisés. On a $G \vdash F$ si et seulement si (1) il existe un homomorphisme **compatible** h de F^p dans G et (2) pour chaque sommet relation **manquant** $-r(t_1,\ldots,t_k)$ à G par rapport à h, on a $G' \vdash F$, où G' est la h-**contradiction** obtenue de G par l'ajout de $+r(h(t_1),\ldots,h(t_k))$.*

Remarquons que contrairement à l'approche de Leclère et Mugnier [2007] qui parcourt aussi bien les h-contradictions que les h-extensions, ici on explore seulement les h-contradictions. Néanmoins nous verrons par la suite que certaines h-extensions sont implicitement parcourues et que toutes les complétions totales sont couvertes.

En nous basant sur le théorème 4.4, nous donnons une première intuition du schéma d'algorithme récursif qui pourrait en résulter (cf. algorithme 9). Notons que nous utilisons l'instruction « choisir **un** homomorphisme compatible… » alors que le théorème 4.4 suggère plutôt une instruction de la forme « **pour chaque** homomorphisme compatible… ». En effet, s'il existe une complétion G' de G telle que $G' \nvdash F$, alors nous pouvons directement conclure que $G \nvdash F$ (G' est un contre-exemple de la propriété 4.1). Cette condition d'arrêt est donc globale. Ainsi quels que soient les choix d'homomorphismes compatibles effectués, nous conclurons toujours que $G \nvdash F$. Et si $G \vdash F$, alors nous aurons $G' \vdash F$ pour toute complétion G' de G. Donc quels que soient les choix d'homomorphismes compatibles effectués nous conclurons toujours que $G \vdash F$.

Algorithme 9: **schémaPropWL**(G)

Données: un graphe polarisé consistant G
Accès: un graphe polarisé F, la partie positive F^p de F
Résultat: vrai si F se déduit de G, faux sinon
début
  si *il n'y a pas d'homomorphisme compatible de* F^p *dans* G **alors**
    retourner *faux* ;
  sinon
    Choisir un homomorphisme compatible h de F^p dans G ;
    pour chaque *sommet relation* $-r(t_1,\ldots,t_k) \in F \setminus F^p$ *manquant à* G *par rapport à* h **faire**
      si *NON(****schémaPropWL****(*$G \cup \{+r(t_1,\ldots,t_k)\}$*))* **alors retourner** *faux* ;
    retourner *vrai* ;
fin

La figure 4.7 présente un arbre de résolution qui pourrait être construit à partir de ce schéma d'algorithme : pour prouver que $G \vdash F$ nous devons prouver que $(G_1 \vdash F) \wedge \cdots \wedge (G_n \vdash F)$. Et pour prouver que $G_1 \vdash F$ nous devons prouver que $(G_{n+1} \vdash F) \wedge \cdots \wedge (G_m \vdash F)$ et ainsi de suite.

La différence fondamentale entre ce schéma et `recCheck` est la façon de parcourir l'espace de recherche. En effet, à chaque appel récursif, `recCheck` sépare l'espace de recherche en deux en construisant deux complétions « opposées » (respectivement une h-extension et une h-contradiction), alors qu'avec ce schéma on construit n h-contradictions, où n est le nombre de sommets relations manquants à la complétion courante par rapport à h. La couverture de l'ensemble des complétions totales est garantie d'une part par la construction de ces n h-contradictions, et d'autre part par la complétion totale G^{neg} (obtenue de G en ajoutant toutes les images par h des sommets relations manquants à G par rapport à h) dans laquelle F se déduit implicitement. Nous illustrons cette déduction implicite en considérant l'homomorphisme compatible

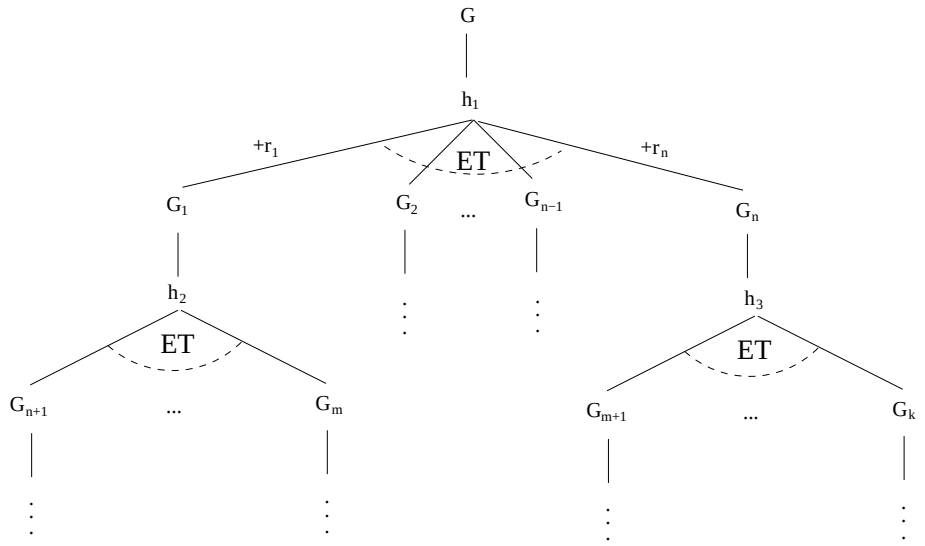


FIGURE 4.7 : un exemple d'arbre de recherche.

h_1 de la figure 4.7 : pour prouver que $G \vdash F$, nous avons à montrer que $(G_1 \vdash F) \wedge \cdots \wedge (G_n \vdash F)$, mais ce qui n'est pas précisé c'est qu'à ce niveau nous avons implicitement $(G^{neg} \vdash F)$, où G^{neg} est la complétion négative obtenue de G par l'ajout des sommets relations négatifs $-r_1(c_1,\ldots,c_k),\ldots,-r_n(d_1,\ldots,d_j)$. En effet, comme h_1 est compatible et que $-r_1(c_1,\ldots,c_k),\ldots,-r_n(d_1,\ldots,d_j)$ sont les images dans G de tous les sommets relations manquants à G par rapport à h_1, h_1 s'étend à un homomorphisme de F dans G^{neg}.

Néanmoins le parcours effectué par le schéma d'algorithme `schémaPropWL` pose un problème majeur : comment être sûr de ne pas construire et donc parcourir plusieurs fois les mêmes complétions ? Par exemple les complétions G_m et G_{m+1} de la figure 4.7 peuvent très bien être identiques.

4.2.2 L'algorithme de Wei et Lausen

Dans cette partie, nous détaillons le fonctionnement de l'algorithme de Wei et Lausen [2003], appelé `WL` dans la suite. Notons tout d'abord que la compréhension de cet algorithme est assez subjective, notamment sur certaines notions-clés telles que les conditions d'arrêt par exemple. Plusieurs échanges avec les auteurs nous ont été nécessaires avant de parvenir à bien les comprendre.

L'algorithme `WL` n'est pas basé sur le schéma présenté précédemment ; il utilise un schéma itératif (cf. algorithme 10) avec un parcours en largeur de l'espace de recherche, en calculant à chaque étape tous les homomorphismes compatibles de F^p dans G au lieu d'en considérer un seul.

Algorithme 10: **schémaWL**(G)

Données: un graphe polarisé consistant G
Accès: un graphe polarisé F, la partie positive F^p de F
Résultat: vrai si F se déduit de G, faux sinon
début
  Soit *AExplorer*← G la liste des complétions à explorer ;
  Soit *fin*←*indéterminé* une variable d'arrêt ;
  tant que *fin* = *indéterminé* **faire**
    AExplorer← Étendre toutes les complétions de la liste *AExplorer* ;
    / Tester les deux conditions d'arrêt */*
    si *il n'y a pas d'homomorphisme compatible de* F^p *dans une complétion qui était à explorer* **alors**
      fin←**faux** ;
    si *il existe une branche* h_i *(à partir de G) valide, c'est-à-dire que pour toutes les complétions* G' *se trouvant dans le sous-arbre de racine* h_i *on a* $G' \vdash F$ **alors**
      fin←**vrai** ;
  retourner *fin ;*
fin

Plus précisément, il développe un arbre de recherche ET/OU « particulier [3] ». Nous en présentons l'intuition sur la figure 4.8 : lors de la première étape on calcule tous les homomorphismes compatibles de F^p dans G, soit $h_1, \ldots, h_n$. Pour chaque h_i, on construit un ensemble de complétions $G_{i,1}, \ldots, G_{i,m}$ (pour plus de clarté nous faisons l'hypothèse que la valeur de m est la même pour chaque h_i, ce qui n'est pas le cas généralement car m est par définition dépendant de h_i). Pour prouver que $(G \vdash F)$ il faut alors prouver que $((G_{1,1} \vdash F) \wedge \cdots \wedge (G_{1,m} \vdash F)) \vee \cdots \vee ((G_{n,1} \vdash F) \wedge \cdots \wedge (G_{n,m} \vdash F))$. Si aucune des deux conditions d'arrêt n'est vérifiée on passe à la prochaine étape en étendant toutes les nouvelles complétions et ainsi de suite. Nous appelons *étape d'expansion k* le fait d'étendre toutes les complétions à la $k^{ième}$ itération.

Concernant le problème d'explorations multiples d'une même complétion, la solution proposée dans [Wei et Lausen, 2003] est de ne considérer que les **nouveaux** homomorphismes (compatibles) de F^p dans la complétion courante. La définition de « nouveau » homomorphisme est la suivante [4] :

3. Ce n'est pas exactement un arbre ET/OU car une des conditions d'arrêt (celle qui est basée sur la propriété 4.1) est globale, c'est-à-dire qu'elle permet de stopper complètement le processus.

4. Notons qu'elle ne figure pas dans [Wei et Lausen, 2003], ce sont les auteurs de l'article qui nous l'ont fournie.

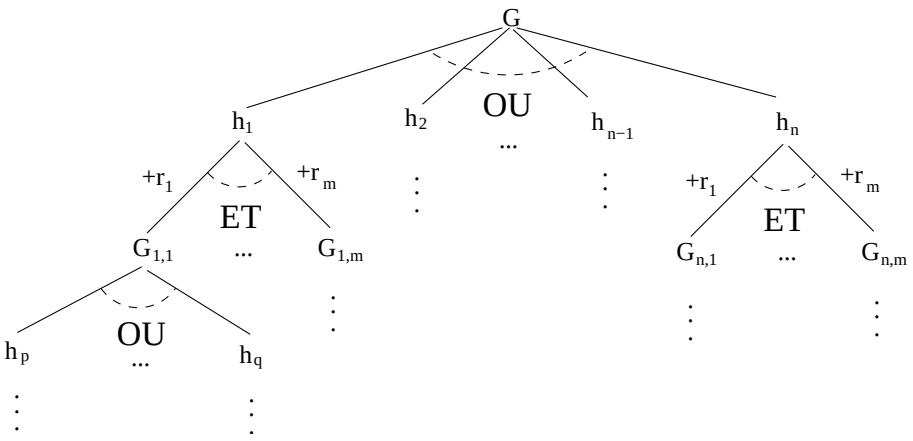


FIGURE 4.8 : un exemple d'arbre de recherche ET/OU.

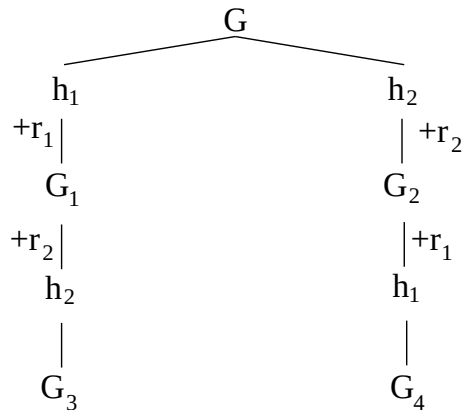


FIGURE 4.9 : parcours de la même complétion ($G_3 = G_4$).

Définition 4.5 *[Nouveau homomorphisme] Soit G' une complétion obtenue par l'ajout de $+r(e_1,\ldots,e_k)$. Un homomorphisme compatible h de F^p dans G' est dit* nouveau *s'il existe $+r(t_1,\ldots,t_k)$ dans F^p tel que $+r(h(t_1),\ldots,h(t_k)) \in G$ et $h(t_1),\ldots,h(t_k) = e_1,\ldots,e_k$.*

Néanmoins cette définition n'est pas satisfaisante pour deux raisons principales :

1. Elle n'évite pas les parcours multiples d'une même complétion. Prenons par exemple l'arbre de recherche 4.8 : les complétions $G_{1,1}$ et $G_{n,1}$ (par exemple) peuvent être obtenues par l'ajout du même sommet relation (notons que ce cas très simple à la première itération peut être généralisé pour des complétions de toutes tailles (cf. l'arbre de recherche de la figure 4.9 où $G_3 = G_4$)).
2. Elle rend l'algorithme `WL` incorrect, c'est-à-dire qu'il peut rater des solutions. Par exemple si nous avons deux nœuds, issus de deux branches différentes, correspon-

dant à deux complétions obtenues respectivement par l'ajout de $+r(c_1,\ldots,c_k)$ et $+r(d_1,\ldots,d_k)$, où $(c_1,\ldots,c_k) \neq (d_1,\ldots,d_k)$, il est possible de ne pas étendre ces nœuds dans le cas où il n'y a pas de « *nouveaux* » homomorphismes. Néanmoins ils auraient pu être étendus en *réutilisant* certains homomorphismes précédents, et nous aurions alors par exemple le nœud obtenu par la combinaison de $+r(c_1,\ldots,c_k)$ et $+r(d_1,\ldots,d_k)$. L'exemple 12 illustre ce problème, prouvant ainsi que cette notion de « nouveau » homomorphisme est trop restrictive et peut empêcher la résolution de certaines instances ; on en conclut que l'algorithme `WL` est incorrect.

Exemple 12. Nous considérons les graphes de la figure 4.10 et cherchons à savoir si $G \vdash F$. Nous allons voir que F se déduit de G alors que l'algorithme `WL` conduit à un échec.

La figure 4.11 montre un arbre de recherche en profondeur prouvant que F se déduit de G (l'ensemble des complétions partielles couvrant toutes les complétions totales est $\{G_2, G_4, G_5, G_6\}$).

Nous montrons que l'algorithme `WL` conduit à un échec. La figure 4.12 montre l'arbre de recherche généré par l'algorithme `WL` après la première étape d'expansion : il y a trois homomorphismes compatibles de F^p dans G (cf. table 4.2). Il y a un seul sommet relation manquant à G par rapport à chaque h_i (respectivement $-r(x,z)$ pour h_1 et h_3 et $-r(y,z)$ pour h_2). On construit alors les complétions G_1, G_2 et G_3, obtenues de G respectivement par l'ajout de $+r(h_1(x),h_1(z))$ (c'est-à-dire $+r(a,c)$), $+r(h_2(y),h_2(z))$ (c'est-à-dire $+r(b,a)$) et $+r(h_3(x),h_3(z))$ (c'est-à-dire $+r(c,c)$). Puis nous testons les conditions d'arrêt : aucune des deux conditions d'arrêt n'est vérifiée. Nous passons donc à l'étape d'expansion suivante, avec désormais les trois complétions G_1, G_2 et G_3 à étendre. Il n'y a pas de *nouveaux* homomorphismes de F^p respectivement dans G_1, G_2 et G_3, nous ne construisons donc aucune nouvelle complétion. Nous testons les conditions d'arrêt : la première condition d'arrêt est vérifiée pour G_1, G_2 et G_3 ; l'algorithme `WL` conclut que F ne se déduit pas de G.

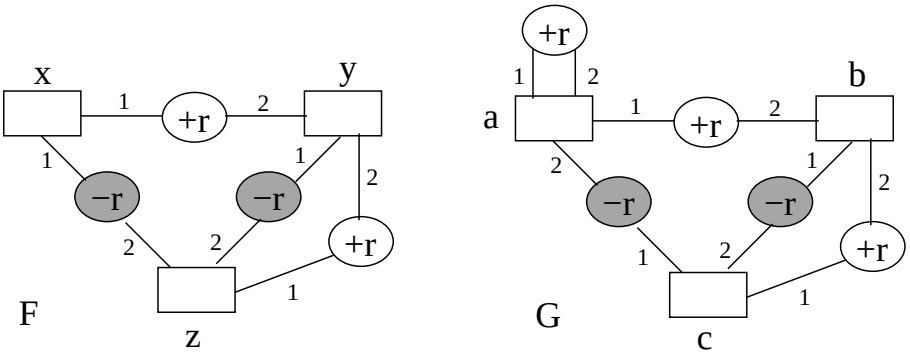


FIGURE 4.10 : deux graphes polarisés F et G.

Afin de résoudre l'erreur de `WL`, nous proposons trois solutions (en considérant *tous* les homomorphismes compatibles de F^p dans une complétion) :

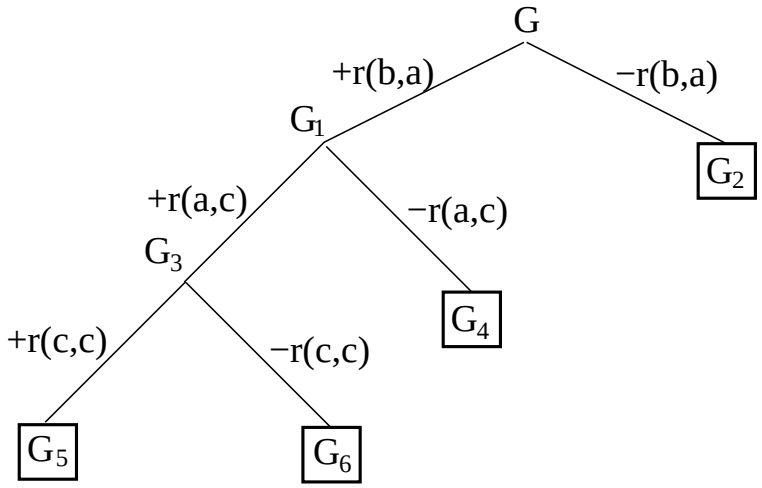


FIGURE 4.11 : un arbre de recherche en profondeur prouvant que $G \vdash F$.

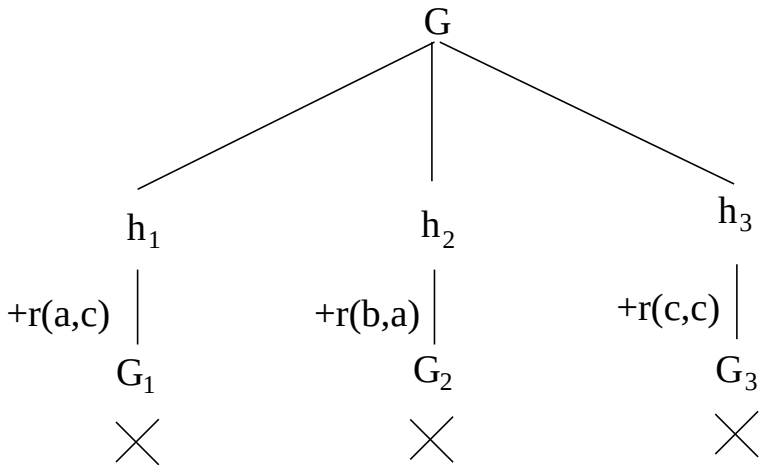


FIGURE 4.12 : l'arbre de recherche généré par `WL`.

h_1	$x \mapsto a, y \mapsto b, z \mapsto c$
h_2	$x \mapsto c, y \mapsto b, z \mapsto a$
h_3	$x \mapsto c, y \mapsto b, z \mapsto c$

TABLE 4.2 : les homomorphismes de l'exemple 12.

