De la pertinence de l'énumération : complexité en logiques propositionnelle et du premier ordre

Johann Brault-Baron

université de Caen
Basse-Normandie

Ecole Doctorale
SIMEM

De la pertinence de l'énumération

Complexité en logiques propositionnelle et du premier ordre

Thèse de doctorat

Présentée et soutenue publiquement le : **Jeudi 18 avril 2013**
par : **Johann Brault-Baron**
afin d'obtenir le : **doctorat de l'Université de Caen Basse-Normandie**
Spécialité : **Informatique et applications**
Sous la direction de : **Étienne Grandjean**

Jury

Composé de :

- **Manuel Bodirsky** chargé de recherche CNRS, HDR, LIX, École Polytechnique, Paris *(rapporteur)*,
- **Luc Segoufin** directeur de recherche INRIA, LSV, ENS Cachan *(rapporteur)*,
- **Bruno Courcelle** professeur émérite, LaBRI, Université de Bordeaux, membre honoraire de l'IUF,
- **Arnaud Durand** professeur, IMJ, équipe logique mathématique, Université Paris 7,
- **Michel Habib** professeur, LIAFA, Université Paris 7,
- **Miki Hermann** chargé de recherche CNRS, HDR, LIX, École Polytechnique, Paris,
- **Bruno Zanuttini** maître de conférences, HDR, GREYC, UCBN,
- **Étienne Grandjean** professeur, GREYC, UCBN *(directeur de thèse)*.

Avant-propos

De ce mémoire

Ce document n'est pas son objet. Si tel était le cas, la seconde partie, qui occupe ici près de quatre-vingt pages, n'en occuperait que neuf : c'est suffisant pour prouver *ex nihilo* son unique résultat, et l'énoncer dans une forme qui révèle d'elle-même sa signification. Pour ce faire, il faudrait s'abstenir de décrire les étapes successives de la démarche qui y a conduit, et donc renoncer à valoriser le progrès accompli ; en un mot, rendre ce document inadéquat à son usage.

Ce manuscrit est donc construit de manière à correspondre à ce qui est attendu d'un mémoire : d'une part, le contenu logique est exposé et trouve son utilité justifiée, permettant l'évaluation des résultats ; d'autre part, la progression est visible, particulièrement dans l'enchaînement des chapitres de la seconde partie, qui est bien de la forme canonique :

1. situation initiale (une étude des requêtes conjonctives acycliques)
2. élément déstabilisateur (dichotomie pour la classe duale)
3. résolution (dichotomie généralisée).

Si le chemin parcouru est visible dans la forme actuelle du manuscrit, le but à atteindre reste bien d'effacer ce chemin et ses errements — jeter l'échelle après y être monté — et de présenter finalement un objet *autonome*, *cohérent* et *simple* ; ce document ne vise qu'à convaincre le lecteur que tous les matériaux nécessaires ont été réunis.

Si ce mémoire ne constitue pas en lui-même le projet dont il décrit l'avancement, en revanche il en illustre la démarche : prenant acte de l'inflation du corpus actuel d'articles scientifiques, et de la variété de notions (souvent redondantes ou d'une complexité inutile) qui en découle, il met en place une sélection des notions et formalismes associés. Il promeut ainsi par l'exemple l'usage de certains au détriment d'autres ; ceux utilisés le sont à titre expérimental, et sont conservés ceux qui s'avèrent pertinents, c'est–à–dire qui permettent simultanément les développements les plus simples et l'émergence d'un sens clair. Ainsi, certaines notions — même très classiques — ont finalement été abandonnées car facteurs de complication sans réelle utilité.

Par ailleurs, ce mémoire se rapproche de l'ambition de son objet : être capable de justifier sa masse de papier dans les archives du lecteur par une masse supérieure d'articles qui peuvent en être retirés car rendus ainsi obsolètes : la première partie par exemple propose des algorithmes d'énumération qui rendent respectivement inutiles au moins deux algorithmes optimaux de décision chacun.

Finalement, ce manuscrit présente certaines des caractéristiques de son objet : le souci d'*autonomie* et la recherche du *minimalisme*, particulièrement visibles dans la seconde partie. L'autonomie s'y exprime par l'usage de notions primitives : ensembles, n-uplets et fonctions ; et

par la construction de toutes les autres à partir de ces premières. Le minimalisme consiste, quant à lui, à faire usage du moins de nouvelles notions possibles, et à choisir les notations qui permettent l'expression la plus limpide des résultats et les preuves les plus concises.[1]

Notes sur la forme

Évidemment, cette recherche de la forme la plus légère serait un échec sans une typographie cohérente avec ce but. Là encore, ce document est l'occasion d'une expérience, de typographie en l'occurrence, dont les choix ont été guidés par trois grands axes :

La cohérence des modes de représentation s'exprime par la normalisation des notations, et par la minimisation des structures de texte : par exemple les symboles intégrés aux figures sont considérés comme homogènes à des formules, et sont, à ce titre, exprimés à l'aide de la même police, taille et graisse que les formules « normales ».[2]

La légèreté des mises en forme, particulièrement cruciale pour éviter les interférences visuelles entre éléments structurants, s'est heurté à la nécessité de disposer d'une hiérarchie « profonde » : parties, chapitres, sections, sous-sections, sous-sous-sections, paragraphes ou environnement logiques (qu'il s'agit de différencier !), et finalement sous-paragraphes comme les listes ou les formules hors-lignes.

L'économie de moyens se traduit par un usage exclusif du noir,[3] de polices restant lisibles sur des tirages de mauvaise qualité et par une mise en page relativement compacte qui limite le nombre de pages du document ; autant d'éléments qui permettent une impression abordable.

1 La notation choisie pour les hypergraphes est à cet égard une réussite : très peu de notations leur sont spécifiques, leur définition permettant de réutiliser les notations ensemblistes, et les énoncés des lemmes et leur preuves sont très concis.

2 Ceci a fait l'objet d'un travail particulier dans la mesure où il s'agissait de produire des figures dans le langage tout à fait malcommode du processeur de texte (voir [RM12, Ros99].) lui-même, aussi on a dû programmer un logiciel qui exploite et complète un logiciel de rendu graphique pour produire des figures dans le langage du processeur de texte. C'est un apport dans la mesure où, à notre connaissance, aucun logiciel ne produit de représentation pour les hypergraphes, un objet pourtant de plus en plus courant en informatique.

3 Les « gris » employés sont destinés à être simulés *via* des remplissages par points par une imprimante *noir et blanc*. Une impression réellement en gris (en *couleur* grise) donnerait un gris trop homogène qui choquerait par son contraste avec le gris de texte, très granuleux.

Introduction

> L'homme de Science le sait bien, lui, que, sans la Science, l'homme ne serait qu'un stupide animal sottement occupé à s'adonner aux vains plaisirs de l'amour dans les folles prairies de l'insouciance, alors que la Science, et la Science seule, a pu, patiemment, au fil des siècles, lui apporter l'horloge pointeuse et le parcmètre automatique sans lesquels il n'est pas de bonheur terrestre possible.
>
> Pierre Desproges – *Vivons heureux en attendant la mort*

L'objet d'étude de l'informatique est le traitement de l'information ; la majeure partie des travaux traite essentiellement d'analyse d'information qui consiste, à partir de données de taille variable, à produire une donnée unitaire. Le problème de décision est la formalisation de cette approche, et joue un rôle central en algorithmique et en complexité. Ceci dit, si considérer uniquement des problèmes de décision permet de se focaliser sur l'analyse de l'information d'entrée, le prix à payer en est l'escamotage de toute considération sur la production de l'information de sortie. Cette vue partielle se reflète à travers toute l'informatique : par exemple la notion de fonction, omniprésente y compris dans l'architecture même des processeurs, reflète la vue d'un calcul comme la production d'un objet de taille fixe.

On va s'intéresser à la synthèse d'information en lui donnant une importance non moindre que celle accordée à l'analyse de l'information. Pour ce, on se place dans un cadre presque classique : les objets produits sont encore et toujours de taille constante, mais en nombre indéterminé cette fois. On aborde donc le flux de sortie d'un programme comme un ensemble plat, non structuré, de données de même nature. Malgré cette limitation, on va tout de même pouvoir établir le bien-fondé de ce modèle dans deux domaines bien distincts qui justifient le découpage de ce document.

Dans une première partie, nous interrogeons la pertinence du problème d'énumération dans un cadre extrêmement classique : la logique propositionnelle. Le résultat fondamental, le théorème de Schaefer [Sch78], a déjà été replacé dans un cadre d'énumération par Nadia Creignou et Jean-Jacques Hébrard [CH97] : alors que les seules classes traitables pour Sat sont les classes de Schaefer auxquelles ils faut ajouter les classes artefacts zéro-valide et un-valide, Creignou et Hébrard ont montré que, en revanche, si l'on considère le problème de satisfaisabilité non triviale, alors les seules classes polynomiales sont les classes de Schaefer, ce qui suggère fortement que le « vrai » problème de décision est le problème de satisfaisabilité non triviale. Ils montrent aussi que ces classes polynomiales sont précisément les classes admettant un algo-

rithme d'énumération à délai polynomial, ce qui établit le lien entre décision et énumération.

Nous appuyons cette conclusion en donnant, pour chacune de ces classes, un algorithme à délai linéaire $\mathcal{O}(|F|)$ (et même mieux) après le temps de la décision :

- pour le cas bijonctif, nous explicitons l'algorithme de Feder [Fed94] qui énumère à délai optimal après précalcul linéaire, et le généralisons à l'énumération des Horn-renommages ;
- pour le cas Horn, nous proposons un algorithme d'énumération à délai quasi-linéaire, voire sans doute linéaire ;
- le cas affine ne requiert qu'une seule astuce simple, présentée dans la thèse, pour être énumérable à délai optimal après un précalcul correspondant à la décision.

Le résultat important est que, au regard de ceux-ci, il ne peut pas exister d'algorithme de satisfaisabilité non triviale qui améliore les bornes (linéaires) données par les algorithmes d'énumération. Ceci prouve qu'*il n'y a pas de véritable algorithme de décision pour la satisfaisabilité non triviale*. On en conclut que l'énumération est le seul véritable problème dont la complexité permet de distinguer les « vraies » classes faciles des classes difficiles.

Dans une seconde partie, nous partons du résultat de dichotomie de G. Bagan [Bag09, BDG07] pour l'énumération des requêtes conjonctives : celle-ci se fait à délai constant après un précalcul linéaire si et seulement si la requête est α-acyclique, c'est-à-dire que son hypergraphe est α-acyclique.

D'abord, nous proposons un formalisme simple et clair qui permet déjà de prouver simplement des résultats informellement connus pour beaucoup et d'en apporter de nouveaux. Il nous permet aussi de simplifier et compléter le résultat de Guillaume Bagan et d'apporter des réponses claires et définitives à ses questions ouvertes.

Ensuite, grâce à un nouvel algorithme, nous proposons un résultat similaire et un peu surprenant sur les requêtes conjonctives *dont les atomes sont niés* : elles sont décidables en temps *quasi*-linéaire si et seulement si elles sont β-acycliques. De plus, le résultat de facilité s'étend *à tout le premier ordre existentiellement quantifié*. Il n'est pas difficile d'étendre ce résultat à l'énumération.

La clef de voûte de cette partie est une généralisation du résultat classique de combinatoire de Brouwer et Kolen[BK80], qui permet d'unifier l'ensemble des résultats de cette partie sous forme d'une seule dichotomie *simple* pour l'énumération dont les deux dichotomies susmentionnées se trouvent être des corollaires. Cet îlot de traitabilité optimal étend les requêtes conjonctives α-acycliques, et comble leurs lacunes les plus évidentes. Une dichotomie pour la totalité du premier ordre existentiel semble désormais accessible.

Par ailleurs, cette dichotomie établit l'équivalence entre la facilité de l'énumération d'une requête conjonctive signée et la facilité de la décision d'une requête conjonctive signée comportant un atome nié de plus, qui contient l'ensemble des variables libres de la première. Ceci établit un lien fort entre décision et énumération qui accorde à cette dernière une pertinence au moins aussi fondée que celle du problème de décision.

Si les deux parties de cette thèse établissent des résultats de classification dichotomique de la complexité, notons qu'il existe par ailleurs dans la littérature une grande variété de résultats de classification, portant en général sur des problèmes de décision et la distinction polynomial/complet pour une classe difficile par réduction polynomiale. Sans vouloir être exhaustifs, citons entres autres :

- la dichotomie de Schaefer susmentionnée, et ses variantes sur la logique propositionnelle, par exemple [CH96, CH97, CKS01, CCHS10, BP11] ;
- des résultats et conjectures de dichotomie sur les CSP [FV98, BCF12] ; et
- des classifications / dichotomies pour les requêtes conjonctives ou du premier ordre [PY99, GSS01] ou du second ordre [GKS04].

La problématique principale de cette thèse est la complexité de l'énumération comparée à celle de la décision, ceci dans deux cadres de la logique classique. Il faut noter que les problèmes d'énumération ont été étudiés dans des cadres et perspectives divers dans la littérature. Citons principalement :

en combinatoire et algorithmique : algorithmes d'énumération portant sur les graphes (voir par exemple [Uno98, Uno01]), ou encore en lien avec la complexité, l'article classique [JPY88] ou la logique [Fed94].

en logique et complexité : en logique propositionnelle [CH97], en logique du premier ordre [DO06, DG07, BDG07, BDGO08, KS11] ou du second ordre [Bag06, Cou09, Str10, DS11, KS].

Sommaire

I Logique propositionnelle 1

1 Logique propositionnelle et énumération 5
- 1 Logique propositionnelle . . . 6
- 2 Satisfaisabilité généralisée . . . 12
- 3 Des formules affines . . . 14

2 Formules bijonctives et Horn-renommages 19
- 1 Des formules bijonctives . . . 20
- 2 Des formules Horn-renommables . . . 28

3 Models of Horn Formulas are Enumerable at Linear Delay 39
- 1 A First Approach . . . 40
- 2 Incremental Circuit Building . . . 42
- 3 The Final Algorithm . . . 43

II Logique du premier ordre 49

4 About Conjunctive Queries 53
- 1 Optimal Enumeration Class and Reduction . . . 55
- 2 Properties of Queries Classes . . . 61
- 3 Conjunctive Queries and Acyclicity . . . 72

5 A Negative Conjunctive Query is Easy iff it is Beta-Acyclic 89
- 1 Preliminaries and Results . . . 91
- 2 Davis-Putnam Resolution w.r.t a Nest Point . . . 95
- 3 Easiness Result . . . 97
- 4 Hardness Result . . . 102

6 Dichotomies pour les requêtes conjonctives signées. 105
- 1 Acyclicité des hypergraphes . . . 106
- 2 Des requêtes . . . 116

7 Encore un peu de combinatoire ? 127

Table des matières 139

Définitions, lemmes et théorèmes 143

Liste des algorithmes et figures 147

Bibliographie 149

Première partie

Logique propositionnelle

Introduction à la première partie

De l'énumération en logique propositionnelle

Un problème de décision particulièrement classique (le plus classique ?) est le problème de satisfaisabilité d'une formule propositionnelle. Nous reprenons le résultat fondamental, le théorème de Schaefer, et le replaçons dans un cadre d'énumération.

Le théorème de dichotomie de Creignou et Hébrard [CH97] établit l'équivalence entre :

- les classes polynomiales pour le problème de l'existence de solutions *non triviales* (concrètement autres que « faux partout » et « vrai partout »),[4]
- les classes admettant un algorithme d'énumération à délai polynomial,
- les fameuses classes de Schaefer (voir par exemple [CKS01]), que l'on peut considérer comme les classes réellement pertinentes.[5]

Nous renforçons cette dichotomie en donnant des algorithmes optimaux pour chacune des classes de Schaefer :

- on présente, au chapitre 1, un algorithme simple à délai constant après le temps de la décision pour les formules affines ;
- on présente, au chapitre 2, pour la première fois, une version *lisible* de l'algorithme de Feder [Fed94], qui énumère les modèles d'une formule bijonctive à délai constant après précalcul linéaire ;
- on présente, au chapitre 3, un premier algorithme d'énumération pour une formule de Horn à délai (quasi-[6]) linéaire.

Ces algorithmes ont ceci d'instructif qu'ils se trouvent rendre obsolète tout algorithme de satisfaisabilité non–triviale, prouvant ainsi qu'il n'y a pas d'algorithme de décision non–triviale intrinsèque, c'est–à–dire qui soit autre chose qu'une version mutilée de l'algorithme d'énumération.

On a de plus une dichotomie très nette entre génération NP-difficile des trois premières solutions[7] et énumération très efficace — seule la première solution coûte, les autres sont au pire linéaires, au mieux gratuites ; ce qui rejoint l'adage connu à propos des pas qui coûtent.

On en conclut que, du moins dans le cadre propositionnel, le problème d'énumération, bien loin d'être un problème exotique ou une variante quelconque, est en fait un problème tout ce qu'il y a de plus pertinent.

---

4 Les deux classes polynomiales triviales pour la décision sont la classe des formules zéro-valides et celle des formules un-valide, pour lesquelles il est facile de tester l'existence d'une telle solution triviale, mais NP-difficile de tester l'existence d'une solution non triviale.

5 Les classes de Schaefer sont les classes bijonctives, affines, Horn et anti-Horn.

6 Voir remarque en fin de la première partie.

7 Soit la décision est NP-complète, soit l'unicité est NP-complète (ex : zéro-valide), soit le problème d'existence d'une troisième solution est NP-complet (ex : à la fois zéro-valide et complémentif).

Organisation de cette partie

Le premier chapitre introduit formellement la problématique, montre que l'algorithme de Davis et Putnam [DP60, DLL62] est adaptable pour l'énumération sans perte de ses « bonnes » propriétés, et résout d'emblée le cas facile des formules affines.

Le second chapitre présente l'algorithme de Feder [Fed94] pour l'énumération des modèles d'une formule bijonctive, en reprenant toutes les étapes, y compris les plus classiques, à travers un exposé qui comprend illustrations, algorithmes et exemples. Si une amélioration de la qualité de la rédaction est encore nécessaire, on a d'ores et déjà tous les éléments pour construire un bon support de cours sur ce sujet.

De plus, une généralisation du problème de décision des formules bijonctives, le problème du Horn-renommage d'une formule F, est lui aussi traité. Sa décision et son unicité étaient connues comme linéaires [Héb94, Héb95] ; on généralise ce résultat en prouvant que l'énumération est à délai $\mathcal{O}(|F|)$ (et en espace total $\mathcal{O}(|F|)$).

On généralise finalement simultanément cet algorithme et l'algorithme de Feder en proposant un algorithme d'énumération des Horn-renommages, qui énumère les $k-1$ premiers renommages d'une formule F à délai $\mathcal{O}(|F|)$, où k est la longueur maximale d'une clause de F (donc majorée par n le nombre de variables de F), puis les suivants à délai $\mathcal{O}(n)$, le tout en espace total $\mathcal{O}\big(|F| + \min(n^2, k|F|)\big)$.

Le troisième chapitre est l'article [BB12b] soumis à *Information Processing Letters*, qui présente un algorithme d'énumération à délai (quasi-[8]) linéaire. Si notre algorithme s'avère finalement une variante généralisée de celui présenté dans [BFS95], on n'a été en mesure de comprendre leur article qu'après en avoir réinventé les points essentiels.

Aussi notre travail est finalement, encore une fois, d'ordre pédagogique. À ce titre, nous lui avons trouvé une forme réellement satisfaisante : l'algorithme sans doute le plus compliqué de ce mémoire est présenté en moins de dix pages, en trois étapes pertinentes, et donne (presque) le sentiment d'être facile.