1. La première revient à permettre de parcourir plusieurs fois la même complétion. Dans le pire des cas, nous pouvons alors parcourir plus de complétions que la méthode brutale de Ullman [1997].
2. La seconde consiste à interdire la construction de deux complétions identiques. Ceci est rendu possible en mémorisant l'ensemble des complétions explorées. Néanmoins cette approche est exponentielle en espace.
3. La dernière consiste à « fusionner » les complétions identiques à la fin de chaque étape d'expansion. Cette « fusion » est possible car toutes les complétions d'une même étape d'expansion ont exactement la même taille. Néanmoins cette approche demande de tester deux-à-deux toutes les complétions d'une même étape (elle reste également exponentielle en espace).

4.2.3 Nouveaux algorithmes basés sur la propriété de Wei et Lausen

Dans cette partie, nous proposons deux algorithmes basés sur la propriété de Wei et Lausen [2003]. Le premier, appelé `breadthCheck`, est construit sur un schéma d'algorithme proche de celui proposé pour `WL` : il effectue un parcours en largeur de l'espace de recherche en calculant à chaque étape tous les homomorphismes compatibles (il corrige et optimise `WL`). Le second, appelé `recAndCheck`, est construit sur le schéma d'algorithme 9 : il effectue un parcours en profondeur de l'espace de recherche.

L'algorithme `breadthCheck` :

Les principales différences de `breadthCheck` par rapport à `WL` sont les suivantes :

1. Nous considérons un sous-graphe pur maximal pour l'inclusion F^{max} au lieu de la partie positive F^p.
2. Nous considérons tous les homomorphismes compatibles de F^+ dans une complétion et fusionnons les complétions identiques à la fin de chaque étape d'expansion.
3. Nous testons les conditions d'arrêt « à la volée » (c'est-à-dire durant la phase d'expansion de chaque complétion) et propageons l'information à tout l'arbre de recherche au lieu de tester les deux conditions d'arrêt à la fin de l'étape d'expansion.

Avant de détailler l'algorithme `breadthCheck`, nous présentons certaines définitions qui nous servirons par la suite :

Définition 4.6 (Graphe d'expansion, gp-nœud, h-nœud) *Un* graphe d'expansion *A est un graphe biparti orienté sans circuit et enraciné composé de deux types de nœuds : les* gp-nœuds *et les* h-nœuds. *On appelle* gp-nœud, *noté* N_G, *(resp.* h-nœud, *noté H) de A tout nœud de A auquel est associé un graphe polarisé G (resp. un homomorphisme). Tout nœud est soit marqué* succès *soit non marqué. Un arc entre un gp-nœud* N_G *(resp. h-nœud H) et*

un h-nœud H (resp. gp-nœud N_G) est étiqueté par un homomorphisme compatible h de F^+ dans G, soit $\mathrm{l}(N_G, H) = h$ *(resp. par le dernier sommet relation* $+r(t_1, \ldots, t_k)$ *ajouté à G, soit* $\mathrm{l}(H, N_G) = +r(t_1, \ldots, t_k)$*, où* l *est une fonction d'étiquetage des arcs de A.*

Nous rappelons également certaines notions classiques en théorie des graphes : soit A un graphe d'expansion et N un nœud quelconque de A. On appelle *fils de N* un nœud N' tel qu'il existe un arc (N, N') dans A et *père de N* un nœud N' tel qu'il existe un arc (N', N) dans A (les fils d'un gp-nœud sont des h-nœuds et vice-versa). On appelle *descendant de N* un nœud N_m de A tel qu'il existe une suite de nœuds $N, N_1, \ldots, N_{m-1}$ telle qu'il existe les arcs (N, N_1) et $(N_i, N_i + 1)$ pour $i = 1 \ldots m - 1$. La *descendance de N* est l'ensemble des descendants de N. N est une *feuille* s'il n'a aucun fils et la *racine* s'il n'a pas de père.

Nous définissons la fusion, dans un graphe d'expansion A, de deux gp-nœuds $N_{G'}$ et $N_{G''}$ (c'est-à-dire les nœuds auxquels sont associés les complétions G' et G'') de la manière suivante : si G' et G'' sont identiques, alors les pères de $N_{G''}$ deviennent des pères de $N_{G'}$ et $N_{G''}$ est supprimé de A.

Nous définissons la propagation de la manière suivante : soit $N_{G'}$ un gp-nœud de A. S'il n'y a aucun homomorphisme compatible de F^+ dans G', alors nous arrêtons tout le processus et nous retournons $G \nvdash F$ (G' est un contre-exemple). S'il y a un homomorphisme h de F dans G' (en pratique il suffit de voir qu'il n'y a aucun sommet relation manquant à G' par rapport à un homomorphisme compatible), nous marquons $N_{G'}$ comme *succès*. Si un nœud N est marqué comme *succès*, nous marquons toute sa descendance comme *succès*. Si tous les fils d'un *h-nœud H* sont marqués comme *succès* alors nous marquons H comme *succès* (nous vérifions la condition ET). Si un des fils d'un gp-nœud $N_{G'}$ est marqué comme *succès*, alors nous marquons tous les pères de $N_{G'}$ comme *succès* (nous vérifions la condition OU). Si la racine de A est marqué comme *succès* alors nous retournons $G \vdash F$. Le sous-algorithme `propagation` (cf. algorithme 13) fait cette propagation.

Finalement, nous obtenons l'algorithme `breadthCheck` (cf. algorithme 11) qui fait appel aux sous-algorithmes `expansion` et `propagation` (cf. algorithmes 12 et 13).

L'algorithme `breadthCheck` termine puisque le nombre de fils de chaque nœud est fini et qu'à chaque étape d'expansion nous augmentons de 1 la taille des complétions de l'étape précédente. Il y aura au plus $(2^{(n_G)^k \times |\mathcal{V}|}) - |G|$ étapes d'expansion, soit la taille d'une complétion totale de G par rapport au vocabulaire de complétion de F et G moins la taille initiale de G.

Nous illustrons le fonctionnement de l'algorithme `breadthCheck` et de ses deux sous-algorithmes `expansion` et `propagation` sur l'exemple 13.

Exemple 13. Nous reprenons l'exemple de la figure 4.10. Nous montrons que F se déduit de G. Soit A notre arbre de recherche avec G comme racine et comme seule feuille non marqué à la première étape d'expansion (pour plus de clarté, dans cet exemple nous donnons le même nom à un gp-nœud et à la complétion qui lui est associée). Nous considérons le sous-graphe $F^+ = r(x, y) \wedge r(z, y)$.

Algorithme 11: **breadthCheck**(F, G)

Données: les graphes F et G associés à deux formules de $FOL\{\exists, \wedge, \neg_a\}$ f et g de la forme : $f = p_1, \ldots, p_n, \neg n_1, \ldots, \neg n_m$ et $g = q_1, \ldots, q_l, \neg a_1, \ldots, \neg a_k$
Résultat: vrai si F se déduit de G, faux sinon
début
  Soit *fin*←*indéterminé* une variable d'arrêt ;
  Soit A un graphe d'expansion de racine G ;
  tant que *fin* = *indéterminé* **faire**
    fin← **expansion**(A) ;
  si *fin* = *succès* **alors**
    retourner *vrai ;*
  sinon
    retourner *faux ;*
fin

Étape 1.

(Expansion) Il y a trois homomorphismes compatibles $h_1, \ldots, h_3$ de F^+ dans G (cf. table 4.3). On construit alors trois nœuds H_1, H_2 et H_3 reliés à G respectivement par un arc étiqueté h_1, h_2 et h_3 (voir la figure 4.13). On construit alors les complétions $G_1 = G \cup \{+r(a,c)\}$, $G_2 = G \cup \{+r(b,a)\}$ et $G_3 = G \cup \{+r(c,c)\}$.
(Propagation) Rien.
(Fusion) Rien.

Étape 2.

(Expansion) Nous passons à l'étape d'expansion 2 avec les trois feuilles non marquées G_1, G_2 et G_3. Il y a deux homomorphismes compatibles de F^+ dans G_1 (h_2 et h_3, qui avaient été calculés à l'étape précédente dans G). On construit alors deux nœuds H_4 et H_5 reliés à G_1 respectivement par un arc étiqueté h_2 et h_3. On construit alors les gp-nœuds $G_4 = G_1 \cup \{+r(b,a)\}$ et $G_5 = G_1 \cup \{+r(c,c)\}$. Nous répétons le même processus pour G_2 et G_3.
(Propagation) Rien.
(Fusion) Nous comparons deux-à-deux les complétions obtenues et nous observons que $G_4 = G_6$, $G_5 = G_8$ et $G_7 = G_9$. Nous fusionnons donc les complétions équivalentes en G_4, G_5 et G_7, ce qui est illustré sur la figure 4.13 par des traits en pointillés et les gp-nœuds G_6, G_8 et G_9 barrés (pour indiquer qu'ils sont supprimés de l'arbre).

Algorithme 12: **expansion**(A)

Données: un graphe d'expansion A de racine G
Accès: un graphe polarisé F, un sous-graphe pur F^{max} de F maximal pour l'inclusion
Résultat: succès si le nœud racine est marquée comme *succès*, échec si un contre-exemple est trouvé, indéterminé sinon

début
  Soit *FE* l'ensemble des feuilles de A ;
  Soit *newF*$\leftarrow \emptyset$ un ensemble de nouvelles feuilles de A ;
  pour chaque $N_{G'} \in$ FE **faire**
    Soit $H = h_1, \ldots, h_r$ l'ensemble des homomorphismes compatibles de F^{max} dans G' ;
    si $H = \emptyset$ **alors**
      /* G' *est un contre-exemple* */
      retourner *échec ;*
    sinon
      pour chaque $h_i \in H$ **faire**
        Ajouter un h-nœud N_{H_j} comme fils de $N_{G'}$;
        Soit *RM* l'ensemble des sommets relations manquants à G par rapport à h_i ;
        si RM $= \emptyset$ **alors**
          /* h_i *est un homomorphisme de* F *dans* G' */
          Marquer N_{H_j} comme *succès* ;
          Soit N le père de N_{H_j} ;
          propagation(A, N) ;
          si G *est marqué* succès **alors**
            retourner succès ;
        sinon
          pour chaque $\sim r(t_1, \ldots, t_k) \in$ RM **faire**
            Soit $G'' = G' \cup \{\overline{\sim r}(h_i(t_1), \ldots, h_i(t_k))\}$ la complétion obtenue de G' en ajoutant le sommet relation $\overline{\sim r}(h_i(t_1), \ldots, h_i(t_k))$;
            Ajouter un gp-nœud $N_{G''}$ comme fils de N_{H_j} ;
            newF$\leftarrow$*newF*$\cup\{N_{G''}\}$;
  pour chaque $N_{G'} \in$ newF **faire**
    pour chaque $N_{G''} \in$ newF $\setminus N_{G'}$ **faire**
      si $G' \equiv G''$ **alors**
        fusionner $N_{G'}$ et $N_{G''}$;
        Supprimer $N_{G''}$ de *newF* et de A ;
  retourner *indéterminé ;*
fin

Algorithme 13: **propagation**(A,N)

Données: un graphe d'expansion A, un gp-nœud $N \in A$
Résultat: succès si le nœud racine est marquée comme *succès*, indéterminé sinon
début
  Marquer N comme *succès* ;
  Soient $N_{H_1}, \ldots, N_{H_n}$ les pères non marqués de N ;
  pour chaque $N_{H_i} \in N_{H_1}, \ldots, N_{H_n}$ **faire**
    si *tous les fils de* N_{H_i} *sont marqués* succès **alors**
      Marquer N_{H_i} comme *succès* ;
      Soit N_p le père de N_{H_i} ;
      propagation(A,N_p) ;
  pour chaque *fils* N_h *de* N *non marqué* **faire**
    Marquer N_h comme *succès* ;
    pour chaque *fils* N_{gp} *de* N_h *non marqué* **faire**
      propagation(A,N_{gp}) ;
  retourner *indéterminé ;*
fin

Étape 3.

(Expansion) Puis nous passons à l'étape d'expansion 3 et nous étendons les feuilles non marquées G_4, G_5 et G_7 (voir le détail sur la figure 4.13).

(Propagation) Il y a un homomorphisme compatible h_4 de F^+ dans G_5 qui peut être étendu à un homomorphisme de F dans G_5 car $-r(h_4(x), h_4(z)) \in G$ et $-r(h_4(y), h_4(z)) \in G$. Nous marquons alors H_{12} comme *succès* (le détail de la propagation est illustré par la figure 4.14 (un nœud marqué comme *succès* est entouré d'un ovale) ; cette étape correspond au point (0)) et nous propageons l'information : nous marquons comme *succès* G_5 car un de ses fils est marqué (1) ; nous marquons H_5 comme *succès* car tous ses fils sont marqués comme *succès* (2). Nous répétons ce processus en marquant comme *succès* G_1 (3), H_1 (4) et G (5). Comme la racine G est marqué comme *succès*, **nous concluons que** F **se déduit de** G **avec** $\{G_5, G_1^{neg}, G_5^{neg}\}$ **comme ensemble des complétions partielles de** G **couvrant les complétions totales de** G, où $G_1^{neg} = G \cup \{-r(a,c)\}$ et $G_5^{neg} = G_1 \cup \{-r(c,c)\}$ correspondent aux complétions des déductions implicites.

La figure 4.15 montre la comparaison entre un choix de recherche avec ou sans fusion, pour $nbTS = nbTC = 4$, $DS = 25\%$, $c = 0\%$, $nbE = 1$, $k = 2$ et $neg = 50\%$, en faisant varier DC en abscisse et en mesurant le temps de résolution. Comme on pouvait s'y attendre, le

h_1	$x \mapsto a, y \mapsto b, z \mapsto c$
h_2	$x \mapsto c, y \mapsto b, z \mapsto a$
h_3	$x \mapsto c, y \mapsto b, z \mapsto c$
h_4	$x \mapsto c, y \mapsto c, z \mapsto a$

TABLE 4.3 : les homomorphismes de l'exemple 13.

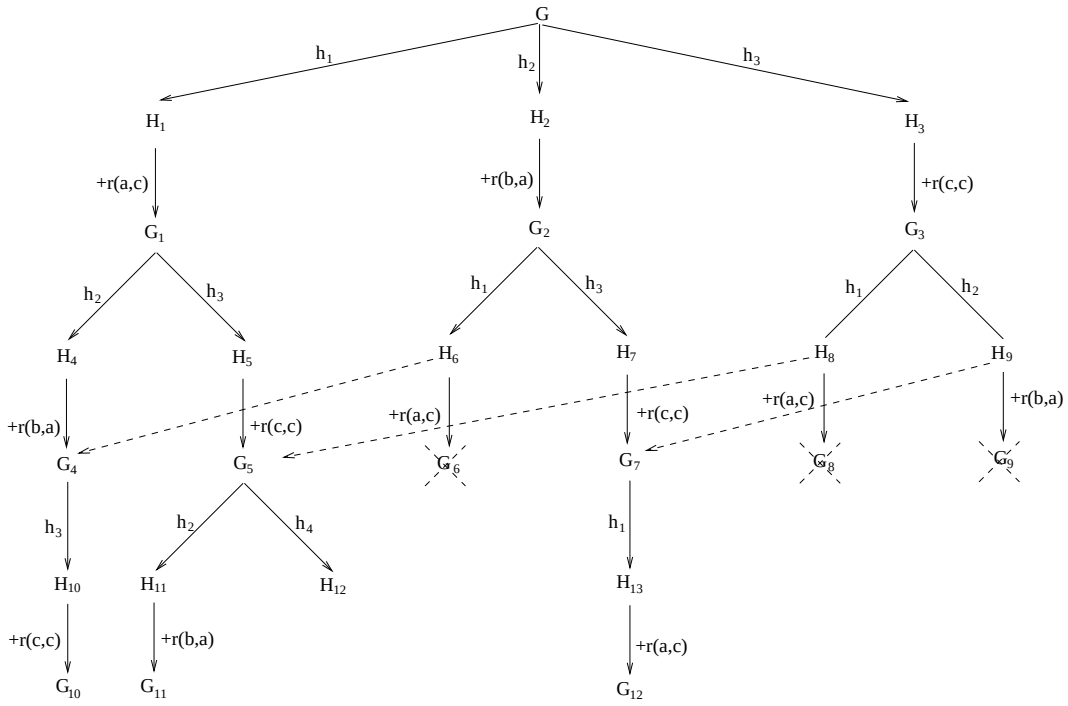


FIGURE 4.13 : le graphe d'expansion généré par `breathCheck` avant la propagation.

choix avec la fusion est toujours meilleur que le choix sans la fusion : par exemple pour *DC* = 25%, le choix avec la fusion est 4 fois meilleur que le choix sans la fusion (environ 0.8s contre 3s20).

Notons que pour une valeur de *nbTS* et *nbTC* supérieure à 4, l'utilisation de l'algorithme `breadthCheck` sans la fusion fait exploser l'espace mémoire.

L'algorithme `recAndCheck` :

L'algorithme `recAndCheck` est directement tiré du schéma d'algorithme `schémaPropWL` (cf. algorithme 9). Il parcours l'espace de recherche en profondeur, mais différemment de

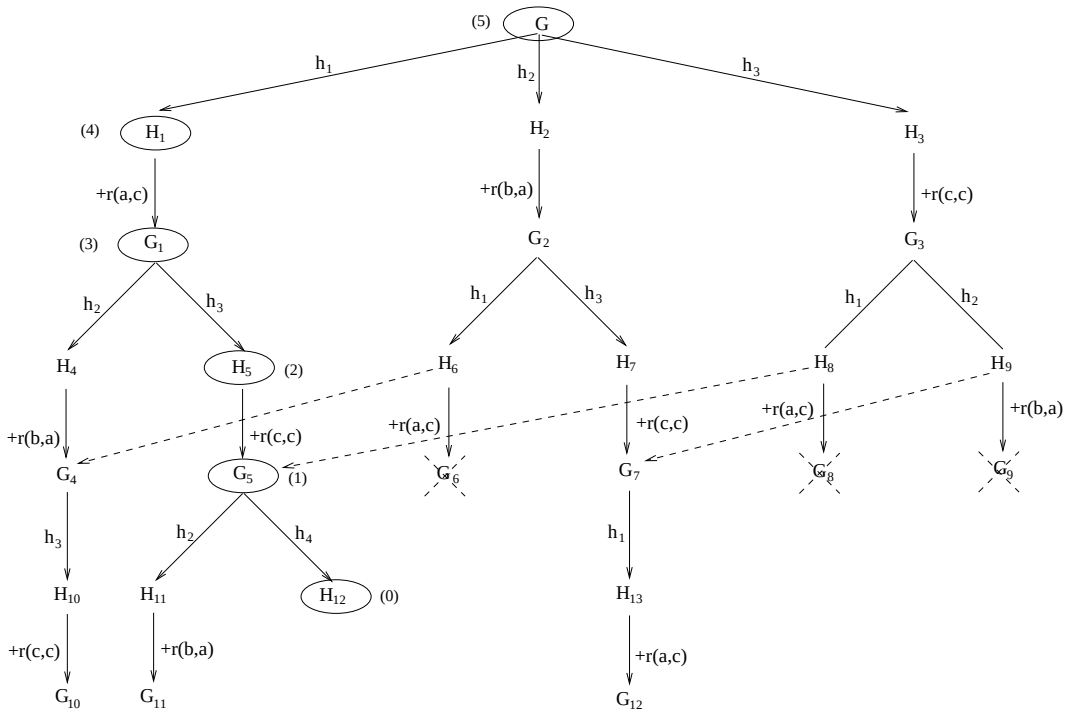


FIGURE 4.14 : le détail de la propagation sur le graphe d'expansion de la figure 4.13.

la méthode utilisée par `recCheck` (cf. discussion de la section 4.2.1). Notons que cet algorithme en est encore à ses débuts, et plusieurs pistes pour l'optimiser sont envisageables :

1. Comment assurer de ne pas parcourir deux fois les mêmes complétions ? À première vue il nous faut mémoriser toutes les complétions déjà explorées.
2. Quel homomorphisme compatible choisir en priorité ? Comment discriminer deux homomorphismes compatibles ? Notons que ce point est également intéressant pour `recCheckPlus`.
3. Quel sommet relation de $F \setminus F'$ manquant à G par rapport à h choisir en priorité ? Comment discriminer de tels sommets relations ? Notons que ce point est également intéressant pour `recCheckPlus`.