8 Voir à ce propos les remarques en fin de première partie.

Chapitre 1

Logique propositionnelle et énumération

> Dans le monde réellement renversé, le vrai est un moment du faux.
>
> Guy Debord – *La société du spectacle* §9

Sommaire

1 Logique propositionnelle 6
1.1 De la logique propositionnelle aux formules CNF . . 7
1.2 Satisfaisabilité et énumération 8
1.3 Des cas particuliers faciles 10
2 Satisfaisabilité généralisée 12
2.1 Décision : dichotomie de Schaefer 12
2.2 Énumération : dichotomie de Creignou & Hébrard . 13
3 Des formules affines 14
3.1 Définitions 15
3.2 Décision et précalcul 15
3.3 Énumération 17

Ce chapitre

Ce chapitre vise déjà à formaliser la problématique esquissée dans l'introduction, et à poser quelques notations. Dans une première section, on introduit, d'abord très informellement, la logique propositionnelle, son lien intime avec les formules CNF, et des faits extrêmement classiques à propos des formules CNF. Si le ton de cette section est très naïf, en revanche la manière de présenter l'algorithme de Davis et Putnam [DP60, DLL62] et la remarque sur l'importance de l'ordre sont tout sauf innocentes.

La seconde section énonce la dichotomie de Creignou et Hébrard [CH97], ce qui constitue la formalisation annoncée de la problématique. Enfin, une section consacrée aux formules affines permet de traiter d'emblée ce cas facile, là où un chapitre est consacré à chacun des autres cas.

Ce chapitre est une ébauche très sommaire d'un éventuel futur papier de synthèse sur cette partie entière, et est clairement le chapitre le moins abouti de ce mémoire. Espérons que le lecteur saura s'en accommoder et pourra rapidement passer aux autres chapitres.

1 Logique propositionnelle

La logique propositionnelle est la plus simple des logiques. Il ne s'agit pas tant d'une logique que d'une proto-logique, c'est-à-dire un élément constitutif d'une vraie logique, comme la logique du premier ordre, considérée dans la seconde partie de ce mémoire.

En effet, la logique propositionnelle prend comme éléments indivisibles des propositions, et non des objets.

Par exemple, si je dis :

- Il n'y a pas de fumée sans feu
- Il y a de la fumée
- Conséquemment, il y a du feu.

C'est un raisonnement qui se tient. Pour exactement les mêmes raisons

- Il n'y a pas de poisson sans bicyclette
- Il y a un poisson
- Conséquemment, il y a une bicyclette.

se tient tout autant. Il n'est aucunement question ici de la validité de faits dans le monde, il est uniquement question de validité du discours dans sa forme, indépendamment du sens qu'il peut avoir.

Dans un premier temps, nous introduisons le formalisme de cette logique.

Définition 1 (formule propositionnelle) Une proposition — que l'on appelle formule, nommée en général F — est l'une des expressions suivantes :

- la conjonction de deux formules F_1 et F_2 qui signifie F_1 *et* F_2 (vraie quand F_1 *et* F_2 sont vraies), notée $F_1 \wedge F_2$,
- la disjonction de deux formules F_1 et F_2 qui signifie F_1 *ou* F_2 (vraie quand F_1 est vraie, *ou* quand F_2 est vraie, ou les deux), notée $F_1 \vee F_2$,
- l'implication de deux formules F_1 et F_2 qui signifie F_1 *implique* F_2 (vraie quand « *si* F_1 est vraie, F_2 aussi »), notée $F_1 \to F_2$,
- la négation d'une formule F_1 qui signifie *il est faux que* F_1 (vraie quand F_1 est fausse), noté $\neg F_1$,
- la constante $\bot$ (toujours faux) ou la constante $\top$ (toujours vrai),
- ou alors une proposition indivisible, encore nommée atome ou variable propositionnelle, en général désignée par une lettre minuscule.

Identifions les propositions dans ces exemples. Le premier cas peut être réécrit :

(*il est faux que* ((il y a de la fumée) *et il est faux que* (il y a du feu))) *et* (il y a de la fumée) *impliquent* (il y a du feu).

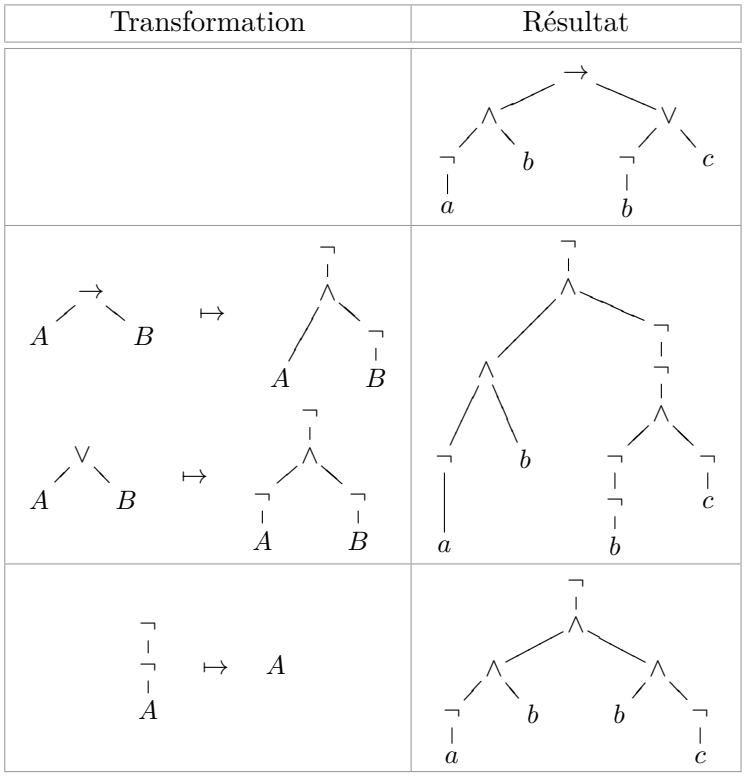
Si on écrit feu pour signifier *il y a du feu* et fumée pour signifier *il y a de la fumée*, et que l'on applique les notations mentionnées, alors les choses deviennent plus lisibles :

$$(\neg(\text{fumée} \wedge \neg\text{feu})) \wedge \text{fumée}) \to \text{feu}$$

Il s'agit là d'une écriture largement inspirée des expressions arithmétiques ; ce n'est pas un hasard si la logique propositionnelle porte aussi le nom de calcul propositionnel.

À cette formule nous associons plusieurs questions :

- Est-elle valide ? (toujours vraie, indépendamment du sens des propositions) Ex : Le ciel est bleu ou le ciel n'est pas bleu.



● **Figure 1.1 :** Un exemple de transformation d'une formule propositionnelle en CNF. Étape 1 : on normalise la formule.

- Est-elle satisfaisable ? (Parfois vraie, suivant le sens des propositions) Ex : Le ciel est bleu.
- Est-elle contradictoire ? (toujours fausse, indépendamment du sens des propositions) Ex : Le ciel est bleu et il n'est pas bleu.

Une proposition valide ne dit rien, on l'appelle encore lapalissade ou tautologie, ou encore théorème. Une proposition satisfaisable divise les mondes possibles en deux catégories : ceux pour lesquels elle est effectivement vraie, et ceux pour lesquels elle ne l'est pas ; par exemple, les lois de la physique sont des propositions satisfaisables, car vérifiées dans notre monde, mais non valides, sans quoi elles ne diraient rien. Une proposition contradictoire n'est vraie dans aucun monde.

La négation d'une proposition contradictoire est valide, une proposition qui n'est pas contradictoire est satisfaisable.

1.1 De la logique propositionnelle aux formules CNF

On va maintenant voir que l'on peut construire, pour toute formule propositionnelle, une formule normalisée qui lui est équivalente, et a une taille proportionnelle.

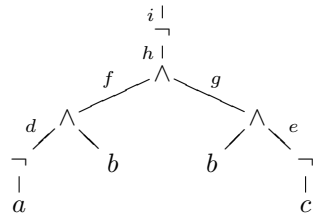
Pour ce, on va présenter le procédé à travers un exemple. Considérons une formule :

$$(\neg a \wedge b) \rightarrow (\neg b \vee c)$$

On peut se la représenter comme un arbre, que l'on peut transformer en un second arbre qui, lui, n'utilise que $\wedge$ et $\neg$ comme connecteurs. On peut simplifier : $\neg\neg F = F$. Un exemple est présenté figure 1.1.

Toujours sur le même exemple, on peut ajouter des variables repré-

sentant les valeurs intermédiaires :



Finalement, les modèles[1] de F sont les modèles de :

$$(i = \neg h) \wedge (h = f \wedge g) \wedge (f = d \wedge b) \wedge (d = \neg a) \wedge (g = b \wedge e) \wedge (e = \neg c) \wedge i$$

Comme $F = F'$ et $(F \to F') \wedge (F' \to F)$ sont équivalents, on a :

$$\begin{aligned}
a = \neg b \quad &\Leftrightarrow \quad (a \to \neg b) \wedge (\neg b \to a) \\
&\Leftrightarrow \quad (\neg a \vee \neg b) \wedge (a \vee b) \\
a = b \wedge d \quad &\Leftrightarrow \quad (a \to (b \wedge c)) \wedge ((b \wedge c) \to a) \\
&\Leftrightarrow \quad (a \to b) \wedge (a \to c) \wedge (\neg b \vee \neg c \vee a) \\
&\Leftrightarrow \quad (\neg a \vee b) \wedge (\neg a \vee c) \wedge (\neg b \vee \neg c \vee a)
\end{aligned}$$

ainsi on a pu construire une formule normalisée — sous forme d'une conjonction de disjonctions — qui est logiquement équivalente[2] à notre formule de départ, et de taille proportionnelle. On introduit maintenant plus formellement cette forme normalisée.

1.2 Satisfaisabilité et énumération

Définition 2 (CNF, clause, litteral, modèle) On appelle *Formule Normale Conjonctive* (*CNF*) une conjonction de clauses. Par extension, on assimilera cette conjonction à un ensemble, aussi $\emptyset$ désignera la conjonction vide, qui est donc équivalente à $\top$.[3]

Une *clause* est une disjonction de littéraux. Un littéral est soit directement une variable x, dont on dit alors qu'il est un *littéral positif*, soit une variable niée $\neg x$, dont on dit alors qu'il s'agit d'un *littéral négatif*. On appelle *littéral opposé* d'un littéral l, noté $\bar{l}$, le littéral portant sur la même variable mais qui est positif si l est négatif et réciproquement. Par extension, on assimilera une clause à un ensemble de littéraux ; en particulier, la clause vide $\emptyset$ désignera la disjonction vide, qui est donc équivalente à $\bot$.

Ainsi donc, une CNF de la forme $\emptyset$ est tautologique, alors qu'une formule de la forme $\{\emptyset\} \cup ...$, c'est-à-dire contenant la clause vide est contradictoire ; en particulier, une CNF sans variables est soit $\emptyset$, la tautologie, soit $\{\emptyset\}$, la contradiction sans variables.

On appelle *solution* ou *modèle* d'une formule F, portant sur les variables $\mathit{Vars}(F)$, une affectation des variables $f : \mathit{Vars}(F) \to \{\bot, \top\}$ telle que, si on remplace chacune des variables x par son affectation

1 On appelle modèle une valuation des variables qui rend la formule vraie. L'ensemble des modèles constitue en quelque sorte le « sens » d'une formule.

2 Ces deux formules sont équivalentes dans le sens suivant : à tout modèle de la première correspond un unique modèle « prolongé » de la seconde, c'est-à-dire qui coïncide sur les variables communes ; et à tout modèle de la seconde correspond un unique modèle « projeté » de la première, c'est-à-dire qui ne conserve que l'affectation des variables présentes dans la première formule.

3 Les deux constantes $\top$ et $\bot$ désignent respectivement « toujours vrai » ou encore 1 et « toujours faux » ou encore 0. Astuce mnémotechnique : $\top$ pour « $\top$oujours vrai », $\bot$ pour « $\bot$ourquoi pas ? », voir citation d'ouverture du chapitre.

$f(x)$ dans F, alors la formule est équivalente à $\top$, i.e. chacune des clauses est vraie. Autrement dit, un modèle d'une formule F de la forme $F(x_1,...,x_n)$, c'est-à-dire portant sur les variables $x_1,...,x_n$ est un n-uplet $(a_1,a_2,...,a_n) \in \{\bot,\top\}^n$ tel que $F(a_1,...,a_n)$ est vraie. On assimile le modèle m d'une formule $F(x_1,...,x_n)$ à une conjonction de littéraux C telle que :

- toutes les variables x_i de F sont présentes dans cette conjonction,
- on ne peut trouver un littéral et son opposé dans cette conjonction,
- et l'unique modèle de la conjonction est un modèle de F.

On assimile aussi cette conjonction à un *ensemble*. En particulier, les algorithmes décriront les modèles sous cette forme ; ainsi $\emptyset$ désignera le modèle vide de la CNF sans variables et satisfaisable, i.e. la CNF tautologique $\emptyset$.

Exemple 1 Considérons la formule :

$$F(a,b,c,d) = (a \vee \neg c \vee b) \wedge (\neg a) \wedge (\neg c \vee \neg d) \wedge (b \vee d)$$

F est un exemple de formule CNF. Dans ce cas, $(a \vee \neg c \vee b)$ est une clause, et a et $\neg c$ sont des littéraux.

Par abus de notation, on écrira $x \in F$ pour signifier que la variable x apparaît dans la formule F.

On a vu que l'on peut toujours supposer une formule propositionnelle sous forme CNF. On va maintenant montrer comment on peut procéder à la décision de l'existence d'un modèle et à l'énumération des modèles d'une telle formule. Précisons d'abord la notion de complexité de l'énumération.

Définition 3 (énumération) On appelle *algorithme d'énumération* un algorithme qui va successivement écrire toutes les solutions d'un problème, ici les modèles d'une formule.

On appelle *délai* le temps qui sépare l'écriture d'une solution de l'écriture de la précédente, ainsi que le temps entre l'écriture de la dernière solution et l'arrêt. Si on considère le délai au sens large, le début de l'exécution est considéré comme l'écriture d'une 0^{e} solution fictive. Sinon, on dira que l'algorithme d'énumération a un précalcul et le temps de ce précalcul sera explicitement mentionné.

L'algorithme 1.1 est un algorithme très classique, dû à Davis et Putnam [DP60, DLL62] pour trouver un modèle d'une formule CNF, que nous avons ici adapté pour qu'il énumère ses modèles au lieu d'en trouver seulement un. Bien noter que tant le temps de décision que le délai d'énumération sont susceptibles d'avoir des valeurs de l'ordre de $\mathcal{O}(2^n|F|)$ où n est le nombre de variables de la formule F et $|F|$ la taille de F définie comme la somme des cardinalités respectives de ses clauses.

Définition 4 (clause unitaire, littéral pur) On appelle aussi *clause unitaire* une clause singleton. On appelle *littéral pur* d'une CNF F un littéral l tel que $\bar{l}$ n'apparaisse dans aucune clause de F.

La correction du traitement associé aux clauses unitaires et aux littéraux purs dans l'algorithme tient à ces deux faits :

- Si $\{l\} \in F$, alors les modèles de F sont exactement les modèles de la formule simplifiée F' construite à partir de F en remplaçant l par $\top$ et $\bar{l}$ par $\bot$, auxquels on rajoute le littéral l.

```
Affecte(F,l) :
    pour C ∈ F :
        si l ∈ C :
            Retirer C de F
        si l̄ ∈ C :
            Retirer l̄ de C
    renvoyer F
ÉnumèreDP(F,V) :
    si ∅ ∈ F :
        arrêter
    si V = ∅ :
        si F = ∅ :
            produire ∅
    ou alors si {l} ∈ F :
        pour m ∈ ÉnumèreDP(Affecte(F, l), V∖{l, l̄}) :
            produire m ∪ {l}
    ou alors si l est pur dans F :
        pour m ∈ ÉnumèreDP(Affecte(F, l), V∖{l, l̄}) :
            produire m ∪ {l}
            si m ∪ {l̄} satisfait F :
                produire m ∪ {l̄}
    sinon
        pour m ∈ ÉnumèreDP(Affecte(F, ¬x), V∖{x}) :
            produire m ∪ {¬x}
        pour m ∈ ÉnumèreDP(Affecte(F,  x), V∖{x}) :
            produire m ∪ {  x}
```

● **Algorithme 1.1 :** Variante de l'algorithme de Davis et Putnam pour l'énumération des modèles d'une formule F portant sur les variables V.

- Si l est pur dans F, les modèles de F sont les modèles de $F \wedge \{l\}$ auxquels on peut ajouter certains [4] modèles m' obtenus à partir des modèles m de $F \wedge \{l\}$ en inversant le signe de l, i.e. $m' = m \smallsetminus \{l\} \cup \{\bar{l}\}$.

L'intérêt de ces traitements est exposé ultérieurement.

1.3 Des cas particuliers faciles

Définition 5 (classes de CNF) On dit d'une clause qu'elle est :
zéro-valide [5] si elle est vide ou contient au moins un littéral négatif,
un-valide si elle est vide ou contient au moins un littéral positif,
Horn si elle contient au plus un littéral positif,
anti-Horn si elle contient au plus un littéral négatif,
bijonctive si elle contient au plus deux littéraux.
On dit d'une CNF qu'elle est zéro-valide (resp. un-valide, Horn, anti-Horn, bijonctive) lorsque qu'elle est uniquement composée de clauses zéro-valides (resp. un-valides, Horn, anti-Horn, bijonctives).

4 On construit tous les m' comme indiqué, on ne conserve que ceux qui sont effectivement des modèles. Ce faisant, on est sûr d'avoir tous les modèles.

5 Une clause zéro-valide qui comporte des variables est satisfaite si on les instancie toutes à zéro, i.e. $\bot$ avec les notations choisies. Le même raisonnement tient pour les clauses un-valides, avec « un » signifiant $\top$.

▶ **Théorème 1 (facilité de certaines classes de CNF)**
Toutes ces classes de CNF sont faciles à décider.

Démonstration. Pour chaque classe :

zéro-valide : il suffit de tester la présence d'une clause vide. S'il n'y en a pas, $(\bot,...,\bot)$ est une solution.

un-valide : il suffit de tester la présence d'une clause vide. S'il n'y en a pas, $(\top,...,\top)$ est une solution.

Horn : On verra que l'algorithme de Davis et Putnam présenté le décide en temps linéaire (voir ci-dessous).

anti-Horn : Ce cas se ramène facilement au cas Horn.

bijonctive : Ce cas fait l'objet du prochain chapitre.

On montre que les CNF de Horn sont décidables en temps linéaire par l'algorithme 1.1.

Notons d'abord qu'une formule CNF zéro-valide est bel et bien décidée par l'algorithme 1.1 le temps d'une descente (sans backtrack) qui est borné par un temps quadratique (linéaire, même).

Une formule de Horn qui n'est pas zéro-valide contient nécessairement une clause de taille un, dite unitaire, qui porte un littéral positif. Alors, cette variable est instanciée à $\top$, et le reste de la formule est satisfaisable si et seulement si la formule originelle l'était. Au bout d'un certain nombre d'instanciations à $\top$ par propagation unitaire, soit la clause vide apparaît, soit la formule est devenue zéro-valide.

Noter que l'algorithme 1.1 trouve un modèle de Horn en temps polynomial, linéaire, même, à bien y regarder. ◀

Remarque 2 (importance de l'ordre) L'ensemble $\mathcal{E}$ de formules, défini ainsi :

$$\begin{aligned}\mathcal{E}_0 &= \{\emptyset\} \cup \{F \mid \emptyset \in F\}\\ \mathcal{E} &= \mathcal{E}_0 \cup \left\{F \;\middle|\; \exists l \text{ (l est pur dans } F \vee \{l\} \in F) \wedge \text{Affecte}(F,l) \in \mathcal{E}\right\}\end{aligned}$$

a la propriété suivante : toute formule de cet ensemble est traitée par l'algorithme 1.1 d'une manière telle que la dernière alternative (le *sinon*) n'est jamais considérée. Cela signifie que la formule est décidable uniquement par propagation unitaire et par instanciation des littéraux purs. Il s'ensuit que les modèles de la formule sont énumérés à délai linéaire $\mathcal{O}(|F|)$, dans un ordre lexicographique correspondant à l'élimination des clauses unitaires et des littéraux purs.