4.3 Algorithme UNSATCheck

Dans cette section, nous présentons une nouvelle méthode, qui consiste en une réécriture du problème *Déduction* dans le problème du test de la satisfiabilité d'une formule propositionnelle sous forme normale conjonctive (c'est-à-dire une conjonction de disjonctions de littéraux propositionnels), appelé le problème UNSAT.

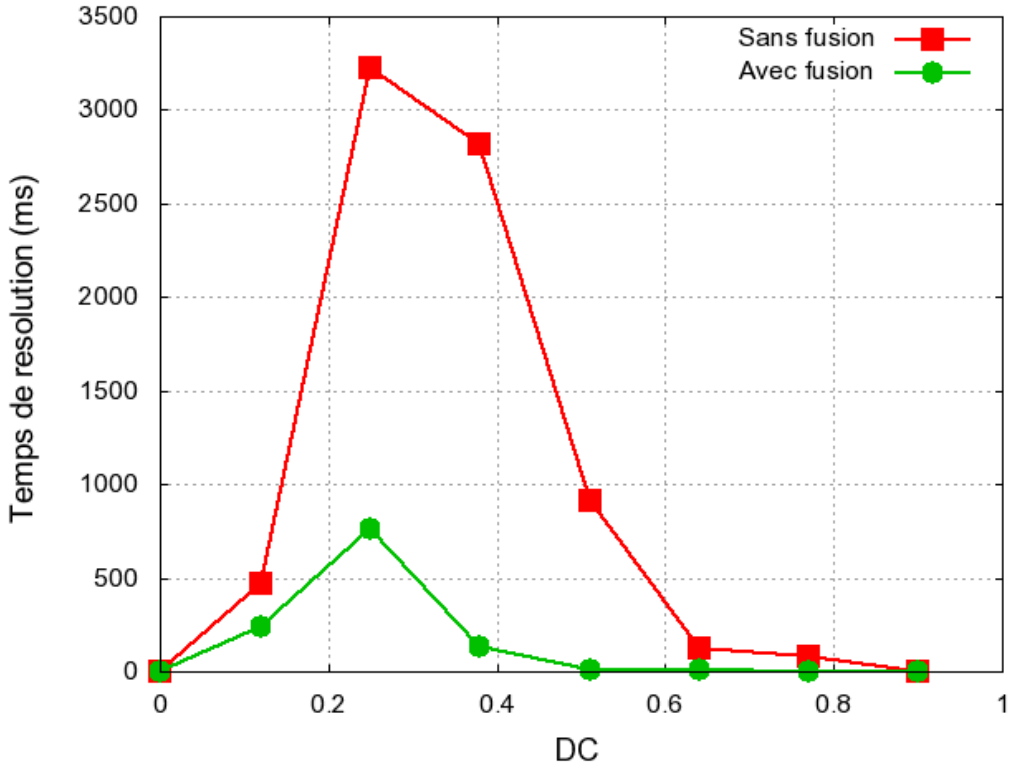


FIGURE 4.15 : influence de la fusion sur le temps de calcul.

Algorithme 14: **recAndCheck**(G)

Données: un graphe polarisé consistant G
Accès: un graphe polarisé F et un sous-graphe pur maximal pour l'inclusion F' de F
Résultat: vrai si F se déduit de G, faux sinon
début
  si *il y a un homomorphisme de F dans G* **alors retourner** *vrai* ;
  $h \leftarrow$ **recHomComp(F,F',G)** ;
  si h = *échec* **alors**
    / Il n'y a pas d'homomorphisme compatible de F' dans G */*
    retourner *faux* ;
  pour chaque *sommet relation* $\sim r(t_1,\ldots,t_k) \in F \setminus F'$ *manquant à G par rapport à h* **faire**
    si *NON*(***recAndCheck***$(G \cup \{\overline{\sim r}(t_1,\ldots,t_k)\})$) **alors retourner** *faux* ;
  retourner *vrai* ;
fin

4.3.1 La méthode

L'idée principale de cette méthode est la suivante : au lieu d'explorer l'espace des complétions par un parcours en profondeur ou en largeur afin de trouver un ensemble de complétions partielles couvrant l'ensemble des complétions totales, nous cherchons à construire un ensemble couvrant *candidat E* en une seule fois. L'arbre de recherche de la figure 4.16 illustre l'intuition de cette méthode : nous construisons un ensemble couvrant candidat E en une seule étape, soit ici $E = \{G_{1,1}, \ldots, G_{1,j}, \ldots, G_{n,1} \ldots, G_{n,k}\}$. Il restera alors à vérifier que E est bien un ensemble couvrant toutes les complétions totales de G.

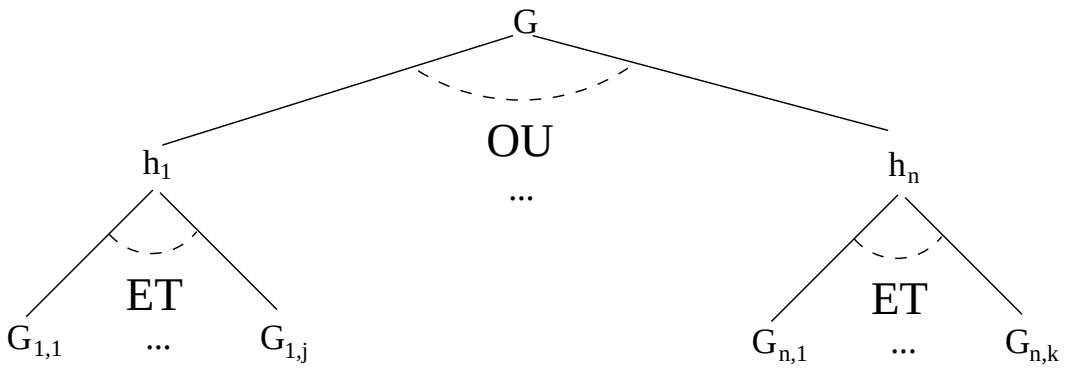


FIGURE 4.16 : arbre de recherche de la méthode étudiée.

Pour construire E, nous prenons en compte tous les homomorphismes compatibles d'un sous-graphe spécifique de F dans G. Puis nous construisons une formule f à partir de E, en considérant, pour chaque homomorphisme compatible h_i, l'ensemble des sommets relations qui doivent être ajoutés à G, soit SRM (pour « sommets relations manquants »), afin que h_i soit un homomorphisme de F dans $G \cup SRM$. Finalement, nous montrons que E est un ensemble couvrant l'ensemble des complétions totales de G si et seulement si f est une formule valide. Plus précisément, nous procédons en trois étapes :

1. Calculer tous les homomorphismes compatibles de F' (le sous-graphe spécial de F) dans G ;
2. Construire une formule propositionnelle F_{prop}, qui fait la disjonction de toutes les conjonctions de sommets relations manquants à G par rapport à chaque homomorphisme compatible calculé à l'étape 1 ;
3. Tester si $\overline{F_{prop}}$ (une instance du problème UNSAT) est insatisfiable : si elle est insatisfiable, alors F_{prop} est valide et $G \vdash F$, sinon $G \nvdash F$.

Étape 1 : calcul de l'ensemble couvrant candidat.

Notre méthode est basé sur un sous-graphe spécifique de F, qui se projette nécessairement dans chaque complétion totale de G quand $G \vdash F$. Nous verrons par la suite que ceci

n'est pas vrai pour tous les sous-graphes purs de F, mais vrai pour un graphe particulier que nous appelons sous-graphe stable :

Définition 4.7 (Sous-graphe stable) *Le* sous-graphe stable *de F, noté F^s, contient tous les sommets termes de F et tous les sommets relations de F dont la relation n'appartient pas au vocabulaire de complétion de F et G (notons que F^s est inclus dans tous les F^{max}).*

Puisque le sous-graphe stable ne contient pas de relations apparaissant dans les littéraux de complétion d'une complétion totale G^c, nous avons la propriété suivante (elle est illustrée par la figure 4.17) :

Propriété 4.2 *Soient deux graphes polarisés F et G. Si h est un homomorphisme de F dans une complétion totale de G, alors h est un homomorphisme compatible de F^s dans G.*

Démonstration : h est un homomorphisme de F dans G^c. h définit donc un homomorphisme de F^s dans G. Comme G est consistant, pour chaque sommet relation $\sim r(t_1,\ldots,t_k)$ de F qui n'appartient pas à F^s, il n'existe pas $\overline{\sim r}(h(t_1),\ldots,h(t_k))$ dans G, et G^c est consistant par construction, donc il n'existe pas de paire de sommets relations opposés $\sim r(c_1,\ldots,c_k)$ et $\overline{\sim r}(d_1,\ldots,d_k)$ de $F \setminus F^s$ tels que $(h(c_1),\ldots,h(c_k)) = (h(d_1),\ldots,h(d_k))$.

□

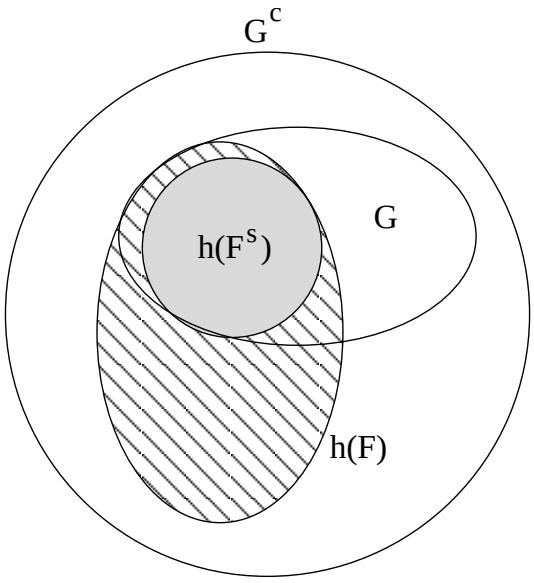


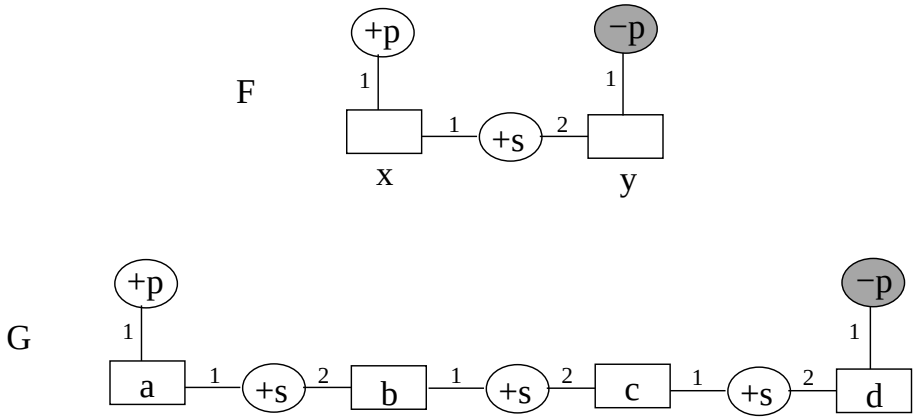
FIGURE 4.17 : illustration de la propriété 4.2

Cette propriété n'est plus vraie si l'on utilise un sous-graphe plus grand que le stable, comme le montre l'exemple 14 :

Exemple 14. Soient F et G les graphes de la figure 4.18. $F^s = s(x,y)$. Soient $F^1 = p(x) \wedge s(x,y)$ et $F^2 = s(x,y) \wedge \neg p(y)$ les deux sous-graphes purs de F incluant strictement F^s. Soit $G_1^c = p(a) \wedge s(a,b) \wedge p(b) \wedge s(b,c) \wedge \neg p(c) \wedge s(c,d) \wedge \neg p(d)$ une complétion totale de G (obtenue en ajoutant $p(b)$ et $\neg p(c)$). Il y a un homomorphisme de F dans G_1^c, $h = \{x \mapsto b, y \mapsto c\}$, mais ce n'est pas un homomorphisme que ce soit de F^1 dans G ou de F^2 dans G.

Ainsi le sous-graphe stable est de taille maximale pour la propriété 4.2.

Notons que nous pouvons remplacer, sans incidence sur le problème *Déduction*, chaque variable de G par une **nouvelle** constante. Cette modification préserve tous les homomorphismes dans G. Dans la suite, G contient uniquement des constantes. La figure 4.18 montre la formule g (et son graphe associé G) obtenue après la transformation de chaque variable de la formule $g = \exists t \exists u \exists v \exists w\ (p(t) \wedge s(t,u) \wedge s(u,v) \wedge s(v,w) \wedge \neg p(w))$ par une **nouvelle** constante $(t \mapsto a, u \mapsto b, v \mapsto c, w \mapsto d)$. Cette transformation est nécessaire pour l'obtention d'une formule propositionnelle à l'étape 2.



$$f = \exists x \exists y\ (p(x) \wedge s(x,y) \wedge \neg p(y))$$
$$g = p(a) \wedge s(a,b) \wedge s(b,c) \wedge s(c,d) \wedge \neg p(d).$$

FIGURE 4.18 : les graphes polarisés associés à f et g.

Exemple 15. Nous considérons les graphes polarisés F et G de la figure 4.18. Il y a trois homomorphismes compatibles de F^s dans G : $h_1 = \{x \mapsto a, y \mapsto b\}$, $h_2 = \{x \mapsto b, y \mapsto c\}$ et $h_3 = \{x \mapsto c, y \mapsto d\}$.

Étape 2 : construction de la formule propositionnelle.

Pour chaque homomorphisme compatible h calculé à l'étape 1, nous construisons une conjonction des sommets relations manquants à G par rapport à h. Chaque complétion

de G obtenue en ajoutant ces sommets relations à G est alors un élément de l'ensemble couvrant candidat.

Définition 4.8 (Conjonction minimale) *La* conjonction minimale *de G par rapport à un homomorphisme compatible h, notée C^m, est la conjonction composée de l'atome* $\blacksquare$ [5] *et des littéraux $\sim r(h(t_1),\ldots,h(t_k))$ tels que $\sim r(t_1,\ldots,t_k) \in (F \setminus F^s)$ et $\sim r(h(t_1),\ldots,h(t_k)) \notin G$.*

Exemple 16. Pour $h_1 : C_1^m = \neg p(b)$; pour $h_2 : C_2^m = p(b) \wedge \neg p(c)$; pour $h_3 : C_3^m = p(c)$.
Nous pouvons alors construire la formule entière, qui est la disjonction de toutes les conjonctions minimales :

Définition 4.9 (Disjonction des Conjonctions Minimales du Stable (DCMS)**)** *On appelle* DCMS(F,G) *la disjonction de l'atome* $\square$ [6] *et des conjonctions minimales de tous les homomorphismes compatibles de F^s dans G (c'est-à-dire $\square \vee C_1^m \vee \cdots \vee C_n^m$, noté $\bigvee C^m$).*

Exemple 17. DCMS$(F,G) = \neg p(b) \vee (p(b) \wedge \neg p(c)) \vee p(c)$.
Le théorème validant cette approche est le suivant (avec G consistant) :

Théorème 4.10 *$G \vdash F$ si et seulement si* DCMS(F,G) *est valide.*

Les définitions et lemmes qui suivent sont utilisés pour prouver le théorème.

Définition 4.11 (Conjonction totale) *Soit G^c une complétion totale de G. La* conjonction totale *de G^c, notée C, est la conjonction de tous les littéraux $\sim r(t_1,\ldots,t_k) \in (G^c \setminus G)$.*

Définition 4.12 (Disjonction des Conjonctions Totales (DCT)**)** *On appelle* DCT(G) *la disjonction de toutes les conjonctions totales de G (c'est-à-dire $C_1 \vee \cdots \vee C_j$ pour j complétions totales de G).*

Lemme 4.3 DCT(G) *est valide.*

Démonstration : immédiate par construction de l'arborescence de complétion.

□

Lemme 4.4 *Si $G \vdash F$ alors quelque soit $C \in$ DCT(G), il existe $C^m \in$ DCMS(F,G) tel que $C^m \subseteq C$ (c'est-à-dire que tous les sommets relations de C^m sont aussi dans C).*

5. $\blacksquare$ représente la tautologie : il est nécessaire lorsque h est un homomorphisme de F dans G, sinon la conjonction est vide.

6. $\square$ représente l'absurde : il est nécessaire lorsqu'il n'y a pas d'homomorphisme compatible de F^s dans G, sinon la disjonction est vide.

Démonstration : soit C une conjonction totale apparaissant dans DCT(G). Comme $G \vdash F$, il existe un homomorphisme h de F dans $G \wedge C$ (cf. théorème 2.19). D'après la propriété 4.2, h est un homomorphisme compatible de F^s dans G. Par définition d'une conjonction minimale, h est un homomorphisme de F dans la complétion $G \wedge C^m$ (où C^m est la conjonction minimale de G issue de h) et C^m ne contient que des littéraux $\sim r(t_1,\ldots,t_k)$ tels que $\sim r(t_1,\ldots,t_k) = \sim r(h(t_1),\ldots,(t_k))$, donc $\sim r(t_1,\ldots,t_k)$ est aussi un littéral de C.

□

Propriété 4.5 *Soit C^m une conjonction minimale de G issue d'un homomorphisme compatible de F^s dans G. $G \wedge C^m \vdash F$.*

Démonstration : immédiate d'après la définition d'une conjonction minimale (cf. définition 4.8).

□

Démonstration du théorème 4.10 : $\Longleftarrow$ Puisque DCMS(F,G) est valide, $G \equiv G \wedge \bigvee C^m \equiv \bigvee (G \wedge C^m)$. D'après la propriété 4.5, on a $\bigvee (G \wedge C^m) \vdash F$. Donc $G \vdash F$.

$\Longrightarrow$ Soit $G \vdash F$. D'après le lemme 4.4, quelque soit $C \in$ DCT(G), il existe $C^m \in$ DCMS(F,G) tel que $C = C^m \wedge C'$ où C' est une conjonction de littéraux. *Par l'absurde* : supposons que DCMS(F,G) soit non valide. Alors il existe une interprétation $\mathcal{I}$ telle que pour tout $C^m \in$ DCMS(F,G), $\mathcal{I} \vdash \neg C^m$. Donc quelque soit $C \in$ DCT(G), $\mathcal{I} \vdash \neg C$. Donc DCT($G$) n'est pas valide, ce qui contredit le lemme 4.3. On en conclut que DCMS(F,G) est valide.

□

Étape 3 : passage à SAT/UNSAT.

La négation de DCMS(F,G) est une forme normale conjonctive propositionnelle, ce qui nous permet d'utiliser un solveur SAT.

Exemple 18. $FC = \overline{\text{DCMS}(F,G)} = p(b) \wedge (\neg p(b) \vee p(c)) \wedge \neg p(c)$. FC est insatisfiable, donc DCMS(F,G) est valide, d'où $G \vdash F$.

L'algorithme `UNSATCheck`

L'algorithme `UNSATCheck` (cf. algorithme 15) décrit cette méthode. Notons que notre implémentation de cet algorithme effectue le test d'insatisfiabilité (appelé **UNSAT**) en faisant appel au solveur *Sat4J*[7].

7. http ://www.sat4j.org/

Algorithme 15: **UNSATCheck**(G)

Données: un graphe polarisé consistant G
Accès: F, F^s
Résultat: vrai si $G \vdash F$, faux sinon
début
  DCMS$(F,G) \leftarrow \square$;
  $h_1, \ldots, h_n \leftarrow$ **tousHomomorphismesCompatibles**(F, F^s, G) ;
  pour chaque $h_i, i = 1 \ldots n$ **faire**
    pour chaque *sommet relation frontière* $\sim r(t_1, \ldots, t_k) \in F \setminus F^s$ **faire**
      $C_i^m \leftarrow \blacksquare$;
      si $\sim r(h_i(t_1), \ldots, h_i(t_k) \notin G$ **alors**
        $C_i^m \leftarrow C_i^m \wedge \sim r(h_i(t_1), \ldots, h_i(t_k))$;
    DCMS$(F,G) \leftarrow$ DCMS$(F,G) \vee C_i^m$;
  retourner ***UNSAT***$(\overline{\text{DCMS}(F,G)})$
fin

4.3.2 Taille de la formule propositionnelle obtenue

Avec cette méthode, nous passons du problème *Déduction* qui est Π_2^P*-complet* au problème *UNSAT* qui est *co-NP-complet* ; toutefois ce deuxième problème prend en entrée une formule dont la taille peut être exponentielle en la taille des données de départ. Nous bornons plus précisément ci-dessous la taille de la formule propositionnelle obtenue.

Soit nb_{hom} le nombre d'homomorphismes compatibles de F^s dans G. Il représente le nombre de conjonctions minimales obtenues dans la formule propositionnelle finale. Il est borné par $n_G^{n_F}$, où n_G et n_F sont respectivement le nombre de termes dans G et F. Néanmoins en pratique, nous nous attendons à ce que plus la taille de F^s soit grande, plus le nombre d'homomorphismes compatibles de F^s dans G soit faible, puisque chaque sommet relation de F^s représente potentiellement une contrainte en plus lors de la recherche d'un homomorphisme compatible de F^s dans G.

La taille d'une conjonction minimale est quant à elle bornée par $|F \setminus F^s|$.

La taille de la formule propositionnelle obtenue est donc bornée par $nb_{hom} * |F \setminus F^s|$.

D'une part, plus la taille de F^s est grande, plus la taille d'une conjonction minimale est petite, d'autre part on s'attend à ce que nb_{hom} diminue avec la croissance de F^s : ces deux points laissent penser que l'efficacité de cette méthode sera directement corrélée à la taille du sous-graphe stable de F.

Il est également à noter que nb_{hom} peut être borné plus précisément : ce n'est pas le nombre d'homomorphismes compatibles de F^s dans G qui importe véritablement, mais plutôt le nombre de substitutions différentes sur les termes apparaissant dans $L = F \setminus F^s$ que l'on obtient par ces homomorphismes compatibles. En effet, deux homomorphismes

compatibles de F^s dans G différents h_1 et h_2 mais tels que $h_1(L)) = h_2(L)$ amèneront la création de la même conjonction minimale.

4.4 Comparaison des algorithmes

Dans cette partie, nous comparons expérimentalement les différents algorithmes de résolution du problème *Déduction*, à savoir `recCheckPlus`, `breadthCheck`, `recAndCheck` et `UNSATCheck`. Notons que nous avons implémenté tous les algorithmes puisque même l'algorithme proposé par Wei et Lausen [2003] n'avait pas été implémenté.

La figure 4.19 montre les résultats de la comparaison des algorithmes `recCheckPlus`, `recAndCheck`, `breadthCheck` et `UNSATCheck` pour $nbTS = nbTC = 5$, $c = 0\%$, $nbE = 1$, $k = 2$, $DS = 24\%$ et $neg = 50\%$. Nous observons, pour ces faibles valeurs de paramètres, que l'algorithme `breadthCheck` est déjà très mauvais par rapport aux deux autres algorithmes. Nous avons vu dans la section 4.2.3 que l'algorithme `breadthCheck` sans la fusion faisait exploser l'espace mémoire. Néanmoins la fusion semble entraîner un surcoût en temps très important. Ainsi l'algorithme `breadthCheck` doit encore être optimisé. Dans la suite nous ne comparons que les algorithmes `recCheckPlus`, `recAndCheck` et `UNSATCheck`.

Nous comparons alors `recCheckPlus` et `recAndCheck`, qui utilisent tous deux un parcours en profondeur de l'espace de recherche mais de manière différente (notons qu'actuellement `recAndCheck` peut parcourir plusieurs fois la même complétion). La figure 4.20 illustre les résultats obtenus pour $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$ et $neg = 50\%$. Nous nous apercevons que `recCheckPlus` est toujours meilleur que `recAndCheck`.

Nous comparons alors `UNSATCheck` et `recCheckPlus`. Nous avons utilisé la taille du vocabulaire de complétion (directement corrélée avec la taille du sous-graphe stable) comme paramètre variable, notée $t_{\mathcal{V}} = |\mathcal{V}|$. En effet, ce paramètre est crucial pour l'algorithme `UNSATCheck` : plus le sous-graphe stable est grand, plus le nombre d'homomorphismes compatibles calculés est petit et donc plus la taille de la formule obtenue est petite. Notons que ce paramètre a aussi une influence sur l'algorithme `recCheckPlus` : plus la taille du vocabulaire de complétion est petite, plus le sous-graphe pur de cardinalité maximale est grand.

Nous construisons 1000 instances aléatoires pour chaque valeur de $t_{\mathcal{V}}$ (pour chaque relation, le pourcentage de négation est égal à 0%, 50% ou 100%) et comparons `UNSATCheck` et `recCheckPlus` avec ces instances. Notons que nous choisissons $nbE = 3$ afin d'avoir quatre valeurs possibles pour $t_{\mathcal{V}}$ (0, 1, 2 et 3). La figure 4.21 montre les résultats obtenus et la table 4.4 précise ces résultats [8].

Nous observons que pour les deux algorithmes la difficulté maximale survient pour $|\mathcal{V}| = 3$. Comme on pouvait s'y attendre, `UNSATCheck` est le plus mauvais des deux car le

8. Nous faisons remarquer que $nbTS = nbTC = 8$ est la valeur maximale avec laquelle l'algorithme `UNSATCheck` peut fonctionner (avec $t_{\mathcal{V}} = 3$ et notre implémentation) : au-delà de 8, l'espace mémoire explose.

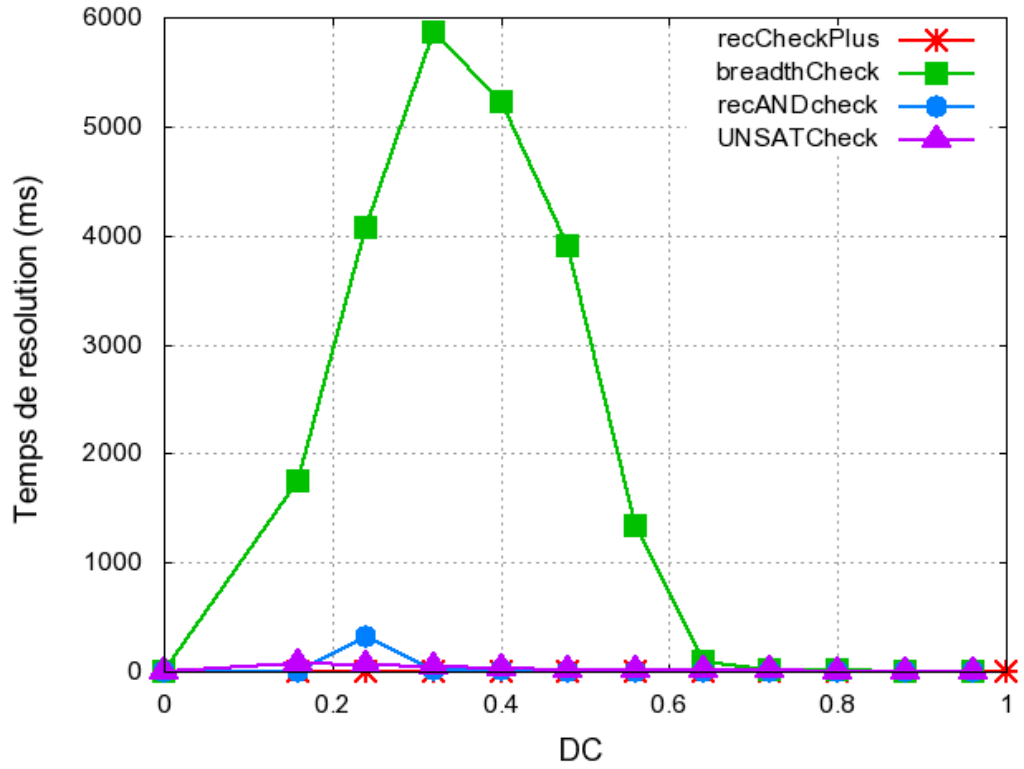


FIGURE 4.19 : comparaison des quatre algorithmes.

$\|\mathcal{V}\|$	$\|F^s\|$	Temps de `recCheckPlus` (ms)	Temps de `UNSATCheck` (ms)			
			Total	Étape 1	Étape 2	Étape 3
3	0	6482	27038	10541	15378	1119
2	6	800	444	172	229	43
1	12	5	20	9	2	9
0	18	1	2	2	0	0

TABLE 4.4 : comparaison détaillée des algorithmes `recCheckPlus` et `UNSATCheck` : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 3$, k=2, $DS = 6\%$, $DC = 51\%$ et $neg = 50\%$.

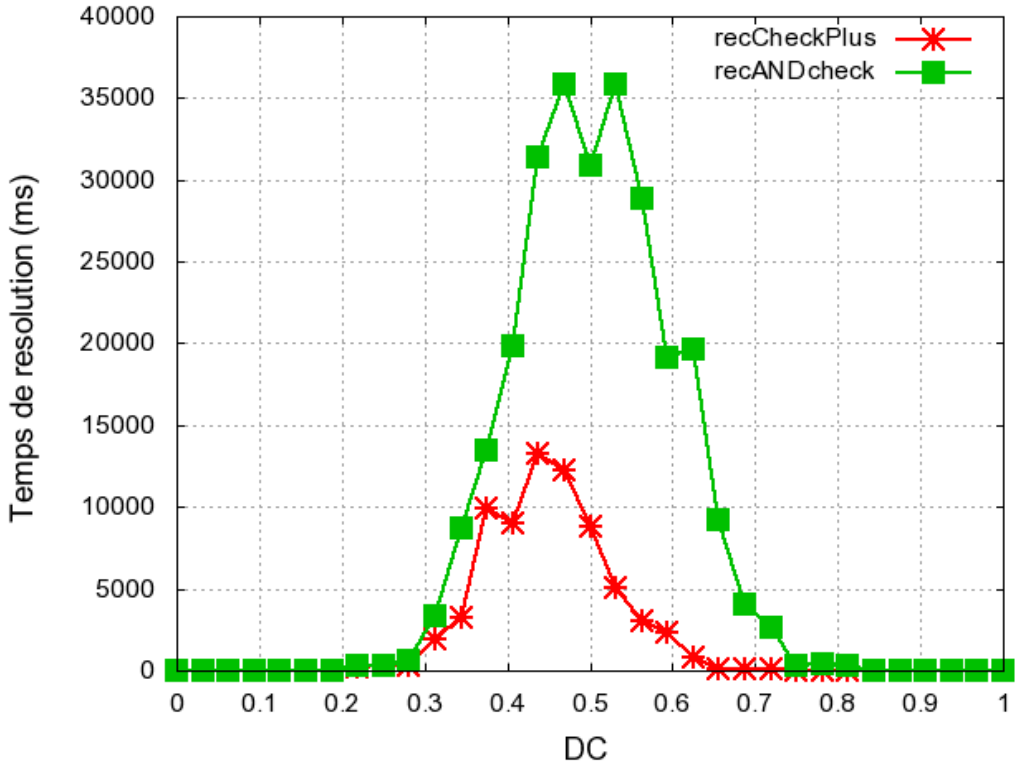


FIGURE 4.20 : comparaison des algorithmes `recCheckPlus` et `recAndCheck`.

sous-graphe stable contient uniquement des sommets termes, ainsi le nombre d'homomorphismes compatibles et donc la taille de la formule obtenue seront exponentiels en la taille des instances de départ. L'augmentation de la taille du sous-graphe stable rend l'algorithme meilleur : pour $|\mathcal{V}| = 2$, `UNSATCheck` est légèrement meilleur que `recCheckPlus`, et quand la taille du sous-graphe stable augmente, les deux algorithmes obtiennent des résultats similaires. Ces premiers résultats montrent que le choix d'un algorithme plutôt que l'autre va dépendre de la taille du sous-graphe stable.

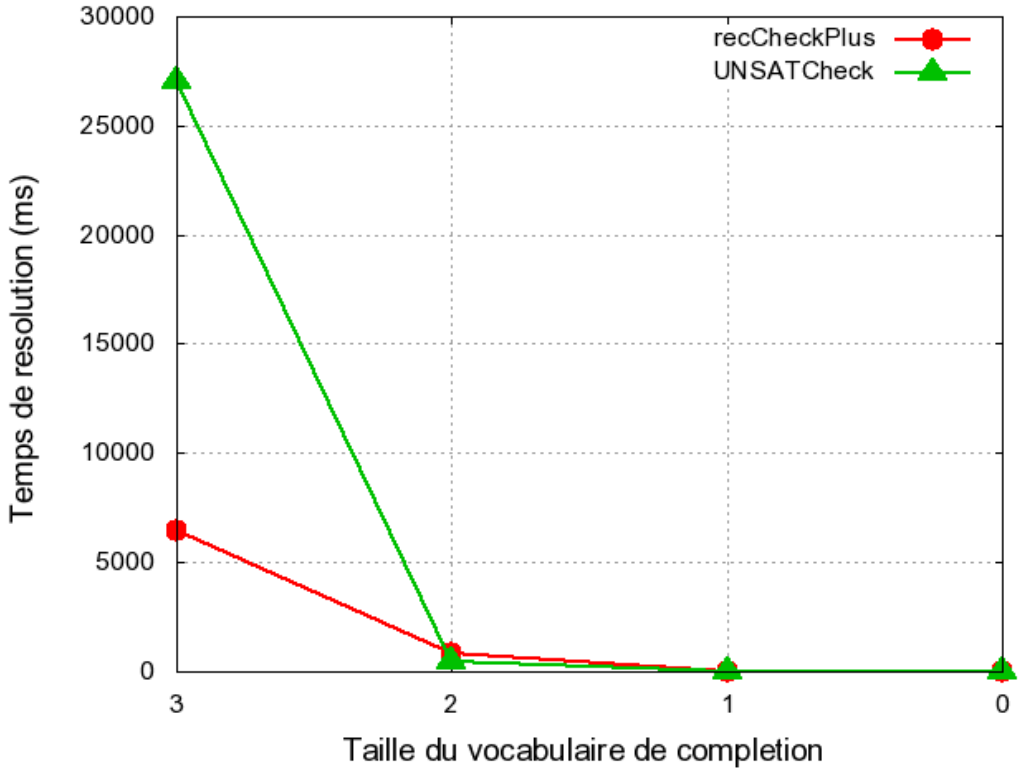


FIGURE 4.21 : comparaison des algorithmes `UNSATCheck` et `recCheckPlus` : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 3$, $k = 2$, $DS = 6\%$, $DC = 51\%$ et $neg = 50\%$.

CHAPITRE

5 Extensions

Préambule

Dans ce chapitre, nous étudions trois extensions du cadre précédent qui concernent la prise en compte des constantes, les réponses à une requête non-booléenne et la prise en compte d'une ontologie légère. Pour chacune d'elles, nous détaillons les modifications qu'elles induisent dans les algorithmes présentés au chapitre 4.

Sommaire

5.1 Prise en compte des constantes 97
5.2 Réponse à une requête non-booléenne 105
5.3 Prise en compte d'une ontologie légère 109

5.1 Prise en compte des constantes

Nous avons vu au chapitre 3, section 3.6.3, que les instances les plus difficiles sont caractérisées par un pourcentage de sommets constantes de 0%. Néanmoins en pratique, les requêtes formulées par un utilisateur sont généralement composées de constantes. Le but de cette section est de caractériser l'influence des sommets constantes sur la résolution du problème *Déduction*.

Les sommets constantes jouent un rôle particulier : comme un sommet constante t étiqueté A dans F apparaît une seule fois dans G (soit t' étiqueté A), il contraint les homomorphismes de F dans G. Pour tout homomorphisme h de F (ou d'un sous-graphe de F) dans G, l'image de t sera toujours t'. De plus, un sommet relation $\sim r(t_1, \ldots, t_k)$ de F tel que

$t_1,\ldots,t_k$ sont des sommets constantes aura la même image dans G pour tout homomorphisme de F dans G. Plus précisément, nous avons les définitions et propriétés suivantes :

Définition 5.1 *[Sommet relation 0-libre, Sous-graphe 0-libre] Un sommet relation* $\sim r(t_1,\ldots,t_k)$ *est dit* 0-libre *si quelque soit* $i = 1\ldots k$, t_i *est un sommet constante. Un sous-graphe de* F *est dit 0-libre, noté* F^0*, s'il ne contient que des sommets relations 0-libres.*

Propriété 5.1 *Soient* F *et* G *deux graphes polarisés et* $\sim r(t_1,\ldots,t_k)$ *un sommet relation 0-libre dans* F*. Pour tous les homomorphismes* $h_1,\ldots,h_n$ *de* F *dans des complétions de* G*,* $\sim r(h_i(t_1),\ldots,h_i(t_k))$ *correspond au même sommet relation dans* G*.*

Démonstration : $\sim r(t_1,\ldots,t_k)$ est un sommet relation 0-libre dans F, donc par définition $t_1,\ldots,t_k$ sont des sommets constantes. Soit $h_1,\ldots,h_n$ l'ensemble des homomorphismes de F dans les complétions de G. Comme un sommet constante t_l étiqueté A_l n'apparaît qu'une seule fois dans F et G, $h_i(t_l) = \cdots = h_j(t_l)$ quelque soit $i = 1\ldots n$, $j = 1\ldots n$, $l = 1\ldots k$ et $i \neq j$. On en conclut que quelque soit $i = 1\ldots n$, $\sim r(h_i(t_1),\ldots,h_i(t_k))$ correspond au même sommet relation dans G.

□

Nous rappelons la définition de sommets relations échangeables introduite au chapitre 2 :

Définition 5.2 *[Sommets relations échangeables] Deux sommets relations* $\sim r(c_1,\ldots,c_k)$ *et* $\overline{\sim r}(d_1,\ldots,d_k)$ *de* F *sont dits* échangeables *par rapport à* G *s'il existe deux complétions totales de* G*,* G_1 *et* G_2*, et deux homomorphismes* h_1 *et* h_2*, respectivement de* F *dans* G_1 *et de* F *dans* G_2 *tels que* $(h_1(c_1),\ldots,h_1(c_k)) = (h_2(d_1),\ldots,h_2(d_k))$*.*

Lemme 5.2 *Un sommet relation 0-libre n'est pas échangeable.*

Démonstration : soient $\sim r(c_1,\ldots,c_k)$ et $\overline{\sim r}(d_1,\ldots,d_k)$ deux sommet relations opposés de F tels que $(c_1,\ldots,c_k)$ sont des sommets constantes étiquetés $A_1,\ldots,A_k$. D'après la propriété 5.1, tous les homomorphismes de F dans une complétion totale de G donnent la même image à $\sim r(c_1,\ldots,c_k)$. Donc toute complétion totale de G contient $\sim r(c'_1,\ldots,c'_k)$ tel que $(c'_1,\ldots,c'_k)$ sont des sommets constantes étiquetés $A_1,\ldots,A_k$. Donc aucune complétion totale ne peut contenir également $\overline{\sim r}(c'_1,\ldots,c'_k)$.