En revanche, si l'on impose l'ordre lexicographique d'énumération, trouver le premier modèle devient NP-difficile. Pour le prouver, considérons une CNF quelconque, par exemple :

$$F\ (x_1,...,x_n) = \bigwedge_i C_i$$

Définissons à partir de F une formule $F' \in \mathcal{E}$ de taille $\mathcal{O}(|F|)$:

$$F'(x_0,...,x_n) = \bigwedge_i C_i \vee x_0$$

Il est facile de prouver que trouver le premier modèle de F' dans l'ordre lexicographique $(x_0,...,x_n)$ permet de décider F, un problème NP-difficile.

Le choix de l'ordre d'énumération n'est donc pas anodin : on peut, suivant le choix de l'ordre, avoir une énumération à délai linéaire ou bien une recherche NP-difficile du premier élément. Les algorithmes d'énumération efficace pour les formules de Horn et les formules bijonctives, qui font l'objet des deux prochains chapitres, tireront précisément leur efficacité d'un choix judicieux de l'ordre d'énumération.

2 Satisfaisabilité généralisée

On se place maintenant dans le cadre élaboré par Schaefer [Sch78] (Voir [CKS01, BCRV03, BCRV04].) où l'on fixe la forme des clauses, c'est-à-dire que l'on dispose, pour écrire une formule, d'un ensemble de « clauses généralisées » fixé, que l'on fait librement porter sur les variables de notre choix. Séparer ainsi le type de clauses que l'on s'autorise, et l'usage que l'on en fait permet précisément de définir des « classes » de problèmes.

Il est important de comprendre que les résultats de facilité pour les CNF s'étendent au problème de satisfaisabilité généralisé, et que, réciproquement, les résultats de difficulté pour le problème de satisfaisabilité généralisé s'étendent aux formules CNF.

2.1 Décision : dichotomie de Schaefer

Le problème de satisfaisabilité généralisée est un problème paramétré par une structure $\mathcal{S}$. Une structure est ici un ensemble de relations sur le domaine booléen. Soit une structure $\mathcal{S}$ constituée des relations $R_1,...,R_n$ d'arités respectives $a_1,...,a_n$. Le problème $\textsc{Sat}(\mathcal{S})$ (resp. $\textsc{ÉnumSat}(\mathcal{S})$) consiste, étant donnée une formule définie comme une conjonction où les atomes sont de la forme $R_i(x_{f(j,1)},...,x_{f(j,a_i)})$, à déterminer si la formule est satisfaisable (resp. énumérer ses modèles).

Par exemple, si on définit la structure $\mathcal{S}$ ainsi :

$$\mathcal{S} = \{R_{oe} = \{(\top,\bot,\bot),(\bot,\top,\bot),(\bot,\bot,\top)\}\}$$

alors $\textsc{Sat}(\mathcal{S})$ consiste à décider l'existence d'un modèle d'une formule de la forme :

$$\bigwedge_{1\leq i\leq m} R_{oe}(x_{f(i)},x_{g(i)},x_{h(i)})$$

où R_{oe} est une relation booléenne ternaire qui n'est vraie que lorsqu'un seul de ses arguments est vrai.

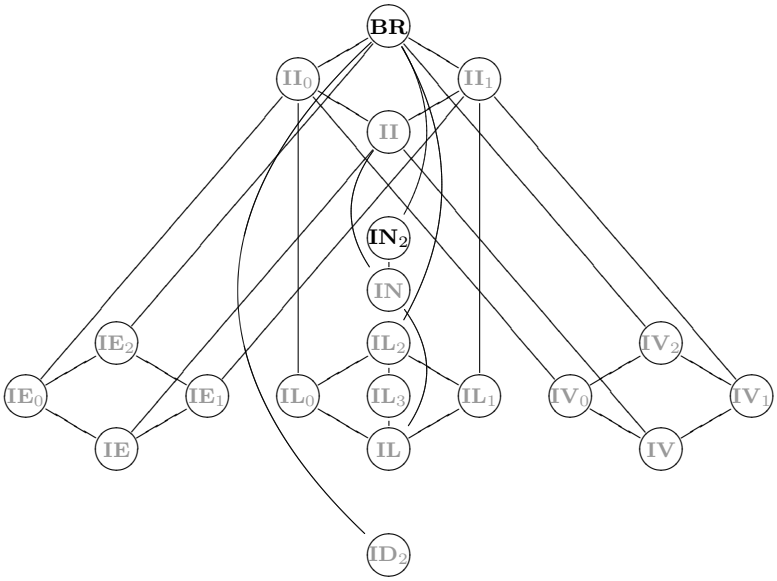
Définition 6 (classes de structures) On dit d'une relation R qu'elle est de la classe ... lorsqu'elle est :

- **zéro-valide** exprimable par une CNF zéro-valide
- **un-valide** exprimable par une CNF un-valide
- **Horn** exprimable par une CNF Horn
- **anti-Horn** exprimable par une CNF anti-Horn
- **bijonctive** exprimable par une CNF bijonctive
- **affine** exprimable par une conjonction de clauses $l_1 \oplus l_2 \oplus ...$ où $a \oplus b \Leftrightarrow (a,b) \in \{(\bot,\top),(\top,\bot)\}$

On dit d'une structure $\mathcal{S}$ qu'elle est zéro-valide (resp. un-valide, Horn, anti-Horn, bijonctive, affine) lorsque *chacune* des relations qu'elle contient est zéro-valide (resp. un-valide, Horn, anti-Horn, bijonctive, affine).

▶ **Théorème 2 (Schaefer [Sch78])**

Le problème $\textsc{Sat}(\mathcal{S})$ est dans P si $\mathcal{S}$ est dans une des classes suivantes : zéro-valide, un-valide, Horn, anti-Horn, bijonctive, affine. Dans les autres cas, il est NP-complet. ◀



Les symboles grisés correspondent aux classes dans P, les symboles noirs aux classes NP-complètes. Les classes sont désignées ainsi :
$\mathbf{II_0}$ zéro-valide
$\mathbf{II_1}$ un-valide
$\mathbf{IE_2}$ Horn
$\mathbf{IV_2}$ anti-Horn
$\mathbf{IN_2}$ complémentive (i.e. l'inversion d'un modèle est un modèle)
$\mathbf{IL_2}$ affine
$\mathbf{ID_2}$ bijonctive
Les autres sont définies par l'intersection des classes les contenant.

● **Figure 1.2 :** Dichotomie de Schaefer présentée dans le treilli de Post.

2.2 Énumération : dichotomie de Creignou & Hébrard

▶ **Théorème 3 (Creignou et Hébrard [CH97])**

Le problème ÉnumSat($\mathcal{S}$) est à délai polynomial si $\mathcal{S}$ est dans une des classes suivantes :

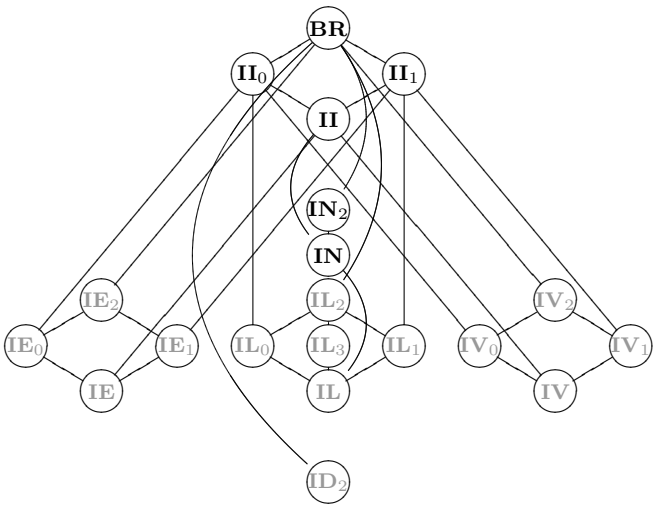
- Horn
- anti-Horn
- bijonctive
- affine

Dans les autres cas, le problème Sat*($\mathcal{S}$), qui consiste à décider l'existence d'un modèle autre que ($\bot$,...,$\bot$) et que ($\top$,...,$\top$), est NP-complet, ce qui implique que le délai d'énumération est non polynomial sous l'hypothèse P≠NP. ◀

Le résultat de facilité est donné par l'algorithme 2. On le présente ici parce qu'il est le seul algorithme simple de réduction du problème d'énumération au problème de décision. Le délai d'énumération est le temps de la décision multiplié par le nombre de variables.

Cet algorithme sera encore plus pertinent dans la seconde partie de ce mémoire, dans le cadre de laquelle il établira un lien fort entre complexité de la décision et complexité de l'énumération.

Remarque 3 (de l'algo de Creignou et Hébrard) Cet algorithme générique, dans sa forme, nous montre une propriété notable : on peut l'utiliser pour énumérer les modèles dans n'importe quel ordre lexicographique. On verra dans les deux chapitres suivants qu'en exploitant



● **Figure 1.3 :** Dichotomie de Creignou et Hébrard, même légende que la figure précédente, à la différence que les symboles grisés correspondent aux classes énumérables à délai polynomial, et les symboles noirs aux classes pour lesquelles ce n'est pas le cas, sous l'hypothèse P≠NP.

Énum($F(x_1,...,x_n)$, i) :
    si $F(x_1,...,x_n)$ est satisfaisable :
        si $i > n$:
            produire $\emptyset$
        sinon
            pour $m \in$ Énum($F(x_1,...,x_n) \wedge \neg x_i$, $i+1$) :
                produire $m \cup \{\neg x_i\}$
            pour $m \in$ Énum($F(x_1,...,x_n) \wedge \ x_i$, $i+1$) :
                produire $m \cup \{\ x_i\}$

● **Algorithme 1.2 :** Algorithme de Creignou et Hébrard [CH97]. On l'appelle initialement avec la formule en premier argument et 1 en second argument.

un certain ordre structurel de la formule, on peut obtenir des délais bien meilleurs, pour un ordre d'énumération qui est une sorte [6] d'ordre lexicographique. L'enjeu des prochains chapitres est donc de s'adapter à la formule au lieu de lui imposer un ordre arbitraire, en contrepartie de quoi on pourra procéder à une énumération réellement efficace.

3 Des formules affines

Le problème de la décision et du calcul d'une solution d'une formule affine est un cas particulier de la résolution d'un système d'équations linéaires, celui où le corps de base est $F_2 = (\mathbb{Z}/2\mathbb{Z}, \oplus, \odot)$. Un tel système est résolu en temps mn^2 (pour m équations à n inconnues) par l'algorithme de Gauss qui permet de décider s'il a au moins une solution, et si oui d'en trouver une.

On va présenter cet algorithme sous une forme adaptée à la logique, et on va montrer qu'à partir d'une telle solution, on peut énumérer toutes les solutions à délai $\mathcal{O}(n)$, ceci en utilisant l'énumération d'un code de Gray.

6 Voir les remarques en fin de partie sur l'ordre d'énumération.

Par souci de complétude et d'homogénéité des définitions et des notations, on redéfinit d'abord les notions de base adaptées aux formules affines, puis on décrit et justifie les deux algorithmes successifs de décision DécideAffine et d'énumération ÉnumAffine.

3.1 Définitions

Puisque l'on se place dans un cadre logique, on notera encore $\bot$ et $\top$ les éléments 0 et 1 du domaine de base.

Définition 7 (formule affine) Une clause affine est de la forme :

1. $x_{i_1} \oplus x_{i_2} \oplus ... \oplus x_{i_k} \oplus \top$, ou bien :
2. $x_{i_1} \oplus x_{i_2} \oplus ... \oplus x_{i_k}$,

où $k \geq 0$ et les x_{i_j} sont des variables telles que $i_1 < i_2 < ... < i_k$. Là encore, on confondra une clause avec l'ensemble de ses opérandes.

Une formule affine est une conjonction de clauses affines que l'on assimile aussi à l'ensemble de ses clauses.

Exemple 4 La clause de type 1 est l'ensemble $\{x_{i_1}, x_{i_2}, ..., x_{i_k}, \top\}$; en particulier, si $k = 0$, la clause $\{\top\}$ est tautologique ; à l'inverse, la clause vide $\emptyset$ (de type 2) est contradictoire.

3.2 Décision et précalcul

On se réfère ici à l'algorithme 1.3. L'algorithme DécideAffine décide pour une formule affine si elle a un modèle, et si oui en construit un. Soit F une formule affine à n variables $x_1, ..., x_n$. Pour chacune des variables x_i, on va choisir [7] une clause $C[i]$ qui, moralement, représente les contraintes qui pèsent sur x_i. La propriété fondamentale de la conjonction des clauses $C[i]$ que l'on va construire est d'être équivalente à la formule F toute entière :

$$F \Leftrightarrow \bigwedge_{1 \leq i \leq n} C[i]$$

Autrement dit, la conjonction des $C[i]$ est une forme normalisée de F.

Pour ce faire, notons d'abord le fait suivant : soit C_1 et C_2 deux clauses affines quelconques. Alors :

$$C_1 \wedge C_2 \Leftrightarrow C_1 \wedge (C_2 \oplus \top \oplus \top) \Leftrightarrow C_1 \wedge (C_2 \oplus C_1 \oplus \top)$$

Par conséquent, l'opération :

$$F \leftarrow \cup \begin{array}{l} \{C \oplus C[i] \oplus \top \mid x_i \in C \in F\} \\ \{C \in F \mid x_i \notin C\} \end{array}$$

dans l'algorithme DécideAffine maintient l'équivalence entre F avant modification et $C[i] \wedge F$ après modification. L'intérêt de cette opération est de faire disparaître la variable x_i de F : les clauses qui ne portaient pas x_i restent identiques, et les clauses qui portaient x_i ne le portent plus (puisque $x_i \oplus x_i$ peut être supprimé). Il s'ensuit qu'à l'issue de l'exécution de la première boucle de la procédure DécideAffine($F(x_1, ..., x_n)$), F ne contient plus de variables. On a l'alternative :

- Soit F contient la clause vide, et est donc contradictoire ;
- sinon, F ne contient que des clauses tautologiques, donc la conjonction des $C[i]$ est équivalente à la formule initiale. Dans ce cas, pour chacune des variables $x_1, ..., x_n$:

7 N'importe quelle clause contenant x_i fera l'affaire.

```
DécideAffine(F(x_1,...,x_n)) :
    pour i ∈ [1,...,n] :
        si x_i ∉ F :
            C[i] ← {⊤}
        sinon
            C[i] ← Choisir C ∈ F telle que x_i ∈ C
            F ← ∪ {C ⊕ C[i] ⊕ ⊤ | x_i ∈ C ∈ F}
                  {C ∈ F | x_i ∉ C}
    si ∅ ∈ F :
        renvoyer ∅,∅
    pour i ∈ [n,...,1] :
        si C[i] ≠ {⊤} :
            pour x_j ∈ C[i] | i ≠ j :
                C[i] ← C[i] ⊕ C[j] ⊕ ⊤
    pour i ∈ [1,...,n] :
        si ⊤ ∈ C[i] :
            V[i] ← ⊥
        sinon
            V[i] ← ⊤
    renvoyer V,C
Gray(m) :
    si m > 0 :
        Gray(m − 1)
        écrire m
        Gray(m − 1)
ÉnumAffine(F(x_1,...,x_n)) :
    V, C ← DécideAffine(F(x_1,...,x_n))
    si V = ∅ :
        arrêter
    pour i ∈ {1,...,n} | C[i] ≠ {⊤} :
        R[i] ← {j | x_i ∈ C[j]}
    [π_1,...,π_k] ← [i ∈ [1,...,n] | C[i] = {⊤}]
    pour i ∈ Gray(k) :
        V[π_i] ← V[π_i] ⊕ ⊤
        pour j ∈ R[π_i] :
            V[j] ← V[j] ⊕ ⊤
        écrire V[1],...,V[n]
```

● **Algorithme 1.3 :** Décision et énumération des formules affines.

- soit $C[i] = \{\top\}$, on dira alors que x_i est un *paramètre*,
- soit $C[i]$ contient x_i et $\forall x_j \in C[i]\ j \geq i$.

Autrement dit, s'il n'est pas contradictoire, le système d'équations associé aux $C[i]$ est triangulaire, de dimension $n - k$ où k est le nombre de paramètres et il est clair que la formule F possède 2^k solutions, correspondant à toutes les affectations possibles des variables paramètres, les autres variables x_i étant alors déterminées de façon unique par les $C[i]$.

La seconde boucle permet d'assurer la propriété suivante : chaque $C[i]$ non tautologique ne contient que x_i et des paramètres x_j avec $j > i$.

Enfin, la troisième boucle calcule une première solution de la formule F, celle où tous les paramètres ont été affectés à $\bot$.

3.3 Énumération

On note $x_{\pi_1},...x_{\pi_k}$ la liste des variables paramètres. Les modèles de F peuvent être générés ainsi : pour chaque affectation $(a_1,...,a_k) \in \{\bot,\top\}^k$ des variables $x_{\pi_1},...,x_{\pi_k}$, simplifier les clauses $C[i]$ de manière à déterminer la valeur de x_i. En procédant ainsi, le délai est la somme des tailles des $C[i]$, donc $\mathcal{O}(n^2)$. Pour obtenir un délai $\mathcal{O}(n)$ on utilise des codes de Gray.

Définition 8 (code de Gray) Un code de Gray de l'ensemble $\{\bot,\top\}^n$ est un ordre de cet ensemble dans lequel deux éléments successifs ne diffèrent que d'un bit i.e. l'élément qui succède à $w_n...w_1$ est de la forme $w_n...w_{i+1}\overline{w_i}w_{i-1}...w_1$.

La fonction Gray de l'algorithme 1.3 énumère les indices dont le bit est modifié. Par exemple, Gray(3) va produire successivement :

Génération d'indices				Combinaison
				$\bot\bot\bot$
		Gray(1)	1	$\bot\bot\top$
	Gray(2)	2	2	$\bot\top\top$
		Gray(1)	1	$\bot\top\bot$
Gray(3)	3	3	3	$\top\top\bot$
		Gray(1)	1	$\top\top\top$
	Gray(2)	2	2	$\top\bot\top$
		Gray(1)	1	$\top\bot\bot$

On va donc énumérer les solutions à délai $\mathcal{O}(n)$ à partir d'une énumération des 2^k interprétations possibles des paramètres selon un code de Gray.

Pour une variable paramètre x_i, on note $R[i]$ l'ensemble des indices des variables x_j telles que $x_i \in C[j]$. On rappelle que $x_{\pi_1},...,x_{\pi_k}$ désignent les variables paramètres.

Chaque pas de la boucle finale de l'algorithme ÉnumAffine ne modifiant l'interprétation que d'un seul paramètre x_{π_i}, les seules autres variables dont les interprétations sont également modifiées sont les x_j d'indice $j \in R[\pi_i]$, que l'on obtient par un parcours séquentiel de $R[\pi_i]$. Ceci garantit un délai $\mathcal{O}(n)$ pour l'énumération des 2^k solutions de F.

Conclusion

On a précisé formellement la problématique esquissée dans l'introduction, et traité le cas facile des formules affines, pour lesquelles on a prouvé que l'énumération se fait à délai $\mathcal{O}(n)$ après le temps nécessaire à la procédure standard de décision.

Pour traiter la question soulevée, il ne reste « plus qu'à » traiter les cas bijonctif et Horn. C'est l'objet des deux autres chapitres de cette partie.

Chapitre 2

Formules bijonctives et Horn-renommages

Donnez le si
Il pousse un if
Faites le tri
Il naît un arbre
Jouez au bridge, et le pont s'ouvre
Engloutissant les canons les soldats
Au fond, au fond affectionné
De la rivière rouge
Ah, oui les Anglais sont bien dangereux.

Boris Vian – *Je voudrais pas crever*

Sommaire

1 Des formules bijonctives 20
1.1 Décision et précalcul 20
1.2 Énumération 22
2 Des formules Horn-renommables 28
2.1 Décision 28
2.2 Forme canonique 31
2.3 Énumération 33

Introduction

Les formules bijonctives sont bien connues pour être décidables en temps linéaire [EIS76] [1], ce qui demeure le cas même si on considère des formules quantifiées [APT79]. Feder a montré [Fed94] que l'on peut énumérer ses modèles à délai $\mathcal{O}(n)$ où n est le nombre de variables, après un précalcul linéaire. Une première section est consacrée à expliciter l'algorithme que Feder explique de manière vraiment lapidaire, et que l'on modifie de façon à énumérer dans un ordre lexicographique, sans altérer le délai.