□

Nous obtenons finalement le théorème 5.3 :

Théorème 5.3 *Soient* F *et* G *deux graphes polarisés et* F^0 *un sous-graphe 0-libre de* F*. S'il n'y a pas d'homomorphisme compatible de* F^0 *dans* G *alors* $G \not\vdash F$*.*

Démonstration : d'après le lemme 5.2, F^0 ne contient pas de sommets relations échangeables. Or d'après le théorème 2.24, s'il n'y a pas d'homomorphisme compatible d'un sous-graphe sans sommets relations échangeables de F dans G alors $G \not\vdash F$.

□

De plus, nous pouvons restreindre le vocabulaire de complétion de F et G en prenant en compte les sommets relations 0-libres. Nous nous appuyons sur le théorème suivant :

Théorème 5.4 *[Leclère et Mugnier, 2008] $G \vdash F$ si et seulement s'il y a un homomorphisme de F dans chaque complétion totale de G par rapport à un vocabulaire de complétion composé des relations apparaissant dans des sommets relations échangeables de F par rapport à G.*

Ainsi le vocabulaire de complétion peut se limiter aux relations apparaissant dans des sommets relations échangeables de F par rapport à G. Nous en déduisons la propriété 5.3 :

Propriété 5.3 *Si r est une relation n'apparaissant pas dans des sommets relations opposés non 0-libres de F, alors r n'est pas nécessaire pour compléter G (c'est-à-dire $G \vdash F$ si et seulement s'il y a un homomorphisme de F dans chaque complétion totale de G obtenue sans considérer r).*

Démonstration : nous concluons directement avec le lemme 5.2.

□

Nous affinons alors la définition 2.20 du vocabulaire de complétion en prenant en compte les sommets relations 0-libres :

Définition 5.5 (Vocabulaire de complétion) *Soient deux graphes polarisés F et G, le* vocabulaire de complétion *de F et G est composé de toutes les relations apparaissant dans des sommets relations opposés et non 0-libres, à la fois dans F et dans G.*

Dans les différents algorithmes présentés (`recCheckPlus`, `breadthCheck` et `UNSAT-Check`), on peut prendre en compte les sommets relations 0-libres en les ajoutant respectivement au sous-graphe de guidage pour le premier, au sous-graphe positif pour le second et au sous-graphe stable pour le dernier.

Nous avons ensuite essayé d'étendre la propriété 5.1 à des sommets relations composés d'un seul sommet terme voisin variable, que nous appelons sommets relations 1-libres :

Définition 5.6 (Sommet relation 1-libre, Sous-graphe 1-libre) *Un sommet relation $\sim r(t_1,\ldots,t_k)$ est dit* 1-libre *si l'un de ses sommets termes voisins est variable (il peut apparaître plusieurs fois dans $t_1,\ldots,t_k$) et les autres sont des sommets constantes. Un sous-graphe est dit* 1-libre*, noté F^1, s'il ne contient que des sommets relations 1-libres ou 0-libres.*

Néanmoins la propriété 5.1 ne s'étend pas aux sommets relations 1-libres comme le montre la figure 5.1 : soient F et G deux graphes polarisés et G' et G'' les complétions partielles obtenues de G en ajoutant respectivement $+r(u,v)$ et $-r(u,v)$. Il existe un homomorphisme de F dans G', soit $h_1 = \{A \mapsto A, x \mapsto u, y \mapsto v, z \mapsto w\}$, et un homomorphisme de F dans G'', soit $h_2 = \{A \mapsto A, x \mapsto t, y \mapsto u, z \mapsto v\}$. Le sommet relation $+s(A,x)$ est 1-libre mais ses images par h_1 et h_2, respectivement $+s(A,u)$ et $+s(A,t)$, sont différentes.

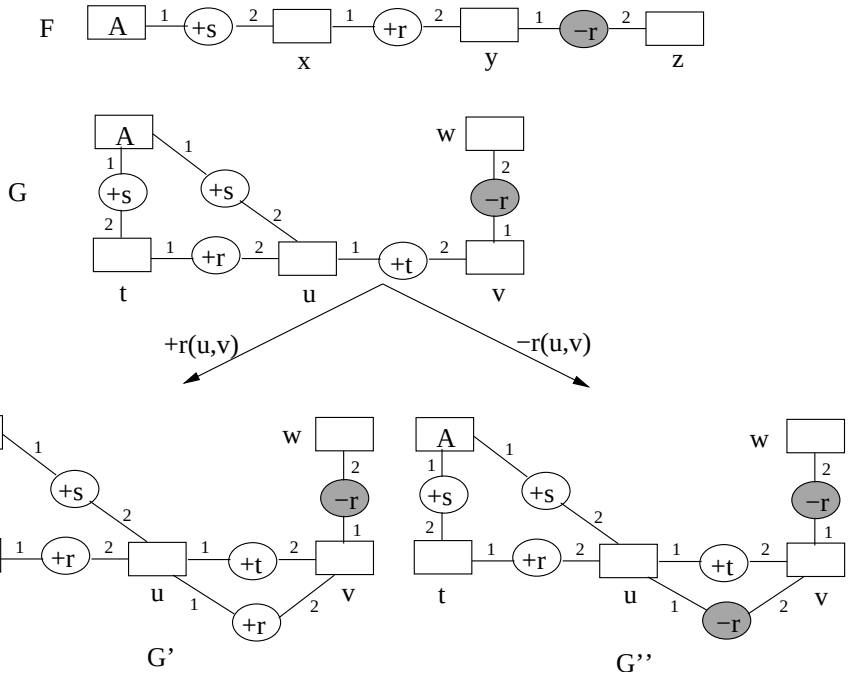


FIGURE 5.1 : non-extension de la propriété 5.1 à des sommets relations 1-libres.

Dans la suite, nous cherchons à définir des sous-graphes de F prenant en compte les constantes tels que s'il n'y a pas d'homomorphisme d'un de ces sous-graphes dans G, alors $G \not\vdash F$. À première vue, il semblerait qu'un sous-graphe respectant cette condition soit le sous-graphe 1-libre de F maximal pour l'inclusion. Néanmoins, l'exemple 19 montre qu'être 1-libre n'est pas une condition suffisante :

Exemple 19. Soient F et G les graphes polarisés de la figure 5.2. F est 1-libre et il n'y a pas d'homomorphisme de F dans G alors que $G \vdash F$. Il suffit de compléter G par $\sim r(A,B)$ pour s'en assurer : il y a un homomorphisme $h_1 = \{A \mapsto A, x \mapsto B, B \mapsto B\}$ de F dans G' (obtenu de G par l'ajout de $+r(A,B)$) et un homomorphisme $h_2 = \{A \mapsto A, x \mapsto A, B \mapsto B\}$ de F dans G'' (obtenu de G par l'ajout de $-r(A,B)$). Donc $G \vdash F$.

Nous définissons alors une notion plus forte qui est celle des sous-graphes *cloués* :

Définition 5.7 (Sous-graphe cloué) *Un sous-graphe est dit* cloué *si :*

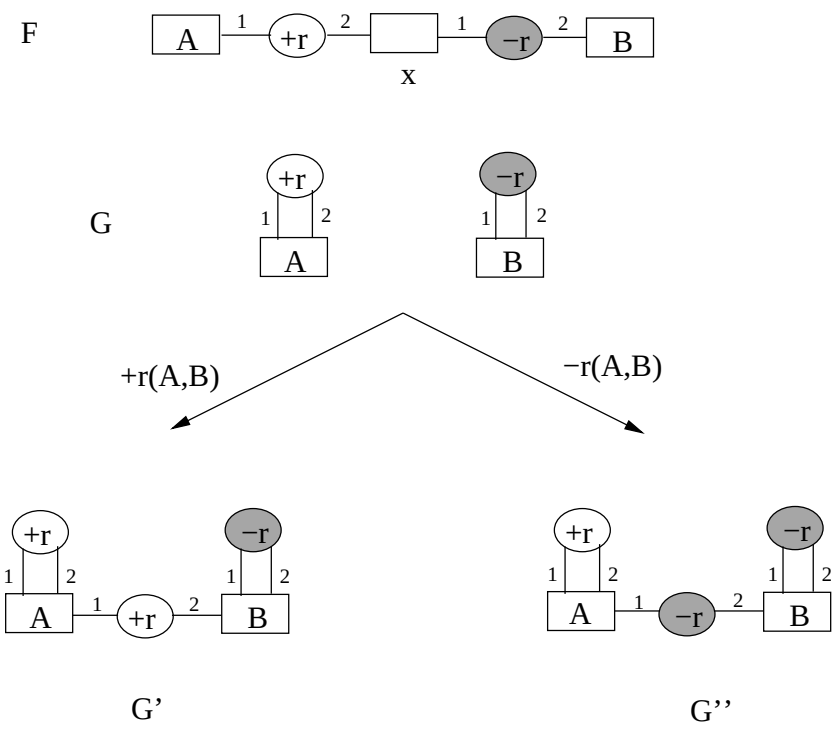


FIGURE 5.2 : non-extension de la propriété 5.3 à des sommets relations 1-libres.

1. *Il est 1-libre.*
2. *Il ne contient pas deux sommets relations opposés, soient* $l_1 = +r(t_1,\ldots,t_k)$ *et* $l_2 = -r(t'_1,\ldots,t'_k)$ *avec* $k \geq 2$*, tels que :*
   - **(a)** l_1 *et* l_2 *ont comme voisin commun un sommet variable, soit* x*.*
   - **(b)** l_1 *a comme voisin un sommet constante A (étiqueté A) et* l_2 *a comme voisin un sommet constante B (étiqueté B) avec* $A \neq B$*.*
   - **(c)** *Pour chaque* $i = 1\ldots k$*, soit* t_i *et* t'_i *sont le même sommet constante, soit* $t_i = x$ *et alors* $t'_i = B$*, soit* $t'_i = x$ *et alors* $t_i = A$*.*

Le graphe F de la figure 5.2 n'est pas cloué car il ne satisfait pas la condition 2 de la définition 5.7. En effet, $+r(A,x)$ et $-r(x,B)$ violent cette condition et on s'aperçoit que ces deux sommets relations sont échangeables par rapport à G. Notons que si l'on considérait $+r(x,A)$ au lieu de $+r(A,x)$, les sommets ne seraient plus échangeables. Nous allons voir que les sommets relations opposés de la condition 2 sont le seul cas d'échangeabilité possible pour un graphe 1-libre. Nous obtiendrons finalement le théorème 5.8 :

Théorème 5.8 *Soient F et G deux graphes polarisés (avec G consistant). Soit* $F^{\text{cloué}}$ *un sous-graphe cloué de F. S'il n'existe pas d'homomorphisme compatible de* $F^{\text{cloué}}$ *dans G, alors* $G \not\vdash F$.

Les définitions et propriétés qui suivent sont utilisées pour prouver le théorème (nous allons montrer que $F^{cloué}$ ne possède pas de sommets relations échangeables).

Définition 5.9 (Graphe éclaté) *Soit F un graphe quelconque. On appelle* graphe éclaté *de F, noté* $F^{\text{é}}$*, le graphe obtenu de F en éclatant chaque sommet constante c étiqueté A de F ayant* $n \geq 2$ *voisins* $r_1, \ldots, r_n$ *en n sommets, c'est-à-dire en remplaçant c par n sommets constantes* $c_1, \ldots, c_n$ *étiquetés A et toute arête* (r_i, j, c) *par* (r_i, j, c_i) *pour* $i = 1 \ldots n$.

Notons que contrairement aux graphes considérés précédemment, on peut avoir plusieurs sommets étiquetés par la même constante. Autrement dit, $F^{\text{é}}$ n'est pas sous forme normale. Cependant, pour la conservation des homomorphismes, l'important est que G soit sous forme normale. La propriété suivante est immédiate :

Propriété 5.4 *Soient F et G deux graphes et* $F^{\text{é}}$ *le graphe éclaté de F. S'il y a un homomorphisme h de F dans G alors il y a un homomorphisme* h' *de* $F^{\text{é}}$ *dans G qui s'obtient de la façon suivante :* $h'(c_i) = h(c)$ *pour tout sommet* c_i *résultant de l'éclatement de c et* $h'(c) = h(c)$ *pour tout autre sommet c. Et réciproquement, si G est sous forme normale, tout homomorphisme de* $F^{\text{é}}$ *dans G fournit un homomorphisme de F dans G.*

Propriété 5.5 *Soit F un graphe 1-libre et* $F^{\text{é}}$ *le graphe éclaté de F. Chaque composante connexe de* $F^{\text{é}}$ *a au plus un sommet variable.*

Démonstration : soit C une composante connexe de $F^{é}$. Par construction, les voisins communs $v_1, \ldots, v_k$ de deux sommets relations de C sont nécessairement des variables. Et comme C est 1-libre, on a $v_1 = \cdots = v_k$.

□

Suite à ces deux propriétés et sans perte de généralité, nous considérons maintenant des graphes cloués composés d'un seul sommet variable.

Définition 5.10 (Position d'échangeabilité, Irréalisable) *Soient F et G deux graphes polarisés et* $l_1 = {\sim} r(c_1, \ldots, c_k)$ *et* $l_2 = \overline{\sim r}(d_1, \ldots, d_k)$ *deux sommets relations opposés de F. Une* position d'échangeabilité *de* l_1 *et* l_2 *dans G est un k-tuple de termes* $(x_1, \ldots, x_k)$ *de G tel que :*

1. *Il existe deux substitutions* π *et* π' *des termes de F dans les termes de G ;*
2. *Quelque soit* $i = 1 \ldots k$, $\pi(c_i) = \pi'(d_i) = x_i$.

Cette position d'échangeabilité est dite irréalisable *(pour* l_1 *et* l_2*) si quelque soit* $i = 1 \ldots k$, $\pi(c_i) = \pi(d_i)$ *(resp.* $\pi'(c_i) = \pi'(d_i)$*).*

Intuitivement, une position irréalisable correspond au fait que π et π' ne peuvent pas être des homomorphismes de F dans des complétions de G, puisqu'ils conduiraient à envoyer deux sommets relations opposés à la même place.

Propriété 5.6 *Soient F et G deux graphes polarisés et $l_1 = {\sim}r(c_1,\ldots,c_k)$ et $l_2 = \overline{{\sim}r}(d_1,\ldots,d_k)$ deux sommets relations opposés de F. Si toutes les positions d'échangeabilité de l_1 et l_2 dans G sont irréalisables, alors l_1 et l_2 ne sont pas échangeables par rapport à G.*

Démonstration : par contradiction : supposons que l_1 et l_2 sont échangeables par rapport à G. Donc il existe une position d'échangeabilité de l_1 et l_2 dans G, soit $(x_1,\ldots,x_k)$, deux complétions totales de G, G_1 et G_2, et deux homomorphismes h_1 et h_2, respectivement de F dans G_1 et de F dans G_2 tels que quelque soit $i = 1\ldots k$, $h_1(c_i) = h_2(d_i)$. Par hypothèse $(x_1,\ldots,x_k)$ est irréalisable, donc quelque soit $i = 1\ldots k$, $h_1(c_i) = h_1(d_i)$ (resp. $h_2(c_i) = h_2(d_i)$). G_1 (resp. G_2) serait donc inconsistant. **Absurde**.

□

Propriété 5.7 *Soit F un graphe 1-libre ayant au plus un sommet variable, noté x, et $l_1 = {\sim}r(c_1,\ldots,c_k)$ et $l_2 = \overline{{\sim}r}(d_1,\ldots,d_k)$ deux sommets relations opposés de F. Si l'une des conditions suivantes est satisfaite alors l_1 et l_2 ne sont pas échangeables (et ceci quelque soit G) :*

(a) *il existe i tel que $c_i = A$ et $d_i = B$ avec $A \neq B$;*

(b) *il existe i et j (avec $i \neq j$) tel que $c_i = c_j = x$, $d_i = A$ et $d_j = B$ (avec $A \neq B$) ;*

(c) *il existe i tel que $c_i = d_i = x$;*

(d) *il existe i tel que $c_i = A$ et $d_i = x$, et $j \neq i$ tel que $c_j = x$ et $d_j = A$.*

Démonstration :

(a) D'après la définition 5.10, il n'y a pas de position d'échangeabilité de l_1 et l_2 dans G car $\pi(c_i) \neq \pi'(d_i)$.

(b) Nous montrons qu'il n'y a pas de position d'échangeabilité de l_1 et l_2 dans G. On a alors $\pi(x) = \pi(c_i) = \pi'(d_i) = A$ et $\pi(x) = \pi(c_j) = \pi'(d_i) = B$. **Absurde**.

(c) Nous montrons que les positions d'échangeabilité de l_1 et l_2 dans G sont irréalisables. Par hypothèse, il existe i tel que $c_i = d_i = x$ et on a $\pi(c_i) = \pi'(d_i)$, donc $\pi(x) = \pi'(x) = y$, où y est un terme quelconque de G. On a forcément $\pi(c_i) = \pi(d_i)$. Soit $j \neq i$: si $c_j = x$ (resp. $d_j = x$) avec $j \neq i$ alors soit $d_j = x$, soit $d_j = A$ et dans ce cas A et y représentent la même constante (resp. soit $c_j = x$, soit $c_j = A$). On a donc $\pi(c_j) = \pi(d_j)$.

(d) Nous montrons que les positions d'échangeabilité de l_1 et l_2 dans G sont irréalisables. Il existe i tel que $c_i = A$ et $d_i = x$ donc $\pi'(x) = A$, et $j \neq i$ tel que $c_j = x$ et $d_j = A$ donc $\pi(x) = A$. Si $c_l = B$ (resp. $d_l = B$) avec $l \neq i$ et $A \neq B$ alors $d_l = B$ (resp. $c_l = B$), sinon

nous sommes soit dans le cas (1) si $d_l = x$ soit dans le cas (2) si $d_l = C$ avec $B \neq C$, et donc $\pi(c_l) = \pi(d_l)$. Si $c_l = A$ et $d_l = x$ (resp. $c_l = x$ et $d_l = A$), alors $\pi(c_l) = \pi'(x) = A$ (resp. $\pi(d_l) = \pi(x) = A$). Or par hypothèse $\pi(d_l) = \pi(x) = A$ (resp. $\pi(c_l) = \pi'(x) = A$) donc $\pi(c_l) = \pi(d_l)$.

□

Propriété 5.8 *Soit F un graphe 1-libre ayant au plus un sommet variable, noté x, et $l_1 = {\sim} r(c_1, \ldots, c_k)$ et $l_2 = \overline{{\sim} r}(d_1, \ldots, d_k)$ deux sommets relations opposés de F. Si l_1 et l_2 ne vérifient pas les conditions de non-échangeabilité de la propriété 5.7, alors pour tout i si $c_i = A$ alors $d_i = A$, si $c_i = x$ alors $d_i = B$ et si $d_i = x$ alors $c_i = C$ avec $B \neq C$.*

Démonstration : nous faisons une étude par cas :

- Si c_i et d_i sont des constantes alors $c_i = A$ et $d_i = A$ car l_1 et l_2 ne vérifient pas la condition (a) de la propriété 5.7.
- Supposons que c_i soit égal à x : alors d_i est une constante B car l_1 et l_2 ne vérifient pas la condition (c) de la propriété 5.7.
  - **(a)** Supposons qu'il existe $j \neq i$ tel que $c_j = x$: alors d_j est la constante B car l_1 et l_2 ne vérifient pas la condition (b) de la propriété 5.7.
  - **(b)** Supposons qu'il existe $j \neq i$ tel que $d_j = x$: alors c_j est une constante C avec $B \neq C$ car l_1 et l_2 ne vérifient pas la condition (d) de la propriété 5.7.
- Supposons que d_i soit égal à x : le raisonnement est similaire au précédent.

Donc quelque soit i, si $c_i = A$ alors $d_i = A$, si $c_i = x$ alors $d_i = B$ et si $d_i = x$ alors $c_i = C$ avec $B \neq C$.