Par ailleurs, l'intérêt des formules de Horn n'est plus à démontrer. Pour l'étendre un peu, la notion de formule Horn-renommable a été proposée, qui permet d'agrandir cette classe (en sacrifiant la stabilité par conjonction). Cette notion a fait l'objet d'un intérêt marqué

1 Le problème de satisfaisabilité des formules bijonctives est aussi appelé 2Sat.

```
V(G) :
    renvoyer ⋃_{(i→j)∈G} {i, j}
Parcours(G,i,n) :
    si ordre[i] = 0 :
        ordre[i] ← −1
        pour k ∈ {j | i → j ∈ G} :
            n ←Parcours(G,k,n)
        ordre[i] ← n
        n ← n + 1
    renvoyer n
ParcoursInverse(G,i,n) :
    si cfc[i] = 0 :
        cfc[i] ← n
        pour k ∈ {j | j → i ∈ G} :
            ParcoursInverse(G,k,n)
CFC(G) :
    n ← 1
    pour i ∈ V(G) :
        ordre[i] ← 0
        cfc[i] ← 0
    pour i ∈ V(G) :
        n ←Parcours(G,i,n)
    pour i ∈ V(G) par ordre[i] décroissant :
        ParcoursInverse(G,i,i)
    G ← {cfc[i] → cfc[j] | i → j ∈ G}
    renvoyer {i → j ∈ G | i ≠ j}
```

● **Algorithme 2.1 :** Construction des Composantes Fortement Connexes.

[Asp80, Lew78, LS89, MM85, CCH+90]. Tester si une formule est Horn-renommable se fait en temps linéaire [Héb94], tester si elle admet un unique Horn-renommage se fait également en temps linéaire [Héb95].

Nous allons dans une seconde section généraliser *simultanément* ces résultats sur le Horn-renommage et sur l'énumération des solutions des formules bijonctives grâce à un algorithme qui énumère les $k-1$ premiers Horn-renommages d'une formule F à n variables composée de clauses de longueur au plus k à délai linéaire $\mathcal{O}(|F|)$, puis énumère les suivants à délai $\mathcal{O}(n)$, tout ceci en espace total $\mathcal{O}\big(|F| + \min(n^2, k|F|)\big)$.

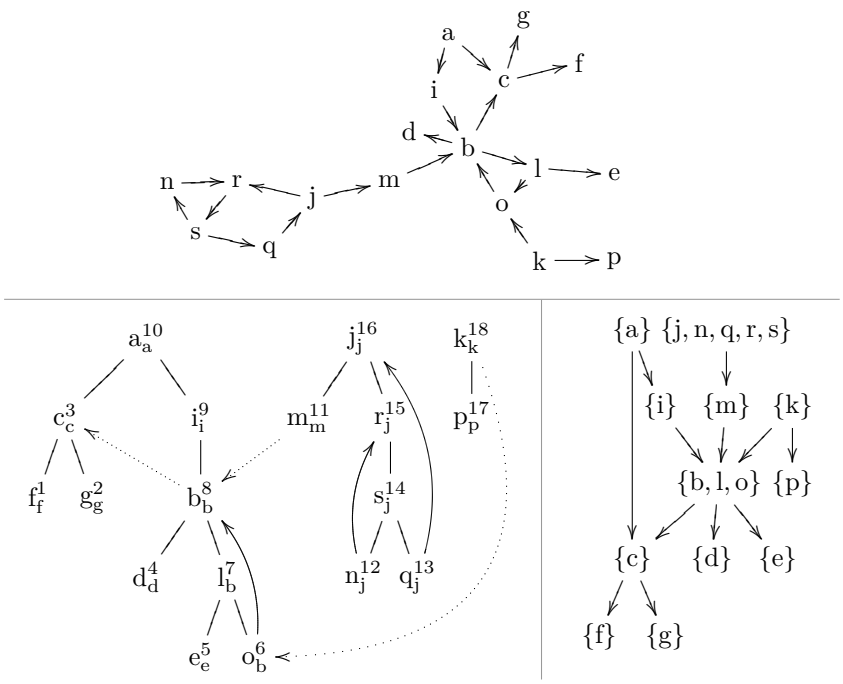
1 Des formules bijonctives

1.1 Décision et précalcul

Dans cette partie, on rappelle comment trouver une solution d'une formule bijonctive, non seulement parce qu'il s'agit du problème le plus simple, mais aussi parce que cela constitue une étape nécessaire à l'énumération.

1.1.1 Décision et première solution

On considère le « graphe d'implication » $\mathcal{G}(F)$ d'une formule bijonctive F. Ses sommets sont les littéraux l et leur négations $\bar{l}$. Les arcs de $\mathcal{G}(F)$ sont tous les arcs de la forme $\bar{l_1} \to l_2$ ou de la forme $\bar{l_2} \to l_1$, pour toutes



● **Figure 2.1 :** Exemple de calcul de composantes fortement connexes.
En haut, le graphe original, dont on souhaite déterminer les composantes fortement connexes.
À gauche, l'arbre construit par la recherche en profondeur. La première passe permet de déterminer les ordres postfixés, indiqués en exposant.
La seconde passe permet de déterminer la classe d'équivalence de chaque sommet, indiquée en indice.
À droite, le graphe acyclique résultant de la contraction des composantes fortement connexes du graphe original.

les clauses $l_1 \vee l_2$ de F. Autrement dit :

$$\mathcal{G}(F) = \left\{ \overline{l_1} \to l_2 \,|\, (l_1 \vee l_2) \in F \text{ ou } (l_2 \vee l_1) \in F \right\}$$

Ce graphe peut être vu comme une conjonction de clauses implicatives équivalente à la formule F, aussi on assimile dans la suite la formule et son graphe d'implication.

Des algorithmes désormais classiques (voir [Tar71, Tar72, CLR91, AHU74, Weg02]) permettent de construire les composantes fortement connexes (CFC) d'un graphe en temps linéaire. Se reporter à l'algorithme 2.1 et à la figure 2.1. Si une composante fortement connexe de $\mathcal{G}(F)$ contient à la fois un littéral et son opposé, on voit immédiatement que la formule est contradictoire puisque la nécessaire équivalence entre les deux littéraux opposés est impossible. Si ce n'est pas le cas, on va voir que F est satisfaisable et surtout comment énumérer ses solutions. On contracte les composantes fortement connexes, et on suppose donc qu'aucune ne contient simultanément un littéral et son opposé ; le graphe d'implication ainsi obtenu ne contient qu'un seul littéral par CFC, et est donc acyclique.

Choisissons un ordre anti-topologique $>$ quelconque des littéraux, ce que l'on sait faire en temps linéaire, par simple effeuillage. Compte-tenu de la symétrie du graphe d'implication, on peut toujours s'arranger pour que l'ordre soit symétrique, i.e. $l < x_j \Leftrightarrow \bar{l} > \neg x_j$, avec l ou bien de la forme x_i ou bien de la forme $\neg x_i$.

Plus précisément, on instancie chaque variable x à $\top$ (vrai) si $\neg x >$

x, et à $\bot$ (faux) sinon (i.e. $x > \neg x$ puisqu'il n'y a pas égalité). Il suffit de trouver la fin d'un ordre topologique comprenant au moins une occurrence de chaque variable. Il est facile de vérifier qu'une telle instanciation des variables ne viole aucune clause : par conséquent cette instanciation est un modèle.

On conserve précieusement notre solution : elle sera utile à l'énumération.

1.1.2 Suite du précalcul : normalisation

Maintenant que l'on dispose d'une solution, on renomme chacune des variables vraies dans la solution en son opposé. Le problème reste exactement le même : les solutions obtenues devront seulement faire l'objet d'exactement le même renommage afin d'être celles de la formule initiale. La formule ainsi transformée a acquis une propriété intéressante : elle est $\bot$-valide, c'est–à–dire vraie dans l'interprétation où chaque variable est mise à $\bot$ (faux). Cela équivaut aussi à dire que chaque clause de F contient au moins un littéral négatif, et est donc d'une des deux formes suivantes :

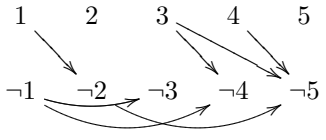
1. $x_i \vee \neg x_j$, que l'on écrira $\neg x_i \to \neg x_j$;
2. $\neg x_i \vee \neg x_j$, équivalente à $x_i \to \neg x_j$ et à $x_j \to \neg x_i$.

Il est facile, en temps linéaire, de renuméroter les variables de F dans un ordre topologique du graphe d'implication, c'est–à–dire un ordre respectant les clauses de la forme 1, autrement dit tel que l'on ait toujours $i < j$ pour toute clause $\neg x_i \to \neg x_j$ de F. À la suite de cette renumérotation, chaque clause de la forme 2 sera écrite sous la forme d'une implication respectant cet ordre : $x_i \to \neg x_j$ avec $i < j$. Au final, F est équivalente à une conjonction de clauses de la forme

1. $\neg x_i \to \neg x_j$ ou
2. $x_i \to \neg x_j$,

avec toujours $i < j$. Cette propriété fondamentale permettra d'obtenir une énumération extrêmement efficace.

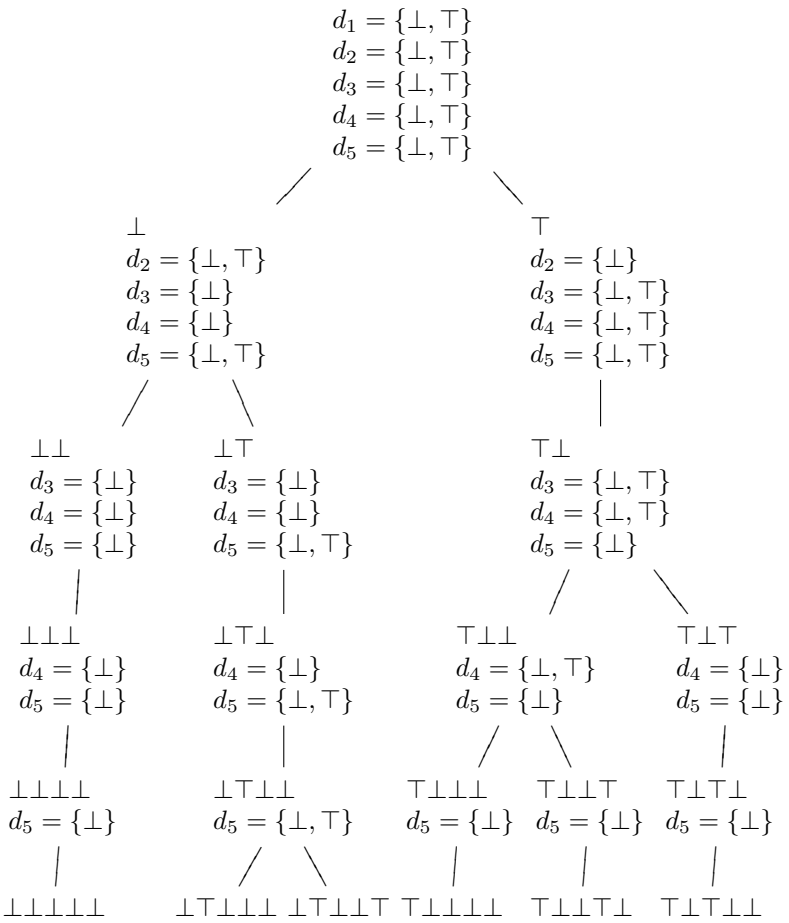
Par exemple, la formule $\bot$-valide $(\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_3) \wedge (x_1 \vee \neg x_4) \wedge (x_2 \vee \neg x_5) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_3 \vee \neg x_5) \wedge (\neg x_4 \vee \neg x_5)$ a pour clauses de la forme 1 les implications $\neg x_1 \to \neg x_3$, $\neg x_1 \to \neg x_4$, et $\neg x_2 \to \neg x_5$ qui respectent déjà l'ordre croissant des indices, et qui ne demande donc aucune renumérotation des variables. La formule F peut se réécrire sous la forme de son graphe d'implication $\mathcal{G}(F)$ ainsi simplifié (plus de double représentation des clauses) où tout arc (implication) va de gauche à droite (indices croissants) et de haut en bas (chaque implication a pour conclusion un littéral négatif) :



Pour simplifier, dans ce schéma comme dans les algorithmes qui vont suivre, on confond chaque variable avec son indice.

1.2 Énumération

On a vu comment trouver une solution en temps linéaire, et on a utilisé cette solution pour normaliser la formule et lui donner des propriétés que nous allons maintenant exploiter.



● **Figure 2.2 :** Exemple d'exécution de l'algorithme naïf.

En se ramenant à cette situation normalisée, on a bien entendu modifié légèrement l'ensemble des solutions, mais il devrait être clair pour le lecteur que ce nouvel ensemble de solutions est en bijection avec l'ancien, et qu'il est très facile [2] de calculer une solution de la formule originelle à partir de son image dans l'ensemble des solutions de la nouvelle formule.

Nous allons présenter l'algorithme d'énumération par étapes successives. On part de l'algorithme le plus naïf, que l'on modifie successivement jusqu'à aboutir à l'algorithme de Feder que l'on améliorera finalement.

1.2.1 Approche type contraintes (naïve)

Un algorithme de recherche naïf typique de la programmation par contraintes exécuté sur la formule de l'exemple précédent est figuré par l'arbre de recherche de la figure 2.2. La notation d_i y représente le domaine de la variable i.

Un premier constat peut être fait : comme ce sont toujours les valeurs $\top$ (vrai) qui sont retirées des domaines — puisque les conclusions des implications sont toujours négatives, et que l'on instancie les variables dans le même ordre croissant— on peut simplifier le filtrage (et son annulation) en se contentant d'un compteur de « tentatives » de retrait

2 Cela se fait en temps linéaire avec la taille d'une solution.

```
Initialisation :
    pour 1 ≤ i ≤ n :
        a[i] ← 0
    Énum(1)
Énum(i) :
    si i = n + 1 :
        écrire v
        arrêter
    v[i] ← ⊥
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    Énum(i + 1)
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] − 1
    si a[i] = 0 :
        v[i] ← ⊤
        pour (i → ¬j) ∈ F :
            a[j] ← a[j] + 1
        Énum(i + 1)
        pour (i → ¬j) ∈ F :
            a[j] ← a[j] − 1
```

● **Algorithme 2.2 :** Remplacement du domaine par l'activité.

```
Initialisation :
    pour 1 ≤ i ≤ n :
        a[i] ← 0
        v[i] ← ⊥
    a[n + 1] ← 0
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    Énum(1)
Énum(i) :
    si i = n + 1 :
        écrire v
        arrêter
    Énum (min ({k > i | a[k] = 0}))
    v[i] ← ⊤
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] − 1
    pour ( i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    Énum (min ({k > i | a[k] = 0}))
    pour ( i → ¬j) ∈ F :
        a[j] ← a[j] − 1
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    v[i] ← ⊥
```

● **Algorithme 2.3 :** Parcours avec contraction des branches unaires.

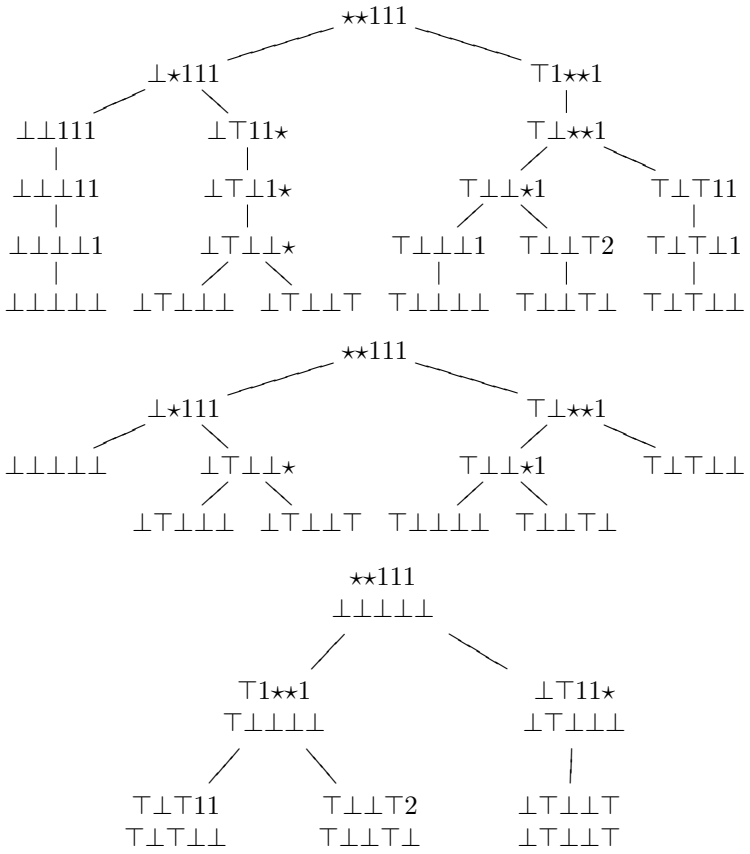
d'une valeur $\top$. On appellera *activité* de i le compteur associé à la variable i, qui sera noté $a[i]$. C'est le nombre d'implications où $\neg i$ est la conclusion et dont la prémisse est vraie, donc de la forme $j \to \neg i$ avec j à $\top$ ou $\neg j \to \neg i$ avec j à $\bot$. Intuitivement, c'est le nombre d'implications qui obligent la variable i à être mise à $\bot$. La condition pour mettre la variable i à $\top$ devient donc $a[i] = 0$; on s'est ainsi débarrassé de l'utilisation de domaines explicites.

En simplifiant ainsi l'algorithme naïf, on obtient l'algorithme 2.2. Ce nouvel algorithme est identique à l'algorithme naïf en termes de parcours : la figure 2.2 décrit aussi son exécution, seulement il est plus simple. Une version avec les activités est donnée figure 2.3.

Comme le précédent, cet algorithme a une propriété particulière qui découle justement de ce que seule la valeur $\top$ peut être retirée du domaine : tout nœud de l'arbre (binaire) de recherche mène à au moins une solution évidente, la solution partielle actuelle complétée avec des $\bot$. En conséquence de quoi, entre deux solutions successives, l'algorithme parcourt $2n$ nœuds au pire. Le délai polynomial est donc évident.

En fait, ce délai est linéaire, c'est–à–dire $\mathcal{O}(|F|)$. Pour le constater, il suffit de noter que lors d'une descente — qui aboutit toujours à la production d'une solution — chaque clause est considérée au plus une fois, et qu'il en est de même pour le *backtrack* correspondant.

Cet algorithme de base va subir dans la suite une série de transformations, dont il s'agira de montrer qu'elles sont équivalentes. Ce seront, en fait, des transformations de l'arbre de recherche par équivalence.



● **Figure 2.3 :** Mise en parallèle de l'exécution des algorithmes 2.2, 2.3 et 2.4. Les valeurs déjà instanciées et les activités sont mises bout à bout —la valeur des variables non instanciées, et l'activité des variables instanciées présentant peu d'intérêt— et dans l'ordre des variables. $\star$ est utilisé pour signifier 0 en mettant en avant la particularité de cette valeur.
L'algorithme 2.5 a le même arbre de recherche que l'algorithme 2.4, et produit la séquence 1#3#34#412#5#521# qui nous donne bien successivement : $\top\bot\bot\bot\bot, \top\bot\top\bot\bot, \top\bot\bot\top\bot, \bot\top\bot\bot\bot, \bot\top\bot\bot\top, \bot\top\bot\bot\bot$..

1.2.2 Parcours de l'arbre de recherche

Algorithme 2.3 Une première étape importante est de ne permettre d'effectuer un choix que sur la prochaine variable pouvant être mise à la valeur $\top$. Pour ce faire, nous exploitons le fait que $(\bot,...,\bot)$ est une solution, on complète donc le précalcul de manière à partir du cas où toutes les valeurs sont fausses ($\bot$), c'est-à-dire que l'on va donc effectuer lors du précalcul tous les filtrages liés à la valeur $\bot$ d'une variable.

Ainsi, on pourra « sauter » tous les filtrages liés à des variables d'activité non nulle, et donc se préoccuper uniquement de la prochaine variable d'activité nulle.

Cette manière de faire affecte la manière dont on procède au filtrage : au lieu d'effectuer des filtrages liés au choix des valeurs, on effectue des filtrages liés au changement par rapport à la valeur $\bot$. Ainsi, lorsque l'on ré-instancie une variable à $\top$, il s'agit à la fois d'effectuer le filtrage lié à cette valeur, mais aussi d'annuler celui lié à la valeur $\bot$; à la fin de chaque appel, on restaurera la situation où la variable courante vaut $\bot$.

Une variable fictive $n+1$ est introduite, dont l'activité est toujours nulle, qui sert de garde pour le calcul du minimum de l'ensemble $\{k > i \mid a[k] = 0\}$.

```
ÉnumFeder :
    pour 1 ≤ i ≤ n :
        a[i] ← 0
        v[i] ← ⊥
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    Énum(0)
Énum(i) :
    v[i] ← ⊤
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] − 1
    pour ( i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    écrire v
    pour i_0 ∈ {k > i | a[k] = 0} :
        Énum(i_0)
    pour ( i → ¬j) ∈ F :
        a[j] ← a[j] − 1
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    v[i] ← ⊥
```

● **Algorithme 2.4 :** Parcours n-aire avec production à chaque noeud.

```
Énum(i) :
    écrire +i
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] − 1
    pour ( i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    si profondeur paire :
        écrire #
    pour i_0 ∈ {k > i | a[k] = 0} :
        Énum(i_0)
    si profondeur impaire :
        écrire #
    pour ( i → ¬j) ∈ F :
        a[j] ← a[j] − 1
    pour (¬i → ¬j) ∈ F :
        a[j] ← a[j] + 1
    écrire −i
```

● **Algorithme 2.5 :** Algorithme final.

Algorithme 2.4 Au lieu d'effectuer une recherche binaire sur les valeurs, et de produire des solutions en arrivant sur les feuilles de l'arbre binaire de choix, nous allons exploiter la consistance toujours maintenue pour produire les solutions à chaque nœud. En effet, on sait en arrivant à chaque nœud que l'instance courante des variables est une solution. On va donc la produire sur le champ, et s'assurer que les appels récursifs produisent de nouvelles solutions. Pour ce, au lieu d'effectuer le choix binaire, on choisit la plus petite variable modifiable que l'on puisse mettre à $\top$. Le choix ne porte donc plus sur les deux valeurs possibles de la variable courante, mais bien sur le nombre de variables actives que l'on va délibérément laisser à $\bot$. En termes de transformation d'arbre de recherche, on utilise la classique équivalence entre arbre binaire étiqueté aux feuilles et arbre général (d'arité quelconque) étiqueté aux nœuds.

Noter l'introduction d'une variable fictive 0 qui permet de parcourir toutes les variables d'activité nulle.

Algorithme 2.5 Tout appel entraîne une écriture. Entre l'appel et la première écriture s'écoule un temps $\mathcal{O}(n)$ où n est le nombre de variables. Entre la dernière écriture et le retour, on aimerait bien prouver qu'il ne s'écoule que $\mathcal{O}(n)$, mais c'est sans espoir.

Le problème n'est pas la descente dans l'arbre — qui coûte $\mathcal{O}(n)$ à chaque nouvelle solution — mais bien la remontée : on est bien incapable de connaître le nombre de retours avant une nouvelle descente. Et si on inverse la génération et le parcours ? Eh bien on a le problème exactement inverse.

Fort heureusement, Feder [Fed94] a trouvé une astuce — simple et étonnante — qui permet de trouver le compromis recherché. Les

« étages » pairs, on génère avant de parcourir les fils ; les « étages » impairs, après. Cela fonctionne : tous les trois nœuds au pire, on génère une solution. D'une manière générale, on peut énumérer les nœuds d'un arbre n-aire à délai constant (précisément, au plus tous les trois nœuds visités) en utilisant cette astuce : si le noeud est de profondeur paire, l'écrire, puis parcourir ses fils ; dans le cas contraire parcourir ses fils puis l'écrire. On peut démontrer par l'absurde que, effectivement, cela fonctionne. La question est développée dans l'annexe.

Il nous reste à produire les solutions sous forme différentielle, c'est-à-dire sous forme d'une liste d'indices de variables ayant changé de valeur depuis la dernière solution produite. Pour ce, le plus simple est de produire l'indice d'une variable à chaque fois qu'elle change, et de produire un caractère de fin de liste avant le parcours des fils si la profondeur est paire, ou après dans le cas contraire. De la sorte, on est sûr de la correction, et qu'un caractère de fin de liste # est bien produit au pire après trois caractères autres.

Noter qu'au début de l'exécution et à la toute fin, l'algorithme produit respectivement un $+0$ et -0 parasites, qui sont sans conséquence.

Déjà, on va préciser un point qui n'est pas explicite dans l'écriture de l'algorithme. Le parcours des fils, qui sont en fait les variables d'activité nulle, doit être fait par indice *décroissant*. Cela aura son importance pour les questions de complexité. L'autre implication importante est que, dans ce cas, le parcours de l'arbre se fait dans un ordre où les solutions sont découvertes dans l'ordre lexicographique. Sur la figure 2.3, ce fait s'exprime par la contrainte que le parcours se fait en profondeur *mais de droite à gauche*.

1.2.3 Finalement

Complexité L'algorithme 2.5 est correct. On veut montrer qu'il est à délai $\mathcal{O}(n)$, où n est le nombre de variables.

La seule difficulté est de maintenir à jour une liste doublement chaînée représentant l'ensemble des variables d'activité nulle. Pour ce, on suppose l'existence d'une liste globale l_g (partagée par tous les appels récursifs), à jour au moins pour les variables d'indices supérieurs à i lors de l'exécution de Énum(i). Lors du filtrage, on constitue une liste locale l_l. Pour parcourir les fils, on le fait dans l'ordre inverse des variables, en intégrant l_l à l_g à la volée. Ainsi, chaque opération de fusion donne lieu à un appel récursif, qui produit une solution : ainsi le coût est constant par solution produite. La liste l_g utilisée par l'appel récursif valide l'hypothèse de récurrence : elle est bien à jour sur sa partie utile.

Post-traitement L'algorithme 2.6 énumère clairement dans l'ordre lexicographique : il suit le parcours de l'arbre de recherche de l'algorithme de Feder, mais cette fois, les instructions # ne servent pas à indiquer le modèle à écrire, mais servent uniquement d'horloge.

L'algorithme de Feder produit chaque symbole de sortie au bout d'un temps $\mathcal{O}(n)$, et tous les trois symboles au plus, un symbole # apparaît. À ce moment, notre post-traitement doit produire un nouveau modèle, qui est le modèle qui suit le dernier modèle écrit dans l'ordre de parcours préfixe.

Il suffit donc, pour prouver le délai $\mathcal{O}(n)$, que le temps qui sépare la lecture du # et l'écriture du modèle soit lui aussi $\mathcal{O}(n)$. Les opérations

```
ÉnumLex(F) :
    f ← FileVide
    m ← (⊥,...,⊥)
    pour o ∈ ÉnumFeder(F) :
        si o = # :
            o′ ← Défile(f)
            tant que o′ est de la forme −i :
                m[i] ← ⊥
                o′ ← Défile(f)
            m[o′] ← ⊤
            produire m
        sinon
            f ← Enfile(f, o)
```

Algorithme 2.6 : Énumération dans l'ordre lexicographique à délai $\mathcal{O}(n)$.

élémentaires entre les deux sont toutes clairement en temps constant, il faut donc seulement prouver le fait que la boucle itère $\mathcal{O}(n)$ fois.

Ce fait est évident si on voit bien que les éléments de la forme $-i$ correspondent à des remontées dans l'arbre, il ne peut donc y en avoir successivement plus que la profondeur de l'arbre, qui est au plus n.

On a prouvé que le délai est $\mathcal{O}(n)$, mais on ne prouve pas la taille $\mathcal{O}(n)$ de la file. On l'affirme tout de même, en reportant une preuve claire de cette affirmation à une version ultérieure (publication à soumettre).

2 Des formules Horn-renommables

Le problème de décision du Horn renommage a été prouvé linéaire [Héb94] ; le problème d'unicité aussi [Héb95] ; nous franchissons une nouvelle étape en prouvant que son énumération se fait à délai linéaire $\mathcal{O}(|F|)$ (et en espace total $\mathcal{O}(|F|)$), ce qui généralise de fait les précédents résultats : notre algorithme est encore une fois une généralisation des algorithmes précédents connus, qui ne sont *a posteriori* que des versions abrégées de notre algorithme.

De plus, nous fournissons un autre algorithme, dont le précalcul est légèrement plus coûteux : il prend un temps $\mathcal{O}\big(n|F|\big)$ et un espace $\mathcal{O}\big(|F| + \min(n^2, k|F|)\big)$, où k est la longueur maximale d'une clause, mais dont le délai est bien meilleur : $\mathcal{O}(n)$.

On propose finalement un algorithme synthétique qui énumère les k premiers Horn-renommages à délai linéaire $\mathcal{O}(|F|)$, puis les suivants à délai $\mathcal{O}(n)$, tout ceci en espace total $\mathcal{O}\big(|F| + \min(n^2, k|F|)\big)$ où k est la longueur maximale d'une clause.

2.1 Décision

Cette section vise à trouver *un* Horn-renommage d'une formule normale conjonctive quelconque. Pour ce faire, on réduit linéairement le problème de recherche d'un Horn-renommage à la recherche d'une solution d'une formule bijonctive.

On reprend ici ce qu'énonce [Héb94].

2.1.1 Définitions

On rappelle quelques définitions.

Une *formule normale conjonctive* (CNF) est une conjonction de clauses $F = C_1 \wedge ... \wedge C_m$, où chaque clause C_i est une disjonction de littéraux. Une telle formule est dite *de Horn* si chacune des clauses C_i contient au plus un littéral positif.

Définition 9 (Horn-renommage) On appelle *renommage* des variables $x_1,...,x_n$ d'une CNF F un n-uplet $r = (r_1,...,r_n) \in \{\bot, \top\}^n$. Pour chaque variable x_i, on note :

$$r(x_i) = \begin{cases} x_i \text{ si } r_i = \bot \\ \neg x_i \text{ si } r_i = \top \end{cases}$$

Le renommage r s'étend naturellement aux littéraux, aux clauses de F, et à la formule F elle-même.

Définition 10 (Horn-renommage) Soit F une formule normale conjonctive à m clauses et n variables $x_1,...,x_n$, de la forme :

$$F = \bigwedge_{i=1}^{m} \bigvee_{j=1}^{n(i)} l_i^j$$

où $n(i)$ est la cardinalité de la i-ième clause $C_i = \bigvee_{j=1}^{n(i)} l_i^j$ et l_i^j est le j-ième littéral de C_i. On dit qu'un renommage r des variables $x_1,...,x_n$ est un *Horn-renommage* de F si la formule renommée $r(F)$ est une formule de Horn : chaque clause $r(C_i) = \bigvee_{j=1}^{n(i)} r(l_i^j)$ contient au plus un littéral positif. On note alors $r \in \text{HornRen}(F)$.

On note HornRen (resp. EnumHornRen) le problème de savoir si une formule CNF possède ou non un Horn-renommage (resp. le problème de l'énumération de ses Horn-renommages).

2.1.2 Réduction de HornRen à Auplus1Sat

Définition 11 (Auplus1) On appelle Auplus1 l'opérateur sur les booléens qui est satisfait ($\top$) si et seulement si au plus un de ses arguments vaut $\top$ i.e. $\text{Auplus1}(l_1,...,l_k) \Leftrightarrow (\overline{l_1} \wedge ... \wedge \overline{l_k}) \vee \bigvee_i \left(\overline{l_1} \wedge ... \wedge \overline{l_{i-1}} \wedge l_i \wedge \overline{l_{i+1}} \wedge ... \wedge \overline{l_k}\right)$.

▶ ***Lemme 1 (HornRen = Auplus1Sat)*** Soit $\bigwedge_{i=1}^{m} \bigvee_{j=1}^{n(i)} l_i^j$ une CNF à n variables $x_1,...,x_n$. On a, $\forall r \in \{\bot, \top\}^n$:

$$r \in \text{HornRen}\left(\bigwedge_{i=1}^{m} \bigvee_{j=1}^{n(i)} l_i^j\right) \quad \Leftrightarrow \quad r \models \bigwedge_{i=1}^{m} \text{Auplus1}_{1 \leq j \leq n(i)} \overline{l_i^j}$$

Par conséquent :

$$\begin{aligned} \text{HornRen} &= \text{Auplus1Sat} \\ \text{EnumHornRen} &= \text{EnumAuplus1Sat} \end{aligned}$$

où, bien entendu, les problèmes Auplus1Sat et EnumAuplus1Sat désignent respectivement les problèmes de satisfaisabilité et celui de l'énumération des modèles d'une conjonction de clauses Auplus1.

Démonstration. Par définition du Horn-renommage. Bien noter que r est vu comme un renommage dans la partie gauche de l'équivalence, c'est-à-dire que $\bot$ signifie « reste en l'état » et $\top$ signifie « est inversé » ; alors que dans la partie droite de l'équivalence, il s'agit d'une affectation des variables. ◀

2.1.3 Réduction linéaire de Auplus1Sat à 2Sat

Cette réduction permettra de trouver, en temps linéaire, une solution au problème Auplus1Sat, ce qui est un préliminaire nécessaire à l'énumération ; malheureusement, l'extension de cette réduction à l'énumération ne sera pas possible, car le nombre de solutions n'est pas préservé dans cette réduction, aussi on en utilisera une autre.

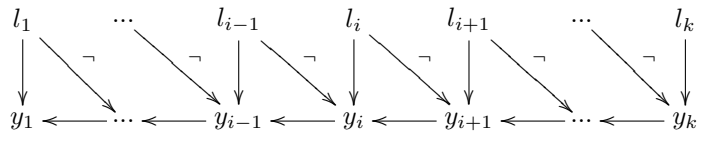
On transforme une conjonction de clauses Auplus1 en une conjonction de clauses bijonctives utilisant de nouvelles variables existentiellement quantifiées. Ceci se fait par la réduction élémentaire suivante.

▶ ***Lemme 2 (réduction linéaire)*** Pour toute conjonction de clauses Auplus1, il existe une formule bijonctive dont certaines variables sont quantifiées existentiellement, équivalente et de taille linéaire.

Démonstration. L'idée de cette réduction est d'introduire, pour chacune des clauses Auplus1, des variables « compteurs » y_i qui indiquent s'il existe un l_j avec $j \geq i$ valant $\top$. Dans ce cas, la mise à $\top$ de l_i est prohibée ; ainsi un seul l_i peut être mis à $\top$. Formellement, la clause Auplus1$(l_1,...,l_k)$ donnera lieu à la formule bijonctive :

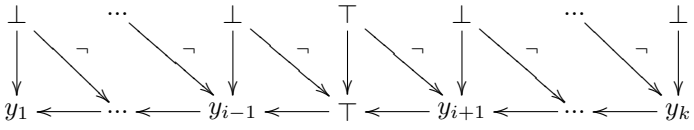
$$(l_k \to y_k) \wedge \bigwedge_{i=1}^{k-1} (l_i \to y_i) \wedge (l_i \to \neg y_{i+1}) \wedge (y_{i+1} \to y_i)$$

Visuellement, la formule bijonctive ressemble à :

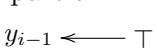


où $a \xrightarrow{\neg} b$ signifie $a \to \neg b$. Prouvons l'équivalence entre la clause Auplus1 et la formule bijonctive associée pour toute interprétation m des littéraux $l_1,...,l_k$.

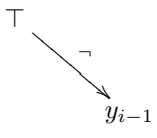
⇒ Supposons $m \models$ Auplus1(l_i). On a deux cas. S'il existe un littéral l_{i_0}, nécessairement unique, tel que $m(l_{i_0}) = \top$, alors on est dans la situation suivante :



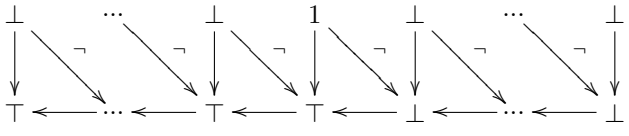
La partie :



nous montre comment va se propager la valeur $\top$ pour tous les y_j avec $j < i$, alors que :



nous prouve que y_{i+1} vaudra $\bot$, qui lui se propagera en remontant la chaîne d'implication $y_j \to y_{j-1}$ qui est équivalente à $\neg y_{j-1} \to \neg y_j$. Finalement, on a bien :



Dans l'autre cas, i.e. lorsque tous les l_i sont à $\bot$, il s'agit de trouver l'affectation des y_i telle que :

$$y_1 \longleftarrow \cdots \longleftarrow y_i \longleftarrow y_{i+1} \longleftarrow \cdots \longleftarrow y_k$$

On voit tout de suite les solutions : l'affectation des y_i est soit de la forme $(\top,...,\top,\bot,...,\bot)$ soit ils sont tous à $\bot$.

$\Leftarrow$ Par transitivité des implications $l_i \to y_i$, $y_i \to y_{i-1}$, et $y_i \to \overline{l_{i-1}}$, on tire les implications $l_i \to (y_1 \wedge ... \wedge y_i)$ et $l_i \to (\overline{l_1} \wedge ... \wedge \overline{l_{i-1}})$, c'est-à-dire Auplus1($\{l_i \mid 1 \leq i \leq n\}$). ◀

Ce lemme justifie donc la transformation de toute clause Auplus1 à k arguments en une conjonction de moins de $3k$ clauses bijonctives.

Cette transformation s'étend immédiatement en une réduction linéaire $F \mapsto F'$ du problème Auplus1Sat au problème 2Sat. On note aussi que toute solution (modèle) de la formule bijonctive F' ainsi obtenue à partir d'une donnée F du problème Auplus1Sat fournit immédiatement une solution pour la formule initiale F : il suffit pour cela de restreindre le modèle de F' aux variables x_i de F. Finalement, on a prouvé la proposition suivante.

▶ **Théorème 4 (Horn-renommage [Héb94])**
On peut décider le problème HornRen en temps linéaire et dans le même temps en trouver une solution particulière, s'il en existe une. ◀

2.2 Forme canonique

On montre maintenant comment exploiter le résultat de la section précédente pour procéder à une énumération efficace. Pour ce, on va dans un premier temps procéder à une normalisation du problème en temps linéaire.

À partir de cette section, les résultats sont nouveaux.

2.2.1 Simplifions

Soit F une formule normale conjonctive (CNF) à m clauses et n variables. On vient de voir comment trouver un Horn-renommage de F, $r_0 \in \{\bot,\top\}^n$ (s'il en existe un) en temps linéaire. Soit F' la formule renommée par r_0, c'est-à-dire, en considérant les formules comme des fonctions booléennes, $F'(x_1,...,x_n) = F((x_1,...,x_n) \oplus_n r_0)$, où :

$$(x_1,...,x_n) \oplus_n (y_1,...,y_n) \underset{\text{définition}}{=} (x_1 \oplus y_1,...,x_n \oplus y_n)$$

où $(x_1,...,x_n) \in \{\bot,\top\}^n$, $(y_1,...,y_n) \in \{\bot,\top\}^n$ et où $\oplus$ désigne le « ou exclusif » (xor). Par définition, F' est de Horn et tout Horn-renommage r' de F' est tel que $r' \oplus_n r_0 = r$ est un Horn-renommage de F. Cela fonctionne dans l'autre sens, i.e $r \oplus r_0 = r'$ (tout renommage est involutif), aussi énumérer les Horn-renommages d'une formule normale conjonctive revient exactement à énumérer les Horn-renommages d'une formule de Horn.