□

Démonstration du théorème 5.8 : soit $F^{cloué}$, $F^{é}$ son graphe éclaté et $F^{é}_1, \ldots, F^{é}_m$ les graphes correspondant aux composantes connexes de $F^{é}$. Chaque $F^{é}_i$ 0-libre est sans sommets relations échangeables (cf. lemme 5.2). Soit $F^{é}_i$ un graphe composé de sommets relations 1-libres ayant comme voisin commun le même sommet variable, soit x. D'après la propriété 5.8, soit ces sommets relations ne sont pas échangeables, soient ils contredisent la condition 2 de la définition 5.7 et $F^{é}_i$ n'est pas cloué. Chaque $F^{é}_i$ est donc sans sommets relations échangeables. Si $G \vdash F$, alors il existe un homomorphisme compatible de chaque $F^{é}_i$ dans G, et l'union de ces homomorphismes forme un homomorphisme compatible de $F^{cloué}$ dans G (cf. propriété 5.4, la compatibilité étant immédiate à vérifier).

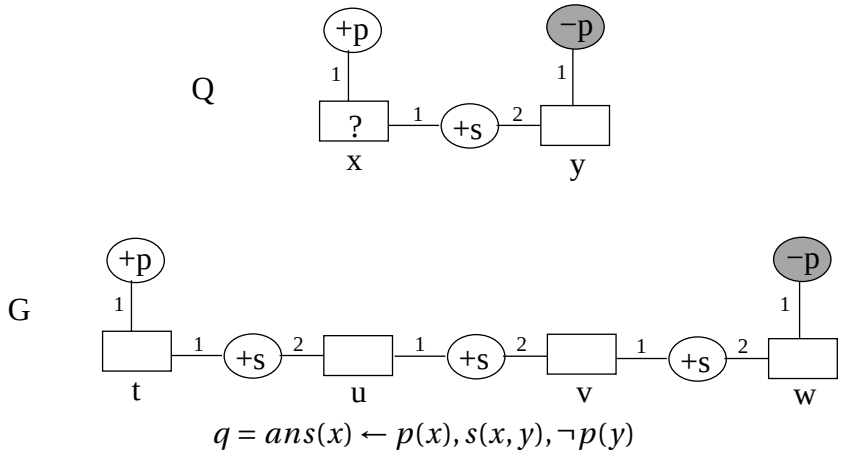
□

5.2 Réponse à une requête non-booléenne

Dans cette section, nous nous intéressons plus spécifiquement au problème de l'interrogation d'une base de connaissances, c'est-à-dire à la recherche de toutes les réponses à une $RC^\neg$ dans une base de connaissances sous l'hypothèse du monde ouvert. Dans la suite nous représentons une $RC^\neg$ par un graphe polarisé composé de sommets termes dits *distingués* (étiquetés par un point d'interrogation sur les dessins), appelé *lambda-GP*.

Définition 5.11 (Lambda-GP) *Un lambda-GP $Q = (t_1, \ldots, t_k)F$ est un graphe polarisé F pourvu d'un tuple de* sommets termes distingués *$t_1, \ldots, t_k$ avec $k \geq 0$, que nous appelons la* partie réponse. *Si $k = 0$, Q représente une $RC^\neg$ booléenne.*

La figure 5.3 montre le lambda-GP associé à une $RC^\neg$ non-booléenne q.



$$q = ans(x) \leftarrow p(x), s(x,y), \neg p(y)$$

FIGURE 5.3 : le lambda-GP Q associé à q et un graphe polarisé G.

Comme nous l'avons souligné au chapitre 2, section 2.2.3, la notion de réponse à une $RC^\neg$ q n'est plus « naturelle » dès lors que nous faisons l'hypothèse du monde ouvert. Il peut exister une « réponse » à q dans une base de connaissances B sans que nous puissions la construire. Ce « problème » est dû au fait que si $G \vdash Q$, où $Q = (t_1, \ldots, t_k)F$, il n'existe pas forcément, pour chaque complétion totale G^c, un homomorphisme h_j de F dans G^c tel que $(t_1, \ldots, t_k)$ se projette sur les mêmes sommets termes de G. Ceci est illustré par l'exemple 20 :

Exemple 20. Soient Q le lambda-GP et G le graphe polarisé de la figure 5.3. Nous allons voir qu'il n'y a aucune réponse à Q dans G. Soient $G_1, \ldots, G_4$ les quatre complétions totales de G par rapport à la relation p (cf. figure 5.4). On observe que pour G_1^c la réponse est v, pour G_2^c la réponse est soit t soit v, pour G_3^c la réponse est u et enfin pour G_4^c la réponse est t.

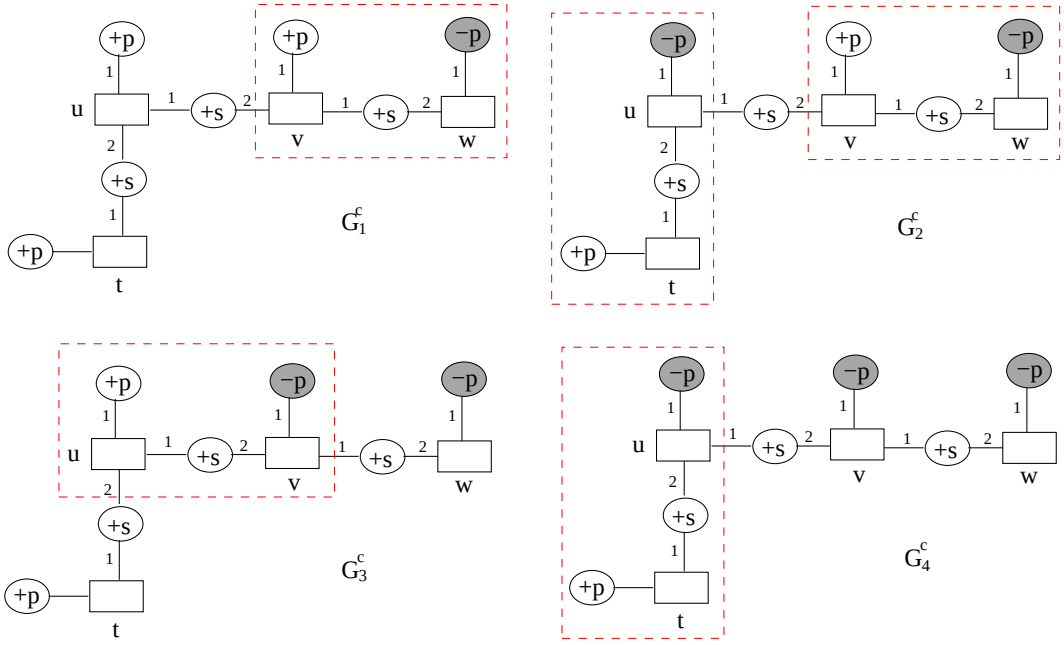


FIGURE 5.4 : les quatre complétions totales de G par rapport à p.

Dans [Mugnier et Leclère, 2006], une définition de réponse à une $RC^\neg$ en monde ouvert est proposée :

Définition 5.12 (Réponse) *Soit $Q = (t_1, \ldots, t_k)F$ un lambda-GP et B une base de connaissances composée d'un ensemble de graphes polarisés, une réponse à Q dans B est un tuple de termes $(u_1, \ldots, u_k) \in B$ tel qu'il existe une substitution π de $(t_1, \ldots, t_k)$ dans B avec $(\pi(t_1), \ldots, \pi(t_k)) = (u_1, \ldots, u_k)$, et pour toute complétion totale B^c de B, π s'étend à un homomorphisme de Q dans B^c.*

Exemple 21. Soient Q le lambda-GP et G le graphe polarisé de la figure 5.5. Nous allons voir qu'il y a une réponse à Q dans G. Soient $G_1, \ldots, G_4$ les quatre complétions totales de G par rapport à la relation r (cf. figure 5.6). Pour chaque complétion totale G^c de G, il existe un homomorphisme h_i tel que $h_i(x) = v$: pour G_1^c on a $h_1 = \{x \mapsto v, y \mapsto w, z \mapsto w\}$, pour G_2^c on a $h_1 = \{x \mapsto v, y \mapsto w, z \mapsto w\}$, pour G_3^c on a $h_2 = \{x \mapsto v, y \mapsto v, z \mapsto w\}$ et pour G_4^c on a $h_1 = \{x \mapsto v, y \mapsto v, z \mapsto w\}$. On en conclut que le tuple (v) est une réponse à Q dans G.

Une manière de calculer toutes les réponses à Q dans G est donc de calculer l'ensemble des substitutions de $(t_1, \ldots, t_k)$ dans G et de tester pour chaque substitution si elle s'étend à un homomorphisme de Q dans chaque complétion totale de G.

Cette méthode « brutale » conduit à l'algorithme `bruteForceAnswers` (cf. algorithme 16) qui construit l'ensemble des complétions totales de G.

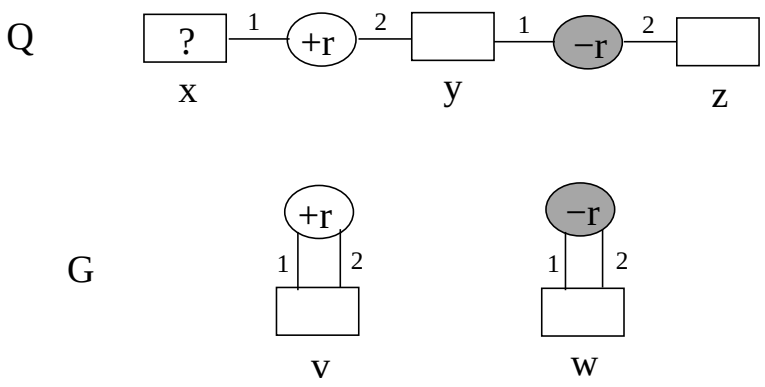


FIGURE 5.5 : un lambda-GP Q et un graphe polarisé G.

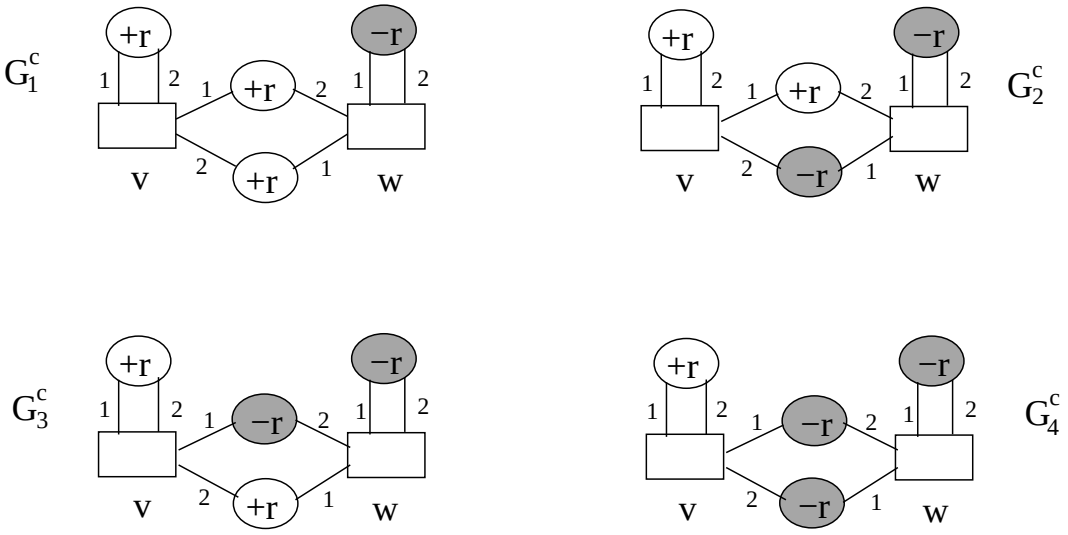


FIGURE 5.6 : les quatre complétions totales de G par rapport à r.

Algorithme 16: **bruteForceAnswers**(Q,G)

Données: un lambda-GP $Q = (t_1,\dots,t_k)F$ et un graphe polarisé consistant G
Résultat: la liste des réponses à Q dans G
début
  $rep \leftarrow Vrai$;
  Soit *Réponses*$= \emptyset$ l'ensemble des réponses à Q dans G ;
  Soit $\Pi = \pi_1,\dots,\pi_n$ l'ensemble des substitutions de $(t_1,\dots,t_k)$ dans G ;
  Soit $G^{complet}$ l'ensemble des complétions totales de G ;
  pour chaque $\pi_i \in \Pi$ **faire**
    pour chaque $G^c \in G^{complet}$ **faire**
      si π_i *ne s'étend pas à un homomorphisme de* Q *dans* G^c **alors**
        $rep \leftarrow Faux$;
        Sortie du Pour ;
    si $rep = Vrai$ **alors**
      Réponses$\leftarrow$*Réponses*$\cup\{(h_i(t_1),\dots,h_i(t_k))\}$;
    sinon
      $rep \leftarrow Vrai$;
  retourner $listeRep$;
fin

Une autre manière de résoudre le problème est de réutiliser l'un des algorithmes précédents comme une « boîte noire ». Plus précisément, nous calculons l'ensemble des substitutions de $(t_1,\dots,t_k)$ dans G, et pour chaque substitution π, nous fixons les images de $t_1,\dots,t_k$ avec $\pi(t_1),\dots,\pi(t_k)$ puis nous appelons au choix `recCheckPlus`, `breadthCheck`, `recANDcheck` ou `UNSATCheck`. Ceci permet de profiter des algorithmes précédents qui visent autant que possible à éviter le calcul de l'ensemble des complétions totales de G. Afin de fixer les images de $t_1,\dots,t_k$ avec $\pi(t_1),\dots,\pi(t_k)$ et pour éviter de modifier les algorithmes existants, on peut instancier partiellement F et G, c'est-à-dire remplacer les $\pi_i(t_j)$ qui ne sont pas des constantes par de *nouvelles* constantes (n'apparaissant ni dans F ni dans G), puis de remplacer dans F chaque t_j par $\pi_i(t_j)$. L'algorithme `computeAnswers` réutilise `recCheckPlus`.

Intuitivement, comme nous appelons dans le pire des cas $n_G^{n_Q}$ fois `recCheckPlus` (dans le cas d'un lambda-GP composé exclusivement de sommets termes distingués), où n_G et n_Q représentent respectivement le nombre de sommets termes dans G et dans Q, on peut penser que les temps de résolution devraient s'accroître de façon exponentielle avec le pourcentage de sommets termes distingués. Néanmoins, comme nous fixons pour chaque substitution les images de certains sommets termes, on s'attend également à ce que la résolution de l'algorithme `recCheckPlus` soit simplifiée, puisque nous pouvons observer que les sommets termes à image fixe se comportent exactement comme des constantes

Algorithme 17: **computeAnswers**(F, G)

Données: un lambda-GP $Q = (t_1, \ldots, t_k)F$ et un graphe polarisé consistant G
Résultat: la liste des réponses à Q dans G
début
  Soit $\Pi = \pi_1, \ldots, \pi_n$ l'ensemble des substitutions de $(t_1, \ldots, t_k)$ dans G ;
  Soit *Réponses*$= \emptyset$ l'ensemble des réponses à Q dans G ;
  pour chaque $\pi_i \in \Pi$ **faire**
    Instancier partiellement F et G avec de nouvelles constantes (remplacer les $\pi_i(t_j)$ qui ne sont pas des constantes par de *nouvelles* constantes, puis remplacer dans F chaque t_j par $\pi_i(t_j)$.) ;
    si ***recCheckPlus***$(G, \emptyset)$ = *succès* **alors**
      Réponses$\leftarrow$*Réponses*$\cup\{(h_i(t_1), \ldots, h_i(t_k))\}$;
  return Réponses ;
fin

(ils ont une image unique), et nous avons vu au chapitre 3, section 3.6.3, que plus il y avait de constantes dans le graphe source (ici Q) plus la résolution était rapide. En perspective, il serait intéressant de faire des expérimentations avec comme paramètre variable le pourcentage de sommets termes distingués dans le graphe source, ce qui n'a pas encore pu être effectué faute de temps.

5.3 Prise en compte d'une ontologie légère

Les résultats présentés dans ce mémoire ont été obtenus pour une ontologie triviale, c'est-à-dire restreinte à un vocabulaire, où tous les éléments sont deux-à-deux incomparables (voir chapitre 2, section 2.2 pour plus de détails). Nous considérons maintenant une ontologie « légère » composée de deux ensembles préordonnés respectivement de concepts et de relations et d'un ensemble d'individus (constantes). Précisons que seules les relations de même arité sont comparables. Dans cette ontologie les concepts et les relations sont atomiques, c'est-à-dire qu'ils n'ont pas de définition, et le seul lien entre différents éléments d'un même ensemble est le préordre (noté $\leq$). Par exemple soient c_1 et c_2 deux concepts, $c_1 \leq c_2$ signifie que c_1 est une spécialisation de c_2.

Dans cette section, nous nous appuyons notamment sur [Leclère et Mugnier, 2008] et [Mugnier *et al.*, 2009] qui nous fournissent l'ensemble des définitions et nous montrons que nos algorithmes peuvent s'étendre sans difficulté à une ontologie légère.

Traduisons maintenant une ontologie légère en un vocabulaire préordonné, c'est-à-dire dans lequel les relations (prédicats) sont munies d'un préordre traduisant la relation de spécialisation. Sans perte de généralité, nous traduisons les concepts en relations d'arité 1. Nous appelons alors $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$, le vocabulaire préordonné associé à une on-

tologie légère $\mathcal{O}$, où $\mathcal{R}_\mathcal{O}$ représente les concepts et les relations de $\mathcal{O}$ et $\mathcal{C}$ l'ensemble de ses constantes. Les étiquettes des sommets relations d'un graphe polarisé G sont donc construites à partir des relations appartenant à $\mathcal{R}_\mathcal{O}$ et comparées à l'aide du préordre sur $\mathcal{R}_\mathcal{O}$:

Définition 5.13 (Ordre étendu sur les étiquettes des sommets relations) *Soit le vocabulaire préordonné $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$ et deux relations de même arité r_1 et r_2 appartenant à $\mathcal{R}_\mathcal{O}$. Si $r_1 \leq r_2$ alors $+r_1 \leq +r_2$ et $-r_2 \leq -r_1$ (nous considérons l'ordre opposé à celui sur $\mathcal{R}_\mathcal{O}$ pour les étiquettes de sommets relations négatifs).*

Nous modifions alors de façon naturelle les définitions de sommets relations opposés, d'homomorphisme, d'homomorphisme compatible et de graphe polarisé inconsistant en prenant en compte cet ordre étendu :

Définition 5.14 (Sommets relations $\leq$-opposés, $\leq$-contradictoires) *Soient deux relations r et s appartenant à $\mathcal{R}_\mathcal{O}$ et $r \leq s$. Les sommets relations $-r(t_1, \ldots, t_k)$ et $+s(u_1, \ldots, u_k)$ sont dits* $\leq$-opposés*, les sommets relations $+r(t_1, \ldots, t_k)$ et $-s(u_1, \ldots, u_k)$ sont dits* $\leq$-fortement opposés *et les sommets relations $+r(t_1, \ldots, t_k)$ et $-s(t_1, \ldots, t_k)$ sont dits* $\leq$-contradictoires.

Définition 5.15 ($\leq$-homomorphisme) *Soient $F = (T_F, R_F, E_F, l_F)$ et $G = (T_G, R_G, E_G, l_G)$ deux graphes polarisés sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$. Un* $\leq$-homomorphisme *h de F dans G est une application des sommets de F dans les sommets de G telle que :*

1. $\forall (r, i, t) \in F, (h(r), i, h(t)) \in G$;
2. $\forall r \in R_F, l_G(h(r)) \leq l_F(r)$;
3. $\forall t \in T_F$, *si t est un sommet constante alors* $l_F(t) = l_G(h(t))$.

Notons que cette définition de $\leq$-homomorphisme correspond à celle appelée « projection » sur les graphes conceptuels de base [Chein et Mugnier, 2009].