Dans la suite, on suppose donc que F est une formule de Horn à m clauses et n variables, c'est-à-dire de la forme :

$$\bigwedge_{i=1}^{l} \left(x_{C(i,0)} \vee \bigvee_{j=1}^{n(i)} \neg x_{C(i,j)} \right) \wedge \bigwedge_{i=l+1}^{m} \bigvee_{j=1}^{n(i)} \neg x_{C(i,j)}$$

où $n(i)$ est le nombre de littéraux négatifs de la i-ième clause et $C(i,j)$ est l'indice de la j-ième variable négative de la i-ième clause, l'indice $C(i,0)$ désignant bien sûr la variable positive unique le cas échéant.

2.2.2 Réduction et composantes fortement connexes

Par le lemme 1, les Horn-renommages de la formule de Horn F de la forme générale donnée ci-dessus sont exactement les modèles de la formule :

$$F_0 = \bigwedge_{i=1}^{l} \text{Auplus1}\left(\neg x_{C(i,0)}, \left(x_{C(i,j)}\right)_{1\leq j\leq n(i)}\right) \wedge \bigwedge_{i=l+1}^{m} \text{Auplus1}\left(\left(x_{C(i,j)}\right)_{1\leq j\leq n(i)}\right)$$

On note que F_0 peut se réécrire de façon équivalente sous la forme de la formule « mixte » suivante :

$$F' = \bigwedge_{i=1}^{l}\bigwedge_{j=1}^{n(i)}\left(x_{C(i,j)} \to x_{C(i,0)}\right) \wedge \bigwedge_{i=1}^{m} \text{Auplus1}_{1\leq j\leq n(i)}\left(x_{C(i,j)}\right)$$

De toutes ces remarques, on déduit immédiatement le lemme suivant.

▶ ***Lemme 3*** Soit $F' = F_1 \wedge F_2$ et $F'' = F_1 \wedge F'_2$ où :

$$F_1 = \bigwedge_{i=1}^{l}\bigwedge_{j=1}^{n(i)}\left(x_{C(i,j)} \to x_{C(i,0)}\right)$$

$$F_2 = \bigwedge_{i=1}^{m} \text{Auplus1}_{1\leq j\leq n(i)}\left(x_{C(i,j)}\right)$$

$$F'_2 = \bigwedge_{i=1}^{m}\bigwedge_{1\leq j<k\leq n(i)}\left(x_{C(i,j)} \to \neg x_{C(i,k)}\right)$$

On note les quatre points suivants :

1. Les modèles de la « formule mixte » F' sont exactement les Horn-renommages de la formule de Horn F.
2. Les formules F_2 et F'_2 étant équivalentes, la conjonction $F' = F_1 \wedge F_2$ est équivalente à la formule bijonctive $F'' = F_1 \wedge F'_2$.
3. La clause F_1 occupe clairement une place $\mathcal{O}(|F|)$.
4. Les CFC (composantes fortement connexes du graphe d'implication) de la formule bijonctive F'' sont les CFC [3] de F_1 ; en particulier, toute CFC de F'' est constituée de littéraux de même signe. ◀

En dépit du fait que la taille de la formule F'_2 et donc celle de $F'' = F_1 \wedge F'_2$ est $\mathcal{O}(|F|^2)$, les points 3 et 4 du lemme 3 justifient qu'on puisse calculer les CFC (du graphe d'implication) de F'' en temps $\mathcal{O}(|F_1|)$ donc $\mathcal{O}(|F|)$, et supposer que chaque CFC qui n'est pas réduite à un élément est un ensemble de littéraux positifs.

▶ ***Lemme 4*** On peut transformer en temps linéaire la formule F' en une formule équivalente de même forme mais acyclique, c'est-à-dire de graphe d'implication sans circuit.

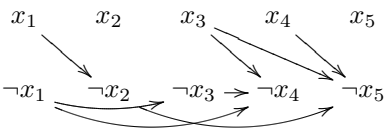
Démonstration. Pour chaque CFC de F', il suffit de remplacer les occurrences des éléments de la CFC par le plus petit élément, ce que

3 Les arcs d'implication $x_{C(i,j)} \to \neg x_{C(i,k)}$ de F'_2 partant chacun d'un littéral positif et aboutissant à un littéral négatif ne peuvent pas participer à un circuit.

Partons d'une formule mixte $F' = F_1 \wedge F_2$ où :

$$F_1 = (x_4 \rightarrow x_3) \wedge (x_4 \rightarrow x_1) \wedge (x_3 \rightarrow x_1) \wedge (x_4 \rightarrow x_3)$$
$$F_2 = \text{Auplus1}(x_1, x_2) \wedge \text{Auplus1}(x_3, x_4, x_5)$$

La représentation de la formule normalisée équivalente $F'' = F_1 \wedge F'_2$ avec les implications de la forme $l \rightarrow \neg x_j$ avec l de la forme x_i ou bien $\neg x_i$, et $i < j$, est la formule suivante :



Les implications du type $\neg a \rightarrow \neg b$ représentent bien F_1 et celles du type $a \rightarrow \neg b$ constituent F'_2.

● **Figure 2.4 :** Exemple de normalisation.

l'on peut faire en temps total linéaire.[4] Suite à cela, on affecte à $\bot$ les variables qui apparaissent plusieurs fois dans une clause Auplus1. Il ne reste plus qu'à propager ces affectations à $\bot$ en suivant les règles suivantes :

- Pour chaque clause $x_i \rightarrow \bot$, on peut remplacer x_i par $\bot$.
- Toute clause de la forme $\bot \rightarrow \_$ est à supprimer.
- Les $\bot$ apparaissant dans des clauses Auplus1 peuvent être supprimés.

La correction ne pose pas de problème. La propagation est destructive donc linéaire. ◀

Point essentiel : on suppose désormais que la formule F' est acyclique, et de la forme :

$$F' = \underbrace{\bigwedge_{1 \leq i \leq p} \neg x_{h(i)} \rightarrow \neg x_{g(i)}}_{F_1} \wedge \underbrace{\bigwedge_{1 \leq i \leq l} \text{Auplus1}(x_{C(i,j)})_{1 \leq j \leq n(i)}}_{F_2}$$

avec les variables renumérotées dans un ordre topologique du graphe d'implication acyclique de F', autrement dit de F_1, c'est-à-dire tel que $h(i) \leq g(i)$ pour tout i, $1 \leq i \leq p$. Un exemple est présenté figure 2.4.

2.3 Énumération

On énumère les modèles de F' en utilisant — directement, ou en adaptant l'algorithme de Feder — l'énumération des formules bijonctives exposée à la section précédente.

La réduction linéaire présentée ci-dessus pour trouver un Horn-renommage introduit beaucoup trop de variables quantifiées existentiellement : il devient difficile d'assurer un délai décent. On procède donc à l'énumération des solutions *via* une réduction plus naïve. Pour pallier les défauts de celle-ci, on propose deux méthodes distinctes :

- une utilisation paresseuse de la réduction $F' \mapsto F''$ présentée ci–dessus lors de l'énumération, qui permet au précalcul de rester

4 On peut toujours supposer que chaque variable donne accès à la liste de ses occurrences.

linéaire : on manipule directement la formule mixte $F' = F_1 \wedge F_2$, de taille linéaire, au lieu de manipuler $F'' = F_1 \wedge F_2$, de taille quadratique,
- un traitement plus lourd lors du précalcul, qui permet d'améliorer le délai.

On proposera finalement une combinaison des deux méthodes qui allie leurs avantages respectifs.

Avant cela, on montre à travers un algorithme « naïf » l'intérêt de la forme de F' obtenue au point précédent. Là encore, on confondra chaque variable x_i avec son indice i.

2.3.1 Énumération « naïve »

Dans un premier temps, on présente un algorithme « naïf », la procédure ÉnumNaïf de l'algorithme 2.7, qui exploite déjà la canonicité de F' obtenue au point précédent.

▶ ***Lemme 5*** L'algorithme 2.7 énumère les modèles de F' à délai linéaire $\mathcal{O}(|F'|)$.

Démonstration. Pour avoir le délai linéaire, il suffit de bien voir que lors d'une descente dans l'arbre de recherche, les opérations sont cumulativement linéaires en la taille de la formule, et que la recherche ne connaît pas d'échec. ◀

Il est à noter que cet algorithme énumère les solutions dans leur ordre lexicographique inverse. Ce n'est pas anodin : on réutilisera cette propriété pour concevoir l'algorithme final (présenté également dans l'algorithme 2.7).

2.3.2 Énumération par extension

▶ **Théorème 5 (énumération des Horn-renommages (adaptation))** On peut énumérer les Horn-renommages d'une formule F à délai linaire $\mathcal{O}(|F|)$.

Démonstration. On cherche directement à intégrer le filtrage lié aux clauses Auplus1 à l'algorithme 2.5 (pour 2Sat). Pour ce, il suffit de remplacer la séquence :

pour $(i \to \neg j) \in F$ **:**
  $a[j] \leftarrow a[j] + 1$

par sa généralisation naturelle :

pour Auplus1$(S) \in F \,|\, i \in S$ **:**
  pour $j \in S \,|\, j > i$ **:**
    $a[j] \leftarrow a[j] + 1$

et faire de même pour l'annulation du filtrage. On appelle algorithme 2.5' l'algorithme 2.5 ainsi modifié. Il n'y a pas de difficulté particulière, si ce n'est qu'au lieu de coûter $\mathcal{O}(n)$, le filtrage coûte désormais $\mathcal{O}(|F|)$ où $|F|$ est la taille de la formule. ◀

Un indice simple de la « supériorité » de l'algorithme 2.5 sur l'algorithme précédent (la procédure ÉnumNaïf de 2.7) est que, si on lui soumet une formule bijonctive, on procédera à l'énumération de ses Horn-renommages (qui sont *exactement* ses modèles) à délai $\mathcal{O}(n)$ où n est le nombre de variables : il généralise l'algorithme de Feder, ce que ne fait pas l'algorithme naïf.

2.3.3 Énumération par réduction

On réduit chacune des clauses Auplus1 à son équivalent sous forme d'une conjonction de clauses bijonctives, de taille nécessairement quadratique.

Désormais, on appelle k la longueur maximum d'une clause de la formule F originelle ; par conséquent, k majore aussi la longueur maximum d'une clause Auplus1.

▶ ***Lemme 6*** On peut, en $k-1$ étapes, coûtant chacune un temps $\mathcal{O}(|F|)$, et en espace total $\mathcal{O}(|F|+n^2)$, construire une formule bijonctive F'' équivalente à F' et qui porte sur exactement les mêmes variables.

Démonstration. On réduit la formule mixte $F' = F_1 \wedge F_2$ à sa forme bijonctive $F'' = F_1 \wedge F_2'$. Autrement dit, F_1 ne change pas.

L'équivalence suivante est évidente :

$$\operatorname*{Auplus1}_{1\leq j\leq n(i)}\left(x_{C(i,j)}\right) \;\Leftrightarrow\; \operatorname*{Auplus1}_{1\leq j<n(i)}\left(x_{C(i,j)}\right) \wedge \bigwedge_{1\leq j<n(i)}\left(x_{C(i,j)} \to \neg x_{n(i)}\right)$$

Il suffit de l'appliquer $k-1$ fois, tout en veillant à retirer les doublons parmi les clauses, pour obtenir la formule F''. On a construit F'', mais *sans redondances*. La taille de F'' étant bornée par n^2, on utilise l'espace annoncé. ◀

Ce lemme prouve que l'on peut énumérer les Horn-renommages d'une formule F à n variables à délai $\mathcal{O}(n)$ après un précalcul en temps $\mathcal{O}(k|F|)$ et en espace $\mathcal{O}(|F|+n^2)$, mais on préfère l'exploiter directement au point suivant pour prouver un résultat meilleur.

2.3.4 Combinaison des différents algorithmes

▶ **Théorème 6 (énumération des Horn-renommages (synthèse))**
L'algorithme 2.7 énumère les $k-1$ premiers modèles de F' à délai $\mathcal{O}(|F'|)$, puis les suivants à délai $\mathcal{O}(n)$, où k est la longueur maximale d'une clause, et n le nombre de variables.

Démonstration. L'algorithme 2.7 combine l'algorithme naïf et l'algorithme ÉnumLex[5] (2.6) de la section précédente en utilisant le lemme 6.

Cet algorithme est construit comme suit. On utilise l'algorithme naïf pour générer les $k-1$ premières solutions à délai $\mathcal{O}(|F|)$. À l'occasion de cette énumération, on intercale entre la production de deux de ces solutions successives une partie de la construction de la formule F'' (voir lemme 6). Ainsi, on peut dissimuler le temps (non linéaire) nécessaire à la construction de F'' dans le délai entre les $k-1$ premières solutions.

Une fois ces $k-1$ solutions produites, on peut lancer l'énumération de l'algorithme de Feder ou sa variante ÉnumLex appliquée à la formule F''. Pour vérifier que le précalcul de ces algorithmes reste linéaire, il faut noter qu'il ne dépend que de la partie F_1 de F'', qui a bien une taille linéaire. La production de solutions à délai $\mathcal{O}(n)$ commence alors.

Le problème, dans tout cela, c'est que, parmi les solutions produites, on va répéter chacune des $k-1$ premières solutions. Pour y remédier, on

5 On aurait pu utiliser l'algorithme Feder généralisé au lemme 5 en combinaison avec l'algorithme de Feder (algorithme 2.5) à la formule F'' (lemme 6) pour avoir le même résultat.

Il suffirait, par exemple, d'utiliser le premier pour générer une solution sur deux des $2k-2$ premières solutions à délai linéaire, puis d'utiliser le second pour générer le reste à délai $\mathcal{O}(n)$.

```
Affecte(F,l) :
    pour C ∈ F :
        si l ∈ C : Retirer C de F
        si l̄ ∈ C : Retirer l̄ de C
    si l de la forme x_i :
        pour Auplus1(S) ∈ F | x_i ∈ S :
            F ← F ∖ {Auplus1(S)} ∪ {{¬x_j} | x_j ∈ S ∖ {x_i}}
    ou alors si l de la forme ¬x_i :
        pour Auplus1(S) ∈ F | x_i ∈ S :
            F ← F ∖ {Auplus1(S)} ∪ {Auplus1(S ∖ {x_i})}
    renvoyer F
ÉnumNaïf(F,i) :
    si i = n + 1 :
        produire ∅
    sinon
        si {¬x_i} ∉ F :
            pour m ∈ ÉnumNaïf(Affecte(F,x_i),i + 1) :
                produire m ⊎ {x_i}
            pour m ∈ ÉnumNaïf(Affecte(F,¬x_i),i + 1) :
                produire m ⊎ {¬x_i}
ÉnumHornRen(F(x_1,...,x_n)) :
    Construire F′ la formule mixte correspondante
    i ← 1
    F″ ← {(¬x_i → ¬x_j) ∈ F′}
    pour m ∈ ÉnumNaïf(F,1) :
        pour Auplus1(S) ∈ F′ :
            pour j ∈ {i + 1,...,Card (S)} :
                F″ ← F″ ∪ {(¬S[i]∨¬S[j])}
        produire m
        si i = k − 1 :
            m_0 ← m
            interrompre
        sinon
            i ← i + 1
    pour m′ ∈ ÉnumLex(F″) :
        si m′ = m_0 :
            interrompre
        produire m′
```

● **Algorithme 2.7 :** Énumération des Horn-renommages par synthèse de l'algorithme naïf et de la réduction à l'énumération des modèles d'une formule bijonctive dans l'ordre lexicographique.

s'est arrangé pour que l'algorithme naïf énumère dans l'ordre lexicographique inverse ; et pour proposer un post-traitement à l'algorithme de Feder, ÉnumLex, qui permet d'énumérer dans l'ordre lexicographique. Nos deux algorithmes vont donc énumérer dans des ordres inverses.

Le premier algorithme sert à produire les $k-1$ dernières solutions dans l'ordre lexicographique, et le second toutes les autres, en partant de celle minimale pour l'ordre lexicographique, puis en les énumérant toutes dans cet ordre. Pour éviter les répétitions, il suffira donc d'interrompre la seconde énumération au moment où elle produira la $(k-1)$-ième solution avant la fin, que l'on a stockée dans la variable m_0. ◀

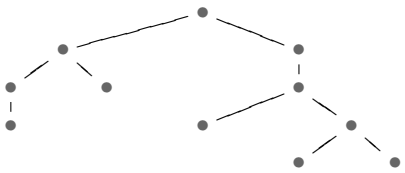
Conclusion

On a complètement explicité l'algorithme de Feder pour l'énumération des modèles d'une formule bijonctive, et proposé un post-traitement qui rend lexicographique l'ordre d'énumération, sans altérer le délai.

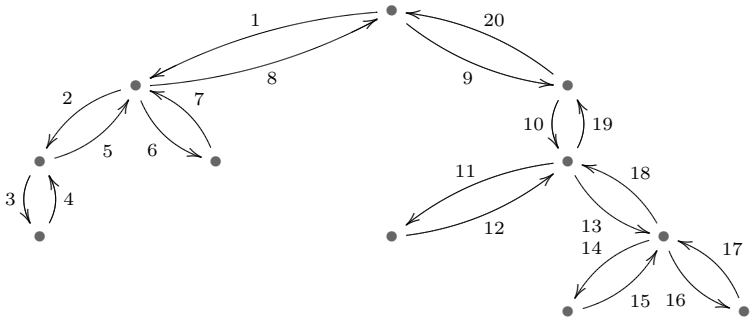
On a généralisé cet algorithme en proposant un algorithme d'énumération des Horn-renommages qui énumère les $k-1$ premiers Horn-renommages d'une formule F à délai $\mathcal{O}(|F|)$, où k est la longueur maximale d'une clause de F, puis les suivants à délai $\mathcal{O}(n)$. Cet algorithme généralise aussi la recherche d'un Horn-renommage en temps linéaire [Héb94] et le test d'unicité d'un Horn-renommage lui aussi en temps linéaire [Héb95].

Annexe : énumération des nœuds d'un arbre à délai constant

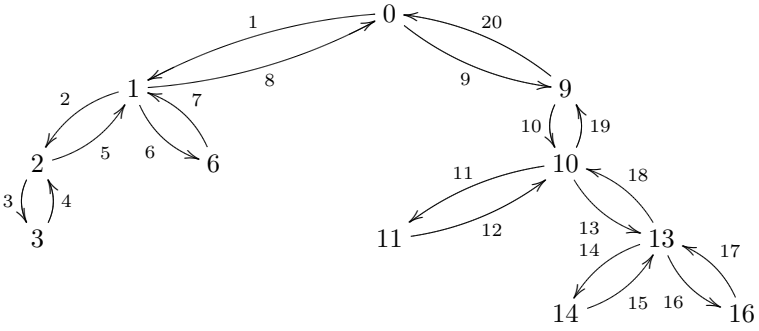
Comment parcourir cet arbre ?



On peut essayer de le parcourir en profondeur, par exemple.

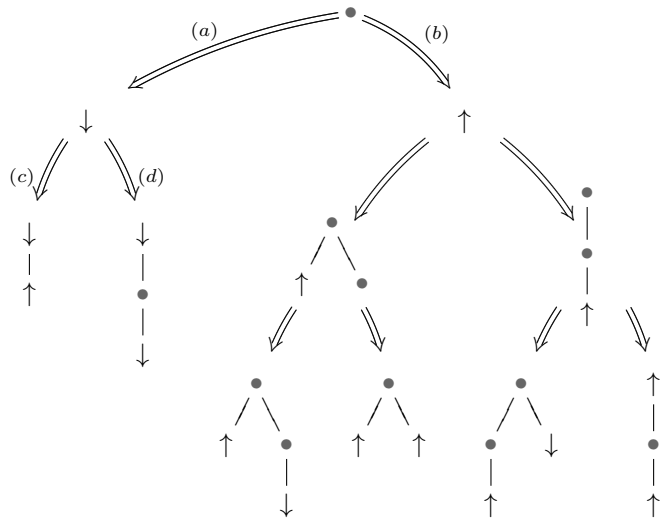


Essayons en préfixe : au bout de combien d'étapes obtient-on chaque nœud ?