Définition 5.16 ($\leq$-homomorphisme compatible) *Soient deux graphes polarisés F et G et F' un sous-graphe de F. Un $\leq$-homomorphisme h de F' dans G est dit* compatible *par rapport à F si :*

- *pour chaque sommet relation $\sim r(t_1, \ldots, t_k)$ de $F \setminus F'$, il n'existe pas de sommet relation contradictoire $\overline{\sim s}(h(t_1), \ldots, h(t_k))$ dans G.*
- *pour chaque paire de sommets relations $\leq$-fortement opposés $+r(c_1, \ldots, c_k)$ et $-s(d_1, \ldots, d_k)$ de $F \setminus F'$, $(h(c_1), \ldots, h(c_k)) \neq (h(d_1), \ldots, h(d_k))$.*

Définition 5.17 (Graphe polarisé inconsistant) *Un graphe polarisé sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$ est dit* inconsistant *s'il contient deux sommets relations $\leq$-contradictoires $+r(t_1, \ldots, t_k)$ et $-s(t_1, \ldots, t_k)$. Sinon il est dit* consistant.

De même, nous étendons les définitions de graphe complet et de complétion :

Définition 5.18 (Graphe $\leq$-complet) *Un graphe G consistant sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$ est dit* $\leq$-complet, *noté G^c, par rapport à un ensemble de relations $\mathcal{E} \in \mathcal{R}_\mathcal{O}$ si : pour chaque relation r d'arité k de $\mathcal{E}$ et chaque k-tuple de termes (non nécessairement distincts) $t_1, \ldots, t_k$ de G, G contient soit $+s(t_1, \ldots, t_k)$ avec $s \leq r$ soit $-s(t_1, \ldots, t_k)$ avec $r \leq s$.*

Définition 5.19 ($\leq$-Complétion, Littéral de complétion) *Une* $\leq$-complétion *G' d'un graphe G consistant sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$ est un graphe obtenu de G par ajouts successifs de nouveaux sommets relations non-redondants (étiquetés par des relations apparaissant déjà dans G ou plus spécifiques et portant sur des sommets termes déjà dans G), appelés* littéraux de complétion, *aussi longtemps qu'aucune inconsistance n'apparaît ($G \subseteq G'$). Chaque ajout représente une* étape de complétion. *Une $\leq$-complétion de G est dite* totale *par rapport à un ensemble de relations $\mathcal{E} \in \mathcal{R}_\mathcal{O}$ si c'est un graphe $\leq$-complet par rapport à $\mathcal{E}$, sinon elle est dite* partielle.

Le théorème 2.19 reste vrai à condition de modifier $G \vdash F$ par $\mathcal{V}_\mathcal{O}, G \vdash F$ et d'utiliser les définitions étendues pour les autres notions :

Théorème 5.20 *[Leclère et Mugnier, 2008] Soient deux graphes polarisés F et G (avec G consistant) sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$. $\mathcal{V}_\mathcal{O}, G \vdash F$* si et seulement si *quelque soit G^c, une $\leq$-complétion totale de G par rapport à l'ensemble des relations apparaissant dans G, il existe un $\leq$-homomorphisme de F dans G^c.*

Nous devons adapter les définitions de sommets relations échangeables et de vocabulaire de complétion :

Définition 5.21 (Sommets relations $\leq$-échangeables) *Deux sommets relations $\sim r(c_1, \ldots, c_k)$ et $\overline{\sim s}(d_1, \ldots, d_k)$ de F sont dits* $\leq$-échangeables *si (1) ils sont $\leq$-opposés, (2) il existe deux $\leq$-complétions totales de G, G_1 et G_2, et deux $\leq$-homomorphismes h_1 et h_2, respectivement de F dans G_1 et de F dans G_2 tels que $(h_1(c_1), \ldots, h_1(c_k)) = (h_2(d_1), \ldots, h_2(d_k))$.*

Nous définissons le graphe saturé de G, noté $G^\mathcal{O}$, comme le graphe obtenu à partir de G en ajoutant tous les sommets relations déductibles à partir du préordre sur $\mathcal{R}_\mathcal{O}$. Autrement dit, on rend explicite les effets du préordre sur les graphes.

Définition 5.22 (Graphe saturé) *Soient un graphe polarisé G sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$. On appelle graphe saturé de G par rapport à $\mathcal{R}_\mathcal{O}$, noté $G^\mathcal{O}$, le graphe polarisé obtenu de G en ajoutant chaque sommet relation $\sim s(t_1, \ldots, t_k)$ non présent dans G tel que $\sim r(t_1, \ldots, t_k)$ appartient à G et $\sim r \leq \sim s$.*

On a alors la propriété suivante (implicite dans [Mugnier *et al.*, 2009]) :

Propriété 5.9 *Soient F et G deux graphes polarisés sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$ et $G^\mathcal{O}$ le graphe saturé de G par rapport à $\mathcal{R}_\mathcal{O}$. On a $\mathcal{V}_\mathcal{O}, G \vdash F$ si et seulement $G^\mathcal{O} \vdash F$.*

Définition 5.23 (Vocabulaire de complétion) *Soient deux graphes polarisés F et G sur $\mathcal{V}_\mathcal{O} = (\mathcal{R}_\mathcal{O}, \mathcal{C})$, un* symbole de complétion *de F et G est une relation apparaissant dans des* sommets relations $\leq$-opposés *$-r(c_1, \ldots, c_k)$ et $+s(d_1, \ldots, d_k)$ telle qu'à la fois $-r$ et $+s$ apparaissent dans F et $G^\mathcal{O}$. Le* vocabulaire de complétion *de F et G, noté $\mathcal{V}_c$, est composé de l'ensemble des symboles de complétion de F et G.*

Prenons par exemple les graphes polarisés F, G et $G^\mathcal{O}$ de la figure 5.7 avec $p \leq q$. Le vocabulaire de complétion de F et G est l'ensemble $\mathcal{V}_c = \{p, q\}$. En effet, $-p$ et $+q$ apparaissent dans F et dans $G^\mathcal{O}$ (notons qu'ils n'apparaissent pas dans G). On peut voir que $G \vdash F$: il suffit de compléter G respectivement par $+p(u)$ et $-p(u)$ pour s'en assurer.

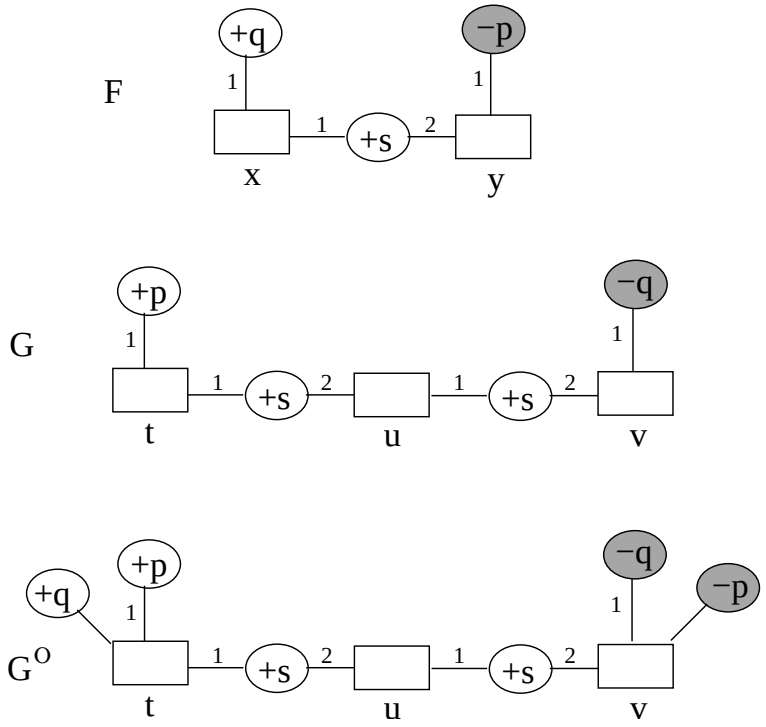


FIGURE 5.7 : vocabulaire de complétion : $V_c = \{p, q\}$ avec $p \leq q$.

Étant données les définitions étendues des notions d'homomorphisme, homomorphisme compatible, graphe complet et vocabulaire de complétion, les algorithmes `recCheckPlus`, `recANDcheck` et `breadthCheck` sont utilisables tels quels. L'adaptation de la méthode menant à l'algorithme `UNSATCheck` est directe : en effet il suffit de voir que $\mathcal{O}, G \vdash F$ si et seulement $G^\mathcal{O} \vdash F$, et donc de considérer $G^\mathcal{O}$ au lieu de G lors des trois étapes de résolution.

CHAPITRE

6 Conclusion

Nous nous sommes intéressés à deux problèmes équivalents concernant les requêtes conjonctives avec négation : le problème de l'inclusion de requêtes et celui de l'évaluation d'une requête booléenne sous l'hypothèse du monde ouvert. Nous avons reformulé ces problèmes sous la forme d'un problème de déduction dans un fragment de la logique du premier ordre.

Nous avons commencé par analyser les trois approches proposées antérieurement pour le résoudre [Ullman, 1997][Wei et Lausen, 2003][Leclère et Mugnier, 2007]. Nous nous sommes focalisés sur les deux dernières qui améliorent la première de façon différente. Dans [Wei et Lausen, 2003] un schéma d'algorithme de parcours en largeur de l'espace de recherche est proposé, mais nous montrons qu'il est incorrect. Nous proposons alors l'algorithme `breadthCheck` qui corrige l'erreur de l'algorithme de [Wei et Lausen, 2003] et améliore le processus. Nous proposons également l'algorithme `recANDcheck`, basé sur la propriété de Wei et Lausen [2003], qui effectue un parcours en profondeur de l'espace de recherche. [Leclère et Mugnier, 2007] propose un schéma d'algorithme de parcours en profondeur de l'espace de recherche (différent de celui utilisé pour `recANDcheck`), et nous présentons trois propositions d'heuristique afin d'optimiser la résolution, qui conduisent à l'algorithme raffiné `recCheckPlus`. Nous proposons également l'algorithme `UNSATCheck`, basé sur une approche différente de celles antérieures, qui conduit à tester l'insatisfiabilité d'une forme clausale propositionnelle, ce qui permet d'utiliser un solveur SAT.

Un des buts de cette thèse était de comparer expérimentalement différents algorithmes de résolution du problème de déduction. Comme nous n'avions pas de jeu de tests adaptés, nous avons mis en place un algorithme de génération aléatoire d'instances du problème de déduction. Nous avons alors analysé l'influence des différents paramètres de génération sur la difficulté de résolution du problème et mis en évidence des valeurs de paramètre pour lesquelles les instances générées sont généralement difficiles. Ce protocole

expérimental nous a alors permis d'estimer expérimentalement la pertinence de nos propositions d'affinement et de comparer les différents algorithmes. Nous avons montré que les heuristiques proposées étaient très pertinentes ; elles permettent de diminuer énormément le temps d'exécution de l'algorithme. Nous avons montré que l'algorithme `breadthCheck` était très mauvais par rapport aux deux autres, et que généralement `recCheckPlus` donnait les meilleurs résultats.

Nous avons également proposé plusieurs extensions de notre cadre d'étude, et montré que nos algorithmes pouvaient très facilement s'étendre pour les prendre en compte. Néanmoins le « prix à payer » de ces extensions, notamment en terme de temps de résolution, n'a pas encore été étudié.

Ce travail ouvre la voie à des perspectives de différente nature : algorithmiques, expérimentales et théoriques.

Tout d'abord, l'algorithme `recCheckPlus` peut encore être largement amélioré, en témoigne la figure 6.1 : elle montre les résultats obtenus, sur les mêmes instances, par `recCheckPlus` en prenant comme sous-graphe de guidage le meilleur choix ou le moins bon. Plus précisément, pour chaque instance nous avons exécuté `recCheckPlus` avec chaque sous-graphe de guidage maximal pour l'inclusion possible (8 pour cette expérimentation) et gardé à chaque fois le meilleur et le moins bon résultat. Notons que nous n'avons actuellement aucune heuristique pour choisir un sous-graphe de guidage maximal pour l'inclusion parmi d'autres ; il est choisi aléatoirement parmi tous les sous-graphes considérés. Néanmoins, il semble difficile de discriminer deux sous-graphes maximaux pour l'inclusion car cela touche à la structure même des graphes. Y a-t-il des structures plus intéressantes que d'autres ? Des structures ou des motifs particuliers à éviter ? De même, dans la phase de choix du littéral de complétion, nous nous basons sur un sommet relation frontière pour déterminer le prochain sommet à ajouter. Or il en existe généralement plusieurs possibles et nous n'avons actuellement aucune heuristique permettant de les discriminer. La question est alors de savoir sur quels critères se baser pour effectuer ce choix.

Une autre perspective serait de faire une comparaison de `recCheckPlus` (qui est actuellement le meilleur algorithme à notre disposition) avec des prouveurs logiques. Nous avons commencé cette comparaison en utilisant le prouveur logique *Prover9* (le successeur du prouveur logique *Otter* [McCune, 2003b]), et son outil complémentaire *Mace4* [McCune, 2003a]. Prover9, qui est basé sur la méthode de résolution, recherche une preuve de $(G \vdash F)$ et Mace4 recherche une interprétation qui soit un contre-exemple pour $(G \vdash F)$ en énumérant les interprétations par domaines de taille croissante. Pour comparer les deux algorithmes, nous exécutons `recCheckPlus` et `Prover9-Mace4` sur les mêmes instances. Pour chaque instance, Prover9 et Mace4 démarrent parallèlement et sont arrêtés lorsque le plus rapide conclut (nous multiplions alors par deux le temps d'exécution obtenu). Notons qu'on utilise la combinaison des deux algorithmes car les résultats qu'ils fournissent séparément sont très mauvais, et que Prover9 et Mace4 sont utilisés comme des boîtes noires, c'est-à-dire qu'on ne fait que les exécuter et qu'on récupère les résultats qu'ils fournissent. Les premiers résultats (figure 6.2) montrent que `recCheckPlus` est plus rapide que la com-

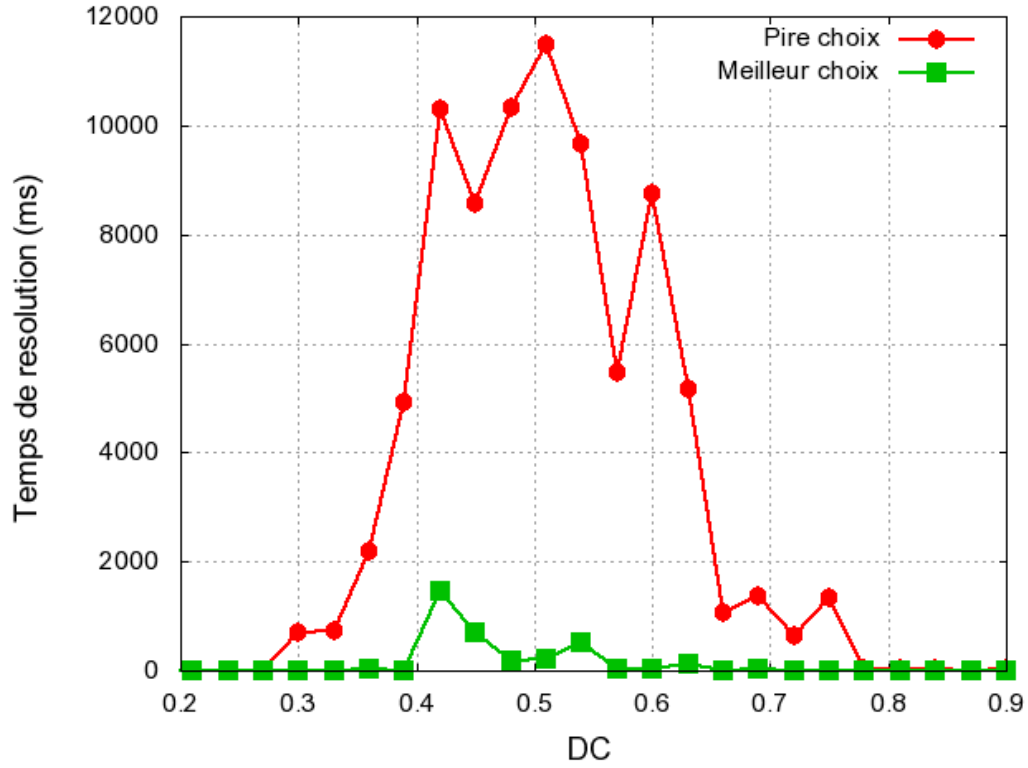


FIGURE 6.1 : Comparaison entre le meilleur et le moins bon choix du sous-graphe de guidage.

binaison de Prover9 et Mace4. Néanmoins, même si ces deux outils ont l'avantage d'être gratuits et simples d'utilisation, ils ne sont à l'heure actuelle pas les plus performants. Il serait donc intéressant d'étendre cette comparaison à d'autres prouveurs logiques. Une autre perspective qui va dans le même sens serait de comparer `recCheckPlus`, non plus à un prouveur général de la logique du premier ordre, mais à un prouveur résolvant un problème qui est dans la même classe de complexité (Π_2^p-complet), tel que le problème *2-QSAT* (aussi appelé *2-QBF*). *QSAT*, obtenu par généralisation du problème *SAT*, est le problème de décision de la satisfiabilité d'une formule propositionnelle sous forme clausale pour laquelle les variables booléennes sont soit quantifiées existentiellement soit universellement. Il est *PSPACE-complet*[1] [Papadimitriou, 1994]. Le problème *2-QSAT*, pour lequel il y a une alternance de deux quantificateurs avant les formules, l'un universel et l'autre existentiel, est un sous-problème de *QSAT* ; il est Π_2^p-complet [Papadimitriou, 1994][Schaefer et Umans, 2002]. Plus précisément, il prend en entrée une formule fermée de la forme

1. *QSAT* fait partie des problèmes les plus durs de *PSPACE*, qui est la classe des problèmes solubles par une machine de Turing déterministe en utilisant un espace polynomial et en temps illimité.

$f = \forall \overrightarrow{X} \exists \overrightarrow{Y} \varphi(\overrightarrow{X}, \overrightarrow{Y})$, où $\overrightarrow{X} = \{x_1, \ldots, x_m\}$ et $\overrightarrow{Y} = \{y_1, \ldots, y_n\}$ sont des ensembles disjoints de variables propositionnelles (c'est-à-dire interprétées par vrai ou faux) et $\varphi(\overrightarrow{X}, \overrightarrow{Y})$ est une conjonction de clauses de taille 3 (c'est à dire que chaque clause est une disjonction de trois variables de $\overrightarrow{X} \cup \overrightarrow{Y}$ ou leur négation). Le problème consiste à tester la satisfiabilité de f. Néanmoins, pour pouvoir transformer nos instances en des instances de *2-QSAT* (ou inversement), nous avons besoin de réductions polynomiales entre ces problèmes. Si théoriquement de telles réductions existent du fait de l'appartenance des deux problèmes à la même classe de complexité, elle n'est pas directe à construire et nous n'en avons pas actuellement. Ainsi un futur travail serait de construire une telle réduction et de comparer expérimentalement `recCheckPlus` aux prouveurs *2-QSAT*.