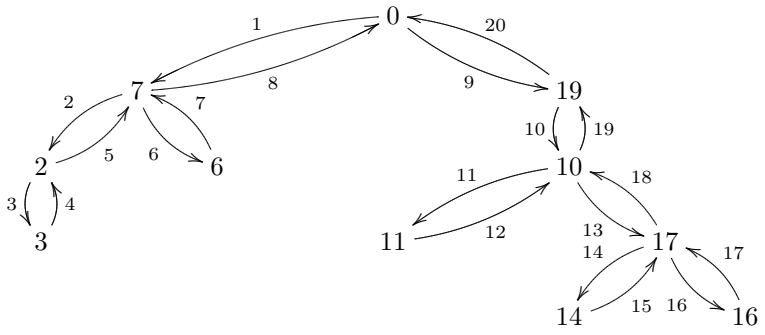
Ça ne fonctionne pas. En postfixe, on a évidemment le problème inverse. Le problème, c'est l'existence de séquences de remontées dans le cas préfixe, et de redescente dans le cas postfixe qui ne sont pas bornées (exemple simple : l'arbre unaire...).

L'astuce, justifiée par la figure 2.5, consiste à produire les nœuds de profondeur impaire lors de la descente, et ceux de profondeur paire lors de la remontée :



On étudie les différents cas possibles : on part du principe qu'un nœud vient d'être produit, et on se demande quel sera le suivant.
On peut lire l'arbre de cas ainsi : un nœud qui vient d'être produit l'a été soit en descendant (a) soit en remontant (b). Dans le premier cas, soit le fils du nœud n'a pas de fils (c) auquel cas il est produit, soit il a un fils qui est produit (d). La deuxième moitié de l'arbre se lit de la même manière.
Il manque bien entendu les cas suivants, encore plus simples : le nœud a été produit en remontant mais n'a pas de père ; le nœud a été produit en descendant mais n'a pas de fils. Le premier cas est trivial : la racine vient d'être produite en remontant : l'énumération est arrivée à son terme (si la racine est considérée de profondeur 1), ou bien on a une contradiction (si la racine est considérée de profondeur 0). Le second est simple : Soit le père du nœud a un fils, qui est produit, soit pas. Dans le second cas, soit le père est la racine, auquel cas l'énumération est soit terminée, soit il y a contradiction (comme précédemment). Sinon, le grand-père est produit.

● **Figure 2.5 :** Étude de cas exhaustive pour prouver le délai constant de l'énumération alternée.



Chapter 3

Models of Horn Formulas are Enumerable at Linear Delay

Abstract The unique satisfiability problem of Horn formulas was proved to be quadratic by Minoux (1992) [Min92]. Using ideas based on those of [Min92], Berman, Franco, and Schlipf (1995) [BFS95] proved it to be nearly linear. We simplify the presentation of their algorithm and adapt it slightly in order to perform enumeration of the solutions at a delay that *is* their unique solution decision time, i.e. $\mathcal{O}(\alpha(|F|)|F|)$ where $|F|$ is the formula size and α is the inverse Ackermann function, or (at choice) $\mathcal{O}(n \log n + |F|)$ where n is the number of variables of F.

Contents

1 A First Approach 40
1.1 Algorithm Overview 40
1.2 First Complexity Considerations 42
2 Incremental Circuit Building 42
3 The Final Algorithm 43
3.1 Using an Union-Find Algorithm 43
3.2 The Final Algorithm 44

Introduction

Counting the models of a propositional Horn formula is a #P-Complete problem [Val79], therefore not tractable. Nevertheless, the unique satisfiability problem, which can be considered as a (very) limited way of counting, i.e. it answers one of the three "numbers" 0, 1, or atleast 2, was proved to be nearly linear by [BFS95]. We can generalize this "bounded counting" to the problem of counting *up to* k, for any fixed k, that returns a value in the set $\{0, 1, ..., k-1, \text{atleast } k\}$.

This paper provides an algorithm that extends the unique solution decision algorithm of [BFS95], and performs exhaustive model enumeration at nearly linear delay. A straightforward corollary of our result is that counting up to k can be done in time $\mathcal{O}(k\alpha(|F|)|F|)$, with production of the corresponding solutions in the same time.

Organisation of this Paper We present the algorithm in three steps. First of all, we give an overview of the algorithm (algorithm 3.1), with the logical argument (called *Trick1*) proving its correctness, and explaining

the main idea. Nevertheless, this algorithm, if implemented poorly, has a $\mathcal{O}(n|F|)$ delay, which is the complexity of the naive algorithm. In sections 2 and 3, we give a more explicit writing of the algorithm on its critical part, with more implementation details, that allow to reach the expected complexity. Each section has its main argument, denoted respectively by *Trick1*, *Trick2*, and *Trick3*, that are *arguments on logic* that are the key points leading to properties of correctness and/or complexity. Note that algorithms 3.2 and 3.3 below only explicitly describe the critical part of the algorithm and are justified resp. by *Trick2* and *Trick3*.

Preliminaries

We assume the reader is familiar with propositional logic. In the whole chapter, F will be a Horn CNF formula on n variables. Since F is assumed in CNF, we will see it as a set of clauses, which are seen as sets of literals, we therefore will use set notations accordingly, e.g., $\emptyset \in F$ means "F contains an empty clause." We define $F(l) = \{C \in F \mid l \in C\}$ and $F[l] = \{C \setminus \{\bar{l}\} \mid C \in F \setminus F(l)\}$. The formula $F[l]$ is obtained by setting l to 1 (true[1]) in the formula F and simplifying it accordingly. We assume no formula contains a tautological clause.

We use the RAM model (see [Gra96]). In this model, an input of size N is a sequence of N integers identified to their binary representation, each one in the interval $[0, N[$. This sequence is contained in N registers of size $\lfloor \log(N-1) \rfloor + 1$ (the size of the binary representation of $N-1$) or $\Theta(\log N)$.

As suggested by [BFS95], we can see the formula as a *bipartite graph* where one of the vertex sets is the set of literals, and the other is the set of clauses; in this graph there is an edge between a literal and a clause if, and only if the literal belongs to the clause. With this model, as in [BFS95], the size $N = |F|$ of a formula is the total number of variable occurrences.

1 A First Approach

1.1 Algorithm Overview

This section starts by stating the adopted search strategy, and shows how it leads to an algorithm, which is presented in Algorithm 3.1. The main idea is the following: we want to perform a recursive search, but with no backtracking due to failure, i.e. the recursive enumeration procedure should always produce at least one solution before backtracking.

That is why, as a precomputation step, we perform *positive* unit propagation. This way, if the formula is satisfiable, then it is zero-valid (i.e. $(0,...,0)$ is a model); the algorithm terminates at this point otherwise. In algorithm 3.1, this is done in the procedure EnumHorn, lines 2–5. In the whole algorithm, models are considered as sets of literals, where every variable appears once. Lines 7-8 can therefore be read as "for every model of the zero-valid formula obtained after positive unit propagation, complete it with the positive literals that were propagated." Zero-validity is the key property that will be maintained

1 In this chapter, we write 0 instead of $\bot$ and 1 instead of $\top$.

```
EnumHorn(F, 𝒱):
    S ← ∅
    while ∃x ∈ 𝒱 {x}∈F :
        F ← F[x]
        S ← S ⊎ {x}
    if ∅ ∉ F :
        for M ∈ EnH0(F, 𝒱 ∖ S) :
            yield M ⊎ S
```

```
EnH0(F, 𝒱) : :
    if 𝒱 = ∅ :
        yield ∅
    elsif ∃x ∈ 𝒱 ∀y ∈ 𝒱 {¬x, y} ∉ F :
        for M ∈ EnH0(F[¬x], 𝒱 ∖ {x}) :
            yield M ⊎ {¬x}
        if {¬x} ∉ F :
            for M ∈ EnH0(F[x], 𝒱 ∖ {x}) :
                yield M ⊎ {x}
    else
        Let S = {x1,...,xk} a circuit of 𝒢(F)
        foreach xi ∈ S do
            Replace xi by x1 in F
            Replace ¬xi by ¬x1 in F
        S ← S ∖ {x1}
        for M ∈ EnH0 (F, 𝒱 ∖ S) :
            if x1 ∈ M :
                yield M ⊎ S
            else
                yield M ⊎ {¬x | x ∈ S}
        Undo the replacements in F
```

● **Algorithm 3.1:** Basic algorithm.

as an invariant, and that guarantees satisfiability is maintained, which is *Trick1*.

We will perform enumeration of the models of this zero-valid formula with the recursive procedure EnH_0 in algorithm 3.1. We need to recursively get rid of variables *in a way that maintains zero-validity*. Since the considered Horn formula is guaranteed to be zero-valid, two different situations occur. In both situations, checking zero-validity is maintained for each recursive call is easy; correctness follows.

The first situation is the case where affecting a given variable x to 1 does not make a positive unit clause appear, i.e. there is no y such that $\{\neg x, y\} \in F$. This case splits into two sub-cases. If $\{\neg x\} \in F$, then the case is easy to deal with. In the other case, x can be set indifferently to 0 or to 1 without affecting the zero-validity property of F, we have therefore to make one recursive call on $F[\neg x]$ and one on $F[x]$.

The second situation is the case where, for every variable x, we can find y such that $\{\neg x, y\} \in F$. As a consequence, the *implication digraph* $\mathcal{G}(F)$ of the formula F, essentially introduced by [Min92] and defined as $\mathcal{G}(F) = \{x \to y \,|\, \{\neg x, y\} \in F\}$, has a circuit (of length at least two): we can therefore replace, in the formula, any variable appearing in the circuit, by a given variable of the circuit; in every model, they have the same value.

```
1  EnH0(F, V, visitList):
2      ...
10     else
11         if visitList = ∅ :
12             Take x ∈ V
13         else
14             x ← Peak(visitList)
15             Take x ∈ {y ∈ V | {¬x, y} ∈ F}
16         while x ∉ visitList :
17             Push(visitList, x)
18             Take x ∈ {y ∈ V|{¬x, y} ∈ F}
19         S ← ∅
20         while Peak(visitList) ≠ x :
21             y ← Pop(visitList)
22             S ← S ⊎ {y}
23             Replace y by x in F
24         for M ∈EnH0 (F, V ∖ S, visitList) :  ...
29         Undo the changes of F
```

● **Algorithm 3.2:** Incremental circuit building, where lines 11–15 of algorithm 3.1 have been substituted by lines 11–23 (with obvious changes), and lines 25–29 are just the same as the lines 17–21 in algorithm 3.1, except the substitution of x_1 by x.

1.2 First Complexity Considerations

Consider algorithm 3.1. The precomputation is linear, we do not need to actually consider it. What is the cost of the instruction "Undo" on line 21? We can imagine that, every time a value is changed in memory, a backup save is made and pushed on some stack without affecting the complexity up to a constant factor; therefore undoing the changes costs at most as much as performing them, consequently we do not need to take it into account. This is the case of line 21, but not only: in particular, in the recursive calls of lines 5 and 8, $F[l]$ (where l is either $\neg x$ or x) is computed in time proportional to the "difference" between F and $F[l]$, i.e. the part of F that is removed, provided $F[l]$ is computed in-place. We therefore compute it in-place, but need to restore F after the recursive call. We can do it in the same time for the reason mentioned in previous paragraph. As a consequence, the first part (lines 2–9) is responsible for a linear delay $\mathcal{O}(|F|)$, i.e. the delay is linear if we exclude the time cost of everything else, that is to say the time needed for computing a circuit, and the variables replacements (lines 11-14).

2 Incremental Circuit Building

We are now concerned with building a circuit of $\mathcal{G}(F)$ (circuit, for short) *incrementally*, i.e. continuing each time what was begun before. In order to do so, we need to detail. To find a circuit, we only need to take any variable, and then to follow a path until we find a variable that was already considered. Then this variable and all the variables that were considered after this first variable form a circuit. Building this circuit is done in algorithm 3.2, with the use of instructions on stacks:

Push(stk,val) adds a value on the top of the stack and returns the modified stack,
Pop(stk) removes the value on the top of the stack and returns this value, and
Peak(stk) returns the values on the top of the stack.

Trick2 is the following invariant of this algorithm: the set of visited variables and edges forms a path of $\mathcal{G}(F)$, maintained as a stack called visitList (assumed initialised to the empty list) in the algorithm. This is the case since after a circuit contraction, the whole circuit becomes a single variable, which is on the top of the stack. It is not hard to see that the first part of the algorithm (corresponding to lines 2–9 of algorithm 3.1) can only remove a variable that is on the top of the stack or unvisited, therefore maintaining the key property that the stack is a set of variables forming a path.

If we except the "Replace" instruction (line 23 of algorithm 3.2), this part is responsible for a linear delay. From now the goal is finding a way to avoid *actually* replacing the variables.

3 The Final Algorithm

3.1 Using an Union-Find Algorithm

We are concerned with having replacements made more easily. To do so, instead of replacing actually variables in the formula, we define equivalence classes, and search for equivalence class while considering a literal in a clause. This allows to make explicit use of the classical union-find algorithm.

Take a data structure representing a partition. The union-find procedures are the following operations:

Find(x) determines which set, given by its representative, x is in;
Union(x,y) merges the two sets represented respectively by x and y into a single set, returns the representative of this new set; and
MakeSet(x) makes a new set $\{x\}$, represented by x.

We are concerned by the time cost of n calls to Union and m calls to Find, where $m \geq n$. By a result of [Tar75], this cost, depending on the union-find variant used, takes either $\mathcal{O}(n \log n + m)$ or $\mathcal{O}(\alpha(m)m)$ (see also [BFS95] for details).

In the algorithm, the sets will be sets of variables, meaning two variables belong to the same set when they are equal in all models. Sets are initially the singletons corresponding to variables; when a circuit is collapsed, all sets corresponding to variables occurring in the circuit are merged with calls to Union. Between two consecutive solutions, obviously, there cannot be more than $n - 1$ calls to Union.

When considering some elements of a clause, such as "the positive literal hold by a clause", or such as "the first variable visiting a clause" (see below), instead of actually considering the variable, that may have been previously merged with another, we consider the representative of this variable x thanks to a call to Find(x). This way, when a circuit is collapsed, we don't need to actually replace the occurrences of variables of the circuit by one given representative x, we will just have to look at literals instances *through the lens* of Find.

Since every instance will be considered a constant number of times

```
1  ...
16 while x ∉ visitList :
17     Push(visitList, x)
18     for C ∈ longCl(x) :
19         if firstVisit(C) ≠ ⊥ :
20             len(C) ← len(C) − 1
21             longCl(x) ← longCl(x) ∖ {C}
22             if len(C) = 1 :
23                 y ← Find(firstVisit(C))
24                 longCl(y) ← longCl(y) ∖ {C}
25                 shortCl(y) ← shortCl(y) ⊎ {C}
26         else  firstVisit(C) ← x
27     Take x ∈ {Find(y) ∈ V | {¬x, y} ∈ shortCl(x)}
28 last ← x; S ← ∅
29 while Peak(visitList) ≠ last :
30     y ← Pop(visitList)
31     S ← S ⊎ {y}
32     z ← Union(x, y)
33     longCl(z) ← longCl(x) ⊎ longCl(y)
34     shortCl(z) ← shortCl(x) ⊎ shortCl(y)
35     x ← z
36 ...
```

● **Algorithm 3.3:** Final algorithm, where lines 16–23 of algorithm 3.2 have been substituted by lines 16–35.

between two consecutive solutions, Find will be called $|F|$ times in the same time. We will see the delay is therefore either $\mathcal{O}(n \log n + |F|)$ or $\mathcal{O}(\alpha(|F|)|F|)$ since clearly $|F| \geq n$.

3.2 The Final Algorithm

We present here the implementation details needed for getting the presented complexity; we have kept almost the same notations and arguments as those of [BFS95]. The main point of this part is a logical argument of importance (*Trick3*, explained below), based on Davis-Putnam resolution and on clause subsumption, that allows to guarantee two given sets of clauses are disjoint, which permits to compute their union in constant time (line 33).

Clauses are considered differently according to the number of negative literals they contain, called their *length* : $\text{len}(C) = \text{Card}\,(\{x \in \mathcal{V} \mid \neg x \in C\})$. When their length is 1, they are *short clauses* else they are *long clauses*. Since the formula can be seen as a bipartite graph, and contains no positive unit clause, clauses are accessed through the *negative* literals they hold. We define $\text{shortCl}(x)$ as the set of short clauses holding $\neg x$ and $\text{longCl}(x)$ as the set of long clauses holding $\neg x$. To each clause C we associate a variable $\text{firstVisit}(C)$ that is the first *visited* variable x such that C holds $\neg x$, and is initialized to $\bot$. Notice both sets $\text{shortCl}(x)$ and $\text{longCl}(x)$ can be implemented as circular doubly-linked lists of pointers on clauses; this way removing an element from such a list or concatenating two disjoint lists — reflecting the union of the disjoint sets they represent — can be done in *constant* time.

Algorithm 3.3 describes how to use and maintain this representation on the critical part of the algorithm; it is easy to see how to adapt the rest of the algorithm with this notation. We assume $\text{MakeSet}(x)$

was already called for every $x \in \mathcal{V}$ during the initialisation, i.e. in the function EnumHorn (see algorithm 3.1).

Now we want to show how we can manage to proceed to union (line 33) of the clauses holding some $\neg x$ and the clauses holding some $\neg y$ in constant time when collapsing a circuit. To do so, we need the respective sets of long clauses associated to the different variables of the circuit to be pairwise disjoint (if we consider that clauses differing only by their address are different). *Trick3* consists, for every considered variable x, in considering every clause holding $\neg x$, and, if not already visited, in marking it as "first visited by x". If it was already visited, then it was by some y, previously visited, that means with a path from x to y in $\mathcal{G}(F)$ and therefore such that the implication $y \to x$ holds by transitivity. The clause C is in the form $\neg x \vee \neg y \vee C'$. We can perform resolution of the "clauses" C and $\neg y \vee x$, and deduce $\neg y \vee C'$, which subsumes C. We can therefore replace C by $\neg y \vee C'$ without affecting the set of models of the formula. This last clause does not hold $\neg x$, and has one negative literal less than C. This justifies lines 20–21 of algorithm 3.3. There only remains to manage the case this clause has become a short clause, which is done in lines 22–25. This way we can make sure the respective sets of long clauses associated to the different variables of the circuit are pairwise disjoint.

Now we give some details that clarify a point the reader might find not so obvious. First of all, short clauses containing a positive literal, i.e. in the form $\{\neg x, y\}$, and short clauses not containing a positive literal, i.e. in the form $\{\neg x\}$, should be stored in two separate lists, say $\text{shortCl}_+(x)$ and $\text{shortCl}_0(x)$. This is easy to maintain. Now we want to prove testing whether there is some x such that no y is such that $\{\neg x, y\} \in F$ can be done in constant time. To do so, we need to maintain the following invariant: the first clause in $\text{shortCl}_+(x)$ (if this set is not empty) is not tautological, i.e. not in the form $\{\neg x, y\}$, with $\text{Find}(x) = \text{Find}(y)$. The critical point is the following: when a circuit is collapsed into a single variable x, $\text{shortCl}_+(x)$ may begin by a tautological clause. To know it, we just have to test whether $\text{Find}(y) = x$ where y is the positive literal hold by the first clause in $\text{shortCl}_+(x)$. If so, remove this clause and repeat the operation until the property is restored. This will cost at most $|F|$ calls to Find between the production of two consecutive solutions.

Recall that the number of calls to Union is at most $n - 1$ where n is the number of variables. This justifies that the enumeration delay complexity of this algorithm is as it is claimed.

Conclusion

We considered that giving the full algorithm would not be a good way to explain it; instead, we gave the main arguments, that make the announced result credible. We let the reader complete the less critical parts of the algorithm, in order to convince himself it has the announced complexity; we made sure no new difficulty would arise.

Acknowledgment The author would like to thank professor Étienne Grandjean for multiple, in-depth readings of this paper, providing valuable feedback on its clarity.

Remarques et synthèse

Remarques sur l'énumération des formules bijonctives

Dans [APT79] il est montré que la décision des formules bijonctives quantifiées se fait en temps linéaire. Le témoin de la validité d'une telle formule est une stratégie gagnante. On peut se demander si l'on peut énumérer de tels témoins : la question n'est pas évidente dans la mesure où il faudrait déjà fixer en quelque sorte la forme attendue pour une stratégie gagnante.