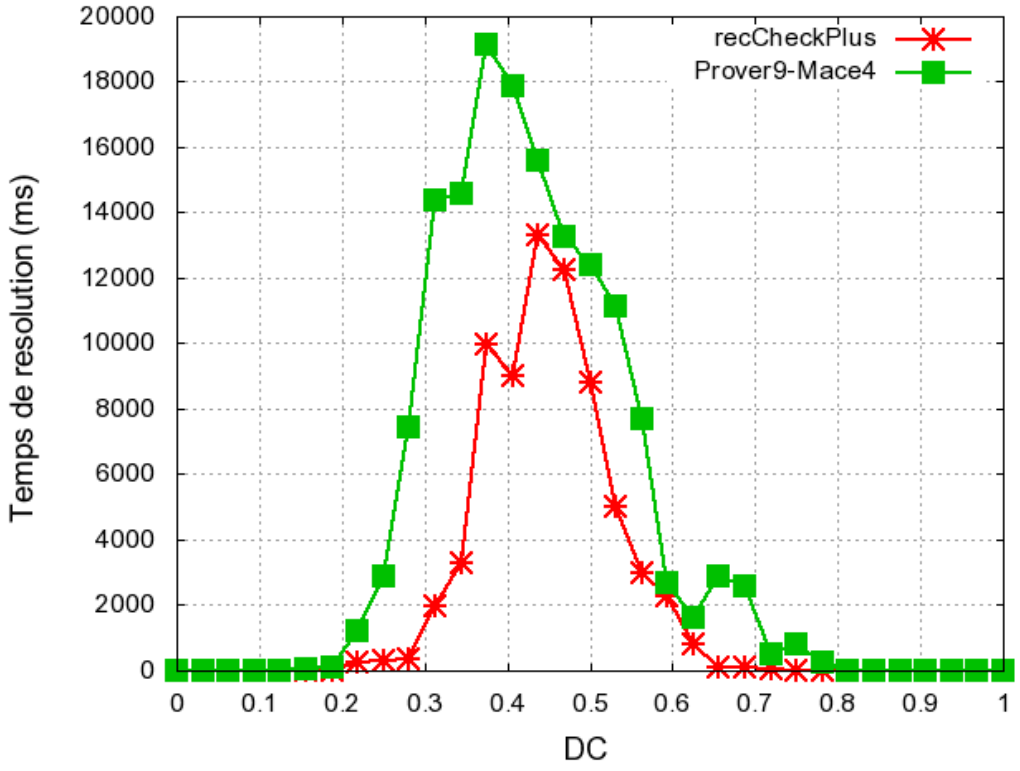


FIGURE 6.2 : Comparaison de `recCheckPlus` et de la combinaison `Prover9-Mace4`.

Dans ce travail, nous avons recherché expérimentalement des valeurs de paramètre pour lesquelles les instances générées étaient « difficiles » à résoudre (en terme de temps d'exécution). Nous avons notamment montré que les instances les plus difficiles à résoudre se trouvaient toujours au début de la transition de phase. Une perspective serait de pouvoir déterminer de façon théorique la « difficulté » d'une instance en fonction de la valeur de certains de ses paramètres. Pour cela il faudrait dans un premier temps être ca-

pable de déterminer l'emplacement de la transition de phase. De tels travaux existent pour plusieurs problèmes *NP-complets*, par exemple le problème de satisfaction de contraintes (CSP). Ils proposent des modèles de génération d'instances assurant l'existence d'une transition de phase, en identifiant l'emplacement exact et par là même assurant la génération d'instances « difficiles ». Peut-on adapter ces résultats au problème que nous étudions ?

Dans cette thèse nous avons considéré la négation classique. Néanmoins il existe plusieurs types de négation, et ils sont tous utiles dans les applications. On pourrait donc envisager de combiner ces différents types dans les requêtes et la base. Une approche intéressante dans ce sens est celle de Wagner [1991, 2003] qui, après une analyse des différents types de négation qui peuvent être trouvés dans les systèmes et langages existants, ainsi que dans la langue naturelle, propose de distinguer plusieurs types de relation. Les relations sont séparées en trois sous-ensembles : les relations sous l'hypothèse du monde clos et celles sous l'hypothèse du monde ouvert, elles-mêmes séparés en deux selon que la loi du tiers exclu s'applique ou non (c'est-à-dire que pour une telle relation, la formule $\forall(x_1,\ldots,x_n)r(x_1,\ldots,x_n) \vee \neg r(x_1,\ldots,x_n)$ est soit valide soit non valide). Ainsi un type de négation correspond à chaque type de relation. La logique proposée pour distinguer ces trois types de relations est une logique avec trois valeurs de vérité (vrai, faux et indéfini). Dans [Mugnier et Leclère, 2006], il est montré que chacun de ces différents types de négation peut être géré par des mécanismes basés sur l'homomorphisme. La question est comment les combiner efficacement dans nos algorithmes ?

D'autre part, nous considérons une ontologie très simple. Peut-on passer à une ontologie plus riche en gardant le même type d'algorithmes ? C'est une question totalement ouverte.

Bibliographie

S. Abiteboul et O. M. Duschka : Complexity of answering queries using materialized views. *In In PODS*, pages 254–263, 1998.

S. Abiteboul, R. Hull et V. Vianu : *Foundations of Databases : The Logical Level*. Addison-Wesley, 1995.

F. N. Afrati, C. Li et P. Mitra : On containment of conjunctive queries with arithmetic comparisons. *In Advances in Database Technology - EDBT*, pages 459–476, 2004.

A. V. Aho, Y. Sagiv et J. D. Ullman : Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.

K. R. Apt et R. Bol : Logic programming and negation : A survey. *Journal of Logic Programming*, 19:9–71, 1994.

F. Baader, D. Calvanese, D. McGuinness, D. Nardi et P. Patel-Schneider : *The Description Logic Handbook*. 2003.

K. Ben Mohamed, M. Leclère et M.-L. Mugnier : De la déduction dans le fragment $\{\exists, \wedge, \neg_a\}$ de la logique du premier ordre à sat. *Journées Nationales de l'IA Fondamentale*, oct 2008.

K. Ben Mohamed, M. Leclère et M.-L. Mugnier : Containment of conjunctive queries with negation : Algorithms and experiments. *In DEXA*, pages 330–345, aug 2010a.

K. Ben Mohamed, M. Leclère et M.-L. Mugnier : Déduction dans le fragment existentiel conjonctif de la logique du premier ordre : algorithme et expérimentations. *Reconnaissance des Formes et Intelligence Artificielle*, jan 2010b.

K. Ben Mohamed, M. Leclère et M.-L. Mugnier : Deduction in existential conjunctive first-order logic : an algorithm and experiments. *Artificial Intelligence : Methodology, Systems, Applications*, jan 2010c.

T. Berners-Lee, J. Hendler et O. Lassila : The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

N. Bidoit : Negation in rule-based database languages : A survey. *Theor. Comput. Sci.*, 78 (1):3–83, 1991.

A. Calì, D. Lembo et R. Rosati : On the decidability and complexity of query answering over inconsistent and incomplete databases. *In PODS*, pages 260–271, 2003.

D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzereni et R. Rosati : Dl-lite : Tractable description logics for ontologies. *American Association for Artificial Intelligence (AAAI)*, 2005.

A. K. Chandra et P. M. Merlin : Optimal implementation of conjunctive queries in relational databases. *In 9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.

M. Chein et M.-L. Mugnier : Conceptual graphs : fundamental notions. *Revue d'Intelligence Artificielle*, 6:365–406, 1992.

M. Chein et M.-L. Mugnier : *Graph-based Knowledge Representation and Reasoning—Computational Foundations of Conceptual Graphs*. Advanced Information and Knowledge Processing. Springer, 2009.

R. Fagin, P. G. Kolaitis, R. J. Miller et L. Popa : Data exchange : semantics and query answering. *In Theoretical Computer science*, 2005.

C. Farré, W. Nutt, E. Teniente et T. Urpí : Containment of conjunctive queries over databases with null values. *In ICDT 2007*, volume 4353 de *LNCS*, pages 389–403. Springer, 2007.

F. Fürst et F. Trichet : Heavyweight ontology engineering. *In OTM Workshops (1)*, pages 38–39, 2006.

I. P. Gent et T. Walsh : Beyond np : the qsat phase transition. pages 648–653. AAAI Press, 1999.

F. Giunchiglia, M. Marchese et I. Zaihrayeu : Encoding classifications into lightweight ontologies. *In Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), Budva*, pages 80–94, 2006.

G. Gottlob : Subsumption and implication. *Inf. Process. Lett.*, 24(2):109–111, 1987.

G. Gottlob, N. Leone et F. Scarcello : Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002.

T. R. Gruber : A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

A. Gupta, Y. Sagiv, J. D. Ullman et J. Widom : Constraint checking with partial information (extended abstract). *In In PODS*, pages 45–55. ACM Press, 1994.

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

T. Hogg et C. P. Williams : The hardest constraint problems : a double phase transition. *Artif. Intell.*, 69(1-2):359–377, 1994.

Anthony K. : On conjunctive queries containing inequalities. volume 35, pages 146–160, New York, NY, USA, 1988. ACM.

G. Kerdiles : *Saying it with Pictures : a logical landscape of conceptual graphs*. Thèse de doctorat, Univ. Montpellier II / Amsterdam, Nov. 2001.

M. Leclère et M.-L. Mugnier : Some Algorithmic Improvments for the Containment Problem of Conjunctive Queries with Negation. *In Proc. of ICDT'07*, volume 4353 de *LNCS*, pages 401–418. Springer, 2007.

M. Leclère et M.-L. Mugnier : An algorithmic study of deduction in simple conceptual graphs with classical negation. *In ICCS '08*, pages 119–132. Springer-Verlag, 2008.

A. Y. Levy, A. O. Mendelzon, Y. Sagiv et D. Srivastava : Answering queries using views, 1995.

A. Y. Levy et Y. Sagiv : Queries independent of updates. *In VLDB '93 : Proceedings of the 19th International Conference on Very Large Data Bases*, pages 171–181, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc. ISBN 1-55860-152-X.

W. McCune : Mace4 reference manual and guide. *CoRR*, cs.SC/0310055, 2003a.

W. McCune : Otter 3.3 reference manual. *CoRR*, cs.SC/0310056, 2003b.

M.-L. Mugnier et M. Leclère : On querying simple conceptuals graphs with negation. *Data and Knowledge Engineering (DKE)*, March 2006.

M.-L. Mugnier, G. Simonet et M. Thomazo : On the complexity of deduction in existential conjunctive first-order logic (long version). Rapport technique RR-09026, LIRMM, sep 2009.

M. Ortiz, D. Calvanese et T. Eiter : Characterizing data complexity for conjunctive query answering in expressive description logics. *In In Proc. of AAAI 2006*, 2006.

C. M. Papadimitriou : *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, 1994. ISBN 0201530821.

R. Reiter : On closed world data bases. *In Logic and Data Bases*, pages 55–76, 1977.

L. Sais : *Problème SAT : progrès et défis*. Hermes, 2008.

Y. P. Saraiya : *Subtree-elimination algorithms in deductive databases*. Thèse de doctorat, Stanford, CA, USA, 1991.

M. Schaefer et C. Umans : Completeness in the polynomial-time hierarchy : A compendium. *SIGACT News*, 33(3):32–49, septembre 2002.

B. M. Smith et S. A. Grant : Sparse constraint graphs and exceptionally hard problems. *In In Proceedings of IJCAI-95*, pages 646–651, 1994.

J. F. Sowa : Conceptual graphs for a data base interface. *IBM J. Res. Dev.*, 20(4):336–357, 1976. ISSN 0018-8646.

J. F. Sowa : Conceptual graphs. *In Information Processing in Mind and Machine*, pages 39–44. Addison-Wesley, 1984.

E. Tsang : *Foundations of Constraint Satisfaction*. 1993.

J. D. Ullman : Information Integration Using Logical Views. *In Proc. of ICDT'97*, volume 1186 de *LNCS*, pages 19–40. Springer, 1997.

G. Wagner : A database needs two kinds of negation. *In MFDBS 91*, pages 357–371. Springer-Verlag New York, Inc., 1991.

G. Wagner : Web rules need two kinds of negation. *In PPSWR*, pages 33–50, 2003.

F. Wei et G. Lausen : Containment of Conjunctive Queries with Safe Negation. *In International Conference on Database Theory (ICDT)*, 2003.

F. Wei et G. Lausen : A unified apriori-like algorithm for conjunctive query containment. *In IDEAS*, pages 111–120, 2008.

W. A. Woods : What's in a Link : Foundations for Semantic Networks. *In* D.G. Bobrow et A. Collins, éditeurs : *Representation and Understanding*. Academic Press, 1975.

M. Yannakakis : Algorithms for acyclic database schemes. *In In VLDB*, pages 82–94, 1981.

Table des figures

2.1 la base de données **CINEMA**. 6
2.2 un exemple simple d'ontologie organisée en une hiérarchie de spécialisation/-généralisation. Les rectangles et les ovales représentent respectivement les concepts et les relations ; T1, T2 et T3 sont les types universels, ils représentent respectivement « tout objet », « toute relation d'arité 2 » et « toute relation d'arité 3 ». 16
2.3 la partie raffinée de l'ontologie de la figure 2.2. 16
2.4 les graphes polarisés associés à f et g. 23
2.5 les quatre graphes représentant les quatre conjonctions de g'. 25
2.6 la méthode de Ullman. 27
2.7 les graphes associés à f et g. 29
2.8 l'arbre généré pour l'exemple 10. 30
2.9 exemple d'arbre de recherche de l'exemple 9. Chaque point noir représente un G^c et chaque carré un G_i. 33

3.1 deux graphes polarisés $F_1^{exemple}$ et $F_2^{exemple}$. 37
3.2 illustration d'une transition de phase de la phase déduction à la phase non-déduction. 40
3.3 illustration de l'inversion des signes d'un graphe. 42
3.4 influence des densités sur le temps de calcul. 44
3.5 influence des densités sur la taille de l'arbre de recherche. 44
3.6 influence des densités sur le pourcentage de déduction. 45
3.7 influence des densités sur le pourcentage de déduction. 46
3.8 influence du pourcentage de négation sur le temps de calcul. 47
3.9 influence du pourcentage de négation sur la taille de l'arbre de recherche exploré. 48

3.10 influence du pourcentage de négation sur la difficulté de résolution (temps de calcul). . . . 48
3.11 influence du pourcentage de négation sur la difficulté de résolution (taille de l'arbre de recherche exploré). . . . 49
3.12 influence du pourcentage de sommets constantes sur le temps de calcul. . . . 50
3.13 influence du pourcentage de sommets constantes sur la taille de l'arbre de recherche exploré. . . . 50
3.14 influence du nombre de sommets termes du graphe cible sur le temps de calcul. 51
3.15 influence du nombre de sommets termes du graphe cible sur la taille de l'arbre de recherche exploré. . . . 52
3.16 influence du nombre de sommets termes du graphe cible sur le pourcentage de timeouts. . . . 53
3.17 influence de l'arité des relations sur le temps de calcul. . . . 54
3.18 influence de l'arité des relations sur la taille de l'arbre de recherche exploré. . . . 54
3.19 influence de l'arité des relations sur le pourcentage de timeouts. . . . 55
3.20 influence du nombre de relations sur le temps de calcul. . . . 57
3.21 influence du nombre de relations sur la taille de l'arbre de recherche exploré. . . . 57

4.1 un bon choix de littéral de complétion : $\sim r(e,b)$. . . . 61
4.2 influence du choix de complétion : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$. . . . 63
4.3 influence de l'ordre d'exploration : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$. . . . 64
4.4 influence du filtrage dynamique sur la taille de l'arbre de recherche exploré : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$. . . . 67
4.5 influence du filtrage dynamique sur le temps d'exécution : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$. . . . 68
4.6 influence du filtrage dynamique sur le nombre de tests d'homomorphisme effectués : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $neg = 50\%$. . . . 69
4.7 un exemple d'arbre de recherche. . . . 73
4.8 un exemple d'arbre de recherche ET/OU. . . . 75
4.9 parcours de la même complétion ($G_3 = G_4$). . . . 75
4.10 deux graphes polarisés F et G. . . . 76
4.11 un arbre de recherche en profondeur prouvant que $G \vdash F$. . . . 77
4.12 l'arbre de recherche généré par `WL`. . . . 77
4.13 le graphe d'expansion généré par `breathCheck` avant la propagation. . . . 83
4.14 le détail de la propagation sur le graphe d'expansion de la figure 4.13. . . . 84
4.15 influence de la fusion sur le temps de calcul. . . . 85
4.16 arbre de recherche de la méthode étudiée. . . . 86
4.17 illustration de la propriété 4.2 87
4.18 les graphes polarisés associés à f et g. . . . 88

4.19 comparaison des quatre algorithmes. 93
4.20 comparaison des algorithmes `recCheckPlus` et `recAndCheck`. 94
4.21 comparaison des algorithmes `UNSATCheck` et `recCheckPlus` : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 3$, $k = 2$, $DS = 6\%$, $DC = 51\%$ et $neg = 50\%$. 95

5.1 non-extension de la propriété 5.1 à des sommets relations 1-libres. 100
5.2 non-extension de la propriété 5.3 à des sommets relations 1-libres. 101
5.3 le lambda-GP Q associé à q et un graphe polarisé G. 105
5.4 les quatre complétions totales de G par rapport à p. 106
5.5 un lambda-GP Q et un graphe polarisé G. 107
5.6 les quatre complétions totales de G par rapport à r. 107
5.7 vocabulaire de complétion : $V_c = \{p, q\}$ avec $p \leq q$. 112

6.1 Comparaison entre le meilleur et le moins bon choix du sous-graphe de guidage. 115
6.2 Comparaison de `recCheckPlus` et de la combinaison `Prover9-Mace4`. 116

Liste des tableaux

2.1 les homomorphismes de l'exemple 10. 30

3.1 valeur de *DC* choisie pour chaque valeur du pourcentage de négation. 47
3.2 valeur de *DC* choisie pour chaque valeur du nombre de sommets termes du graphe cible. 51
3.3 influence du nombre de relations sur la densité du graphe source. 58

4.1 influence du filtrage dynamique : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 2$, $k = 2$, $DS = 9.375\%$, $DC = 43.75\%$, $neg = 50\%$. 68
4.2 les homomorphismes de l'exemple 12. 77
4.3 les homomorphismes de l'exemple 13. 83
4.4 comparaison détaillée des algorithmes `recCheckPlus` et `UNSATCheck` : $nbTS = nbTC = 8$, $c = 0\%$, $nbE = 3$, k=2, $DS = 6\%$, $DC = 51\%$ et $neg = 50\%$. 93

Abstract

In this thesis, we consider problems at the intersection of two areas: databases and knowledge bases. We focus on two equivalent problems on conjunctive queries with negation : *query containment* and *query answering with boolean queries* while making the open-world assumption. We reformulate these problems as a deduction problem in a first-order logic fragment. Then we refine existing algorithm schemes and propose new algorithms. To study and compare them experimentally, we propose a random generator and we analyze the influence of parameters on problem instances difficulty. Finally, we analyze the improvements brought out by our refinements and we compare experimentally the algorithms.

Keywords: *Databases, Knowledge bases, Deduction, Negation, Conjunctive Queries with Negation, Graphs, Homomorphism, Algorithm, Heuristics, Experiments*

Résumé

Dans cette thèse, nous nous intéressons à des problèmes à la croisée de deux domaines, les bases de données et les bases de connaissances. Nous considérons deux problèmes équivalents concernant les requêtes conjonctives avec négation : l'*inclusion de requêtes* et l'*évaluation d'une requête booléenne* sous l'hypothèse du monde ouvert. Nous reformulons ces problèmes sous la forme d'un problème de déduction dans un fragment de la logique du premier ordre. Puis nous raffinons des schémas d'algorithmes déjà existants et proposons de nouveaux algorithmes. Pour les étudier et les comparer expérimentalement, nous proposons un générateur aléatoire et analysons l'influence des différents paramètres sur la difficulté des instances du problème étudié. Finalement, à l'aide de cette méthodologie expérimentale, nous comparons les apports des différents raffinements et les algorithmes entre eux.

Mots clefs : *Bases de données, Bases de connaissances, Déduction, Négation, Requêtes conjonctives avec négation, Graphes, Homomorphisme, Algorithme, Heuristiques, Expérimentations*

LIRMM — 161, rue Ada — 34095 Montpellier cedex 5 — France