En fait, dans le cas d'une formule bijonctive quantifiée F, les stratégies gagnantes admettent une forme canonique simple, que l'on peut effectivement énumérer à délai $\mathcal{O}(n)$ après précalcul $\mathcal{O}(|F|)$, en se ramenant facilement à l'énumération des formules bijonctives.

Remarques sur l'énumération des modèles de Horn

L'article de Pretolani [Pre93] indique que le problème d'unicité pour Horn se fait en temps linéaire. Le cœur de l'argument tient essentiellement à ce que le problème *union-find* se traite en temps linéaire quand les classes d'équivalence sont des intervalles, ce qui est aussi le cas dans notre solution, argument reposant sur [GI91]. On a toujours un doute sur la validité de cet argument !

- Si l'argument n'est pas valide, le temps de décision des algorithmes respectifs de [Pre93] et de [BFS95] est $\mathcal{O}(\alpha(|F|)|F|)$ où α est la fonction de Ackermann inverse ; leurs algorithmes sont alors des cas particuliers du nôtre.
- Si l'argument est valide, notre algorithme en l'état énumère en fait (et à notre insu !) à délai linéaire $\mathcal{O}(|F|)$, et est donc optimal, par conséquent tout algorithme de test d'unicité en temps linéaire est encore un cas particulier du nôtre.

Dans les deux cas, notre prétention à généraliser le meilleur algorithme d'unicité connu est fondée. Par souci d'optimalité, cependant, il serait souhaitable que l'argument de Pretolani soit effectivement valide. Nous n'avons malheureusement pas dans l'immédiat le temps de le vérifier.

Remarques sur l'ordre d'énumération

La question de l'ordre d'énumération a son importance dans la mesure où elle reflète la structure de l'algorithme (l'ordre et la manière de construire des solutions partielles) et où la structure du problème peut être respectée à travers des contraintes d'ordre, qui permettent des énumérations efficaces. Les remarques 2 (page 11) et 3 (page 13) vont clairement dans ce sens.

Ainsi, la possibilité d'utiliser l'algorithme de Creignou et Hébrard [CH97] pour énumérer dans une grande diversité d'ordres différents se paye assez cher : la complexité souffre d'un facteur n supplémentaire. Si au lieu de plaquer sur une instance l'attente d'un ordre donné, on se laisse guider par un ordre « naturel » de cette instance, on peut alors énumérer avec un délai équivalent au temps de la décision non triviale.

On définit brièvement ce que l'on entend par *ordre lexicographique étendu*. On dit d'un ordre $<$ sur les n-uplets $(x_1,...,x_n) \in \{\bot,\top\}^n$ qu'il est *lexicographique étendu* s'il existe $i \in \{1,...,n\}$ et un signe $s \in \{\neg,\emptyset\}$ tels que :

- pour toute paire de n-uplets $x = (x_1,...,x_n)$ et $y = (y_1,...,y_n)$, si $s(x_i) > s(y_i)$ alors $x > y$;
- l'ordre $<$ restreint à l'ensemble des n-uplets de la forme

  $$(x_1,...,x_{i-1},\bot,x_{i+1},...,x_n)$$

  est un ordre lexicographique étendu, et
- de même avec $\top$.

Dans ce cas, on dit que i est le bit de poids fort, et que s est sa polarité. Quelles contraintes pèsent sur l'ordre d'énumération ?

- L'algorithme de Creignou et Hébrard énumère dans n'importe quel ordre lexicographique étendu.
- L'algorithme que l'on fournit pour les formules affines pose uniquement des contraintes de polarité.
- L'algorithme de Feder avec notre post-traitement énumère dans un ordre lexicographique étendu complètement contraint.
- L'algorithme d'énumération de Horn pose uniquement des contraintes de bit de poids fort.

On laisse au lecteur le soin de constater que les ordres d'énumération de la seconde partie de ce mémoire sont tous des ordres lexicographiques classiques, qui imposent une pondération des variables qui est exactement un ordre d'élimination (au sens expliqué dans cette partie) reflétant la structure du problème.

Synthèse

Pour la classe générale des formules propositionnelles CNF et ses classes polynomiales pour l'énumération, ainsi que pour les problèmes de décision, d'unicité, et d'énumération des Horn-renommages, les meilleures complexités connues sont les suivantes.

	Décision	Unicité	Énumération
général	NP-complet [Coo71, Lev73]	CoNP-difficile [BG82]	NP-difficile CoNP-difficile
bijonctive	$\mathcal{O}(\|F\|)$ [EIS76, APT79]	$\mathcal{O}(\|F\|)$ [HJ85]	délai $\mathcal{O}(n)$ après $\mathcal{O}(\|F\|)$ [Fed94]
Horn-ren.	$\mathcal{O}(\|F\|)$ [Asp80, CCH$^+$90]	$\mathcal{O}(\|F\|)$ [Héb95]	délai $\mathcal{O}(\|F\|)$ puis $\mathcal{O}(n)$ [Chapitre 2]
Horn	$\mathcal{O}(\|F\|)$ [DG84, IM82]	$\mathcal{O}(\alpha(\|F\|)\|F\|)$ [BFS95]	délai $\mathcal{O}(\alpha\|F\|\|F\|)$ [Chapitre 3]
affine	Gauss	Gauss	délai $\mathcal{O}(n)$ après Gauss [Chapitre 1]

Dans ce tableau, $|F|$ et n désignent respectivement la taille de la formule et le nombre de variables.

Deuxième partie

Logique du premier ordre

Introduction à la seconde partie

À quoi reconnaît-on une question difficile ?

Par question, on signifie le problème de savoir si un énoncé logique est valide dans le monde, c'est–à–dire dans l'ensemble des faits. L'énoncé porte sur l'articulation logique des faits entre eux.

Un exemple de question peut être : est-il vrai qu'« il n'y a pas de fumée sans feu » ? Ici l'énoncé est l'assertion entre guillemets ; répondre à cette question consiste à considérer l'ensemble des faits et, à leur lumière, déterminer si, en effet, l'énoncé est bien vrai dans ce contexte.

Par difficulté d'une question, on entend l'expression du temps requis pour y répondre uniquement en fonction de l'étendue du monde considéré. On ignore ainsi la composante de discours pur, le temps passé à jouer logiquement avec l'expression de la question, et on se focalise plutôt sur le temps passé à considérer les différents faits qui composent le monde.

Ainsi, lorsqu'un énoncé est en fait un théorème, la question associée sera considérée comme facile, puisque la réponse sera « oui » indépendamment du monde considéré, et par conséquent, il n'est pas même nécessaire de regarder le monde pour y répondre. Évidemment, cette facilité élude le temps nécessaire à établir la nature tautologique de l'énoncé. On ne s'intéresse donc à la complexité d'une question que dans son rapport au monde, et non dans son rapport au discours.

À ce stade, il devient difficile de repousser encore l'introduction du formalisme. Les énoncés considérés seront exprimés dans la langue un peu limitée du fragment existentiel de la logique du premier ordre. Le terme existentiel signifie que le problème considéré est un problème de satisfaisabilité, c'est-à-dire d'existence d'un état de faits. Un énoncé aura ainsi la forme $\phi = \exists x_1 ... \exists x_n \quad \psi$ où ψ est une formule sans quantificateurs. Une question sera donc un objet $Q(\phi)$ qui signifie exactement : est-ce que ϕ est bien vraie dans le monde considéré ? C'est–à–dire que $Q(\phi)$ est une fonction qui à un monde $\mathcal{M}$ associe une réponse notée $Q(\phi)(\mathcal{M})$.

Nous décrirons la difficulté d'une question par une fonction qui exprime le temps requis pour y répondre en fonction de la taille du monde. Ainsi, si une question posée dans un monde $\mathcal{M}$ de taille $|\mathcal{M}|$ trouve sa réponse au pire au bout d'un temps $k|\mathcal{M}| \log |\mathcal{M}|$, on dira d'elle qu'elle est facile ; si au contraire le temps nécessaire n'est borné par aucune fonction de la forme $k_1|\mathcal{M}|(\log |\mathcal{M}|)^{k_2}$ alors on dira d'elle qu'elle est difficile.

Le résultat de cette seconde partie du mémoire est une condition nécessaire et suffisante que doit remplir un énoncé pour que la question associée soit facile. Cette condition ne porte en fait que sur la manière dont les éléments atomiques de la formule se partagent les variables ; les autres aspects de l'énoncé sont ainsi prouvés sans pertinence.

Organisation de cette partie

Le quatrième chapitre procède à une étude systématique *ex nihilo* de la classe des requêtes *conjonctives acycliques*, une classe bien connue (voir par ex. [CM77, Yan81, Fag83, Fag93, AHV95, GSS01, FFG02, Kol03, FG06]), et propose des énoncés et des preuves claires de faits informellement connus ou nouveaux. Il s'agit d'une ébauche d'article qui contient toutes les définitions fondamentales sur les requêtes.

Le cinquième chapitre est l'article [BB12a] paru à la conférence CSL 2012. Il étudie la classe duale de celle des requêtes conjonctives, les requêtes conjonctives négatives. Une requête conjonctive est de la forme :

$$\phi \;=\; \exists x_1 ... \exists x_n \;\bigwedge_i \alpha_i$$

Il s'agit du problème de satisfaisabilité d'une conjonction d'atomes positifs, qui est NP-complet [CM77, AHV95]. Son problème dual :

$$\mathrm{dual}(\phi) \;=\; \forall x_1 ... \forall x_n \;\bigvee_i \alpha_i$$

est le problème de validité d'une disjonction d'atomes, connu pour être CoNP-complet. Pour le décider, on peut décider sa négation :

$$\neg\mathrm{dual}(\phi) \;=\; \exists x_1 ... \exists x_n \;\bigwedge_i \neg\alpha_i$$

La conclusion de ce chapitre suggère fortement que la classe des requêtes de la troisième forme, que l'on appelle *conjonctives négatives*, a le pouvoir d'exprimer des requêtes conjonctives dites signées qui généralisent cette classe et la précédente. En quelque sorte, si les requêtes conjonctives constituent un cas particulier un peu trivial de requêtes conjonctives signées, en revanche la classe des requêtes conjonctives négatives est une forme *normalisée* sous laquelle toutes les requêtes conjonctives signées peuvent se mettre.

On peut aussi voir cette classe comme une généralisation des formules CNF. Alors que les dichotomies de Schaefer et de Creignou et Hébrard donnent un critère sur la structure qui permet de savoir si la difficulté du problème va être polynomiale ou NP-difficile, ce chapitre donne un critère sur la classe d'équivalence de la formule, donnée par le bidual de son hypergraphe (qui peut être obtenu en assimilant tous les sommets portés par un même ensemble d'arêtes à un seul d'entre eux), qui détermine si le temps de décision sera linéaire ou pas.

Le sixième chapitre unifie et généralise les deux chapitres précédents à travers un résultat simple et riche d'enseignements. Ce résultat met en jeu le rôle central d'un nouveau concept associé aux requêtes signées : l'hypergraphe bicolore et une notion d'acyclicité associée qui généralise simultanément l'alpha et la bêta-acyclicité. La difficulté tient essentiellement à la combinatoire dans la mesure où les outils algorithmiques, pour l'essentiel, étaient déjà présents dans le chapitre précédent.

Le septième chapitre fournit une preuve bien plus simple du résultat de combinatoire du chapitre précédent qui permet d'unifier les caractérisations des différentes notions d'acyclicité des hypergraphes, et de plus apporte la seule preuve de combinatoire qui manquait pour que l'ensemble de nos preuves soit pleinement autonome.

Chapter 4

About Conjunctive Queries

> The general form of proposition is the essence of proposition.
>
> Ludwig Wittgenstein – *Tractatus logico-philosophicus* §5.471

Abstract Acyclic conjunctive queries have known a much deserved interest (see, for example [Fag83] and [BFMY83]). Moreover, [BDG07, Bag09] proved (almost) the following dichotomy:

- Conjunctive Queries can be decided optimally fast if and only if they are acyclic.

Furthermore, they defined a combinatorial criterion such that the solutions of a conjunctive query can be enumerated optimally fast if and only if this criterion is satisfied. We come back on these results, and provide a simple, clean formalism allowing both concise and complete proof of these results, with the proof giving ready-to-use algorithms as a bonus.

We provide extensions of our result with lexicographically ordered enumeration, counting and quantified versions of this problem.

Contents

1 Optimal Enumeration Class and Reduction 55
1.1 Optimal Enumeration Class 55
1.2 A Reduction 58
2 Properties of Queries Classes 61
2.1 Queries Definition and Basic Properties 61
2.2 Conjunctive Queries 66
2.3 Additional Properties and Summary 70
3 Conjunctive Queries and Acyclicity 72
3.1 Quantifier-Free Conjunctive Queries and Acyclicity . 72
3.2 Existential Conjunctive Queries and Acyclicity-based Properties 79
3.3 The Whole Picture 84

Introduction

Motivations This chapter is about classifying queries with regards to their data complexity. This goal illustrates an attempt to provide an adequate formalism to reason about data complexity of queries.

Expressed with this formalism, previously informally known results find both *clear statements* and *simple proofs*. We also show that natural questions arising while considering such issues find a natural translation into this formalism, that are once again answered with simple proofs. This paper is at least as concerned with didactics as by stating new facts; the new results are just side-effects of our formalism: when things are well-formulated, some facts become obvious.

As an example, the $\mathcal{Q}(\phi)(\mathcal{S})$ notation we use is very similar to the often used $\phi(\mathcal{S})$ notation, except this is well-typed, allows fixed-query complexity considerations to be completely natural, and additionally allows to consider problems other than enumeration. Another aspect of this quest of notation adequacy is exploiting finiteness of considered worlds in order to get rid of some heavy formalism inherited from mathematics. A typical example of this would be hypergraphs: we see hypergraphs as sets of sets. Not mentioning the set of its vertices is not a problem since we can define it as the union of all its edges. This leads to significant simplifications in writings, and allows for tautological-looking definitions such as "a subhypergraph is a subset of an hypergraph". More generally, we will be only manipulating very simple objects: sets, tuples, natural numbers, and functions.

Preliminaries In a preliminary first section, we introduce notions having a classifying effect. On one hand, we introduce complexity-based notions of absolute easiness/hardness; on the other hand, we introduce a notion of comparative expressive power. The first one is described through the respective complexity classes of different problems we can associate to a query: *decision*, *count*, *enumeration*, and *Jeth*. For each problem, we consider three kinds of complexity classes we call *optimal*, *easy* and *tractable* that reflect different levels of tractability particularly relevant when considering database applications.

The second one is expressed by a simplistic yet essential reduction $\preccurlyeq$. This reduction can be thought as "is a particular case of"; it fits the idea of compared expressive power, which is more precise than compared complexity. Absolute hardness-based criteria are rougher than $\preccurlyeq$: while two queries reducing to each other have exactly the same complexities for all problems, two queries having the same complexity with regards to every problem may nevertheless still be incomparable under our reduction. Moreover, proving a reduction gives an algorithm; since all easiness results are proved with the reduction $\preccurlyeq$ and its extension $\prec$, this document provides all the algorithms granting the (numerous) proposed easiness results, with a greatly factorized code. Our reduction $\preccurlyeq$ also allows for defining another natural notion we need: the equivalence of two queries. Having this reduction between two problems done in both ways is an equivalence relation $\eqsim$ that reflects the intuitive idea that the two problems are the same *up to irrelevant encoding conventions*.

Around expressive power The second section develops on comparative expressive power, mostly on equivalence classes. First of all, we introduce queries in a slightly unusual way: we write them with *explicit distinct domains symbols* for each variable, which is a writing borrowed from Constraint Satisfaction Problems. This notational choice is strongly motivated: with this notation, every directed graph problem belongs to the equivalence class of some query, and so do some undirected graph

problems; this means such queries include many of the most natural problems. Furthermore, distinguishing relation symbols have no impact on expressive power. We show that, in atoms, only the set of variables matters. We investigate the notion of subquery, and prove that variables weak removal reflects such a notion, thanks to our reduction again.

We also state an informally known result about the most studied restriction of queries, *conjunctive* queries. We prove that their equivalence class only depends on an object reflecting how the variables are shared between atoms: the *hypergraph*. Furthermore, we give a notion of *pseudo-minor*[1] that is close to the intuitive notion of a minor, and such that any conjunctive query with a given hypergraph has more expressive power than any conjunctive query admitting a pseudo-minor of this hypergraph as its hypergraph. All these results are summarized in Theorem 8 and Theorem 9.

Getting a classification In a third section, we proceed to queries classification with regards to their complexity. The results of the previous section makes it really easy to focus on the essential. For each problem, we prove that acyclicity-based notions are equivalent to the different problems membership to the given complexity classes. In particular, we find Bagan's dichotomy [BDG07, Bag09] again expressed by a clearer statement, this time with no hypergraph conformity assumption, and under an irreducible canonical complexity hypothesis. Hardness proofs are all based on the previous section result on pseudo-minors; easiness proofs all rely on that any tractable existential conjunctive query belongs to the equivalence class of some quantifier-free conjunctive query; the easiness of these are proved in an *ad hoc* way for each problem: differences between problems are only considered then.

This sum of facts results in the concluding figure 4.5 which gives a clear — thanks to equivalence class — and complete picture of existential conjunctive query complexity for all the mentioned problems. This picture is striking in its simplicity, and in the fact that enumeration alone, if we also consider its all-lexicographical-order variant, can lead to the full classification, the other problems just "follow" this classification.

1 Optimal Enumeration Class and Reduction

First of all, we introduce complexity classes of interest: in particular, we will focus on time-optimal complexity class for enumeration problems. We will introduce the (very few) complexity tools we use that will make complexity statements and their respective proofs concise yet intuitive.

1.1 Optimal Enumeration Class

First of all, we introduce some naïve complexity class that we would like to reflect the intuitive idea of "time optimal for writing output", in particular, "time optimal for enumeration". The idea is to define the class of enumeration algorithms that only take the minimal required time to produce their output.

1 A similar notion was recently defined by [AGK12].

Definition 12 (ConstD) We call *constant-delay*, written ConstD, an enumeration problem for which there is an algorithm that produces objects such as the time between two consecutive objects a and b is $O(|a| + |b|)$. The beginning and the end of enumeration are considered as objects of constant size.

Example 5 Given two sets E and F, producing $\{(a, b) \in (E \cup F)^2 \mid a \neq b\}$ is in ConstD *if sets are implemented as sorted lists*:

```
Enum(E, F):
    S ← ∅
    for x ∈SortedListFusion(E, F) :
        for y ∈ S :
            yield (x, y)
            yield (y, x)
        S ← S ∪ {x}
```

Notice that:

- if sets are just implemented as lists, we supposedly cannot have such a low complexity;
- even ordered list with duplicates would not work — this input must be canonical;
- if sets are defined as vectors in $\{\top, \bot\}^n$, describing membership of elements $1, ..., n$ to the corresponding set, this still does not work;
- this would not work either if we had to compute another set operation instead of union;
- the algorithm is quite tricky considering the simplicity of the problem;
- the important property we want is just that the first solution must be generated in a time not depending on output size, i.e. not quadratic.

This shows how membership to this class depends on the chosen data structure, which is the main objection: *this class is not robust*. This example also shows other defects such as over-constraining definition and need for tricky thus needlessly complicated algorithms. As a consequence, we introduce a similar class *allowing a precomputation* in linear time, that can be used to produce convenient data structures. This gives this new class its robustness: the precomputation time allows for normalizing this input, putting it in the best fitting form.

Our definitions are based on the RAM model as defined and studied in a series of paper [Gra94a, Gra94b, Sch97, GS02, BG02, BG04, GO04, Gra96, DG07, BDGO08] which allows to define robust complexity classes, DLin for linear-time decision and ConstD∘Lin [BDGO08] for enumeration (see below), that is usually referred to as CD_{lin}. [2]

Definition 13 (ConstD∘Lin) We call *deterministic linear* (*linear*, for short) a problem in DLin (Lin, for short) such as defined in [Gra96, GS02, Sch97, GO04]. By extension, we consider that a function producing its result in time linear in input size is also in Lin.

We call ConstD∘Lin the class of enumeration problems for which there is an algorithm A proceeding to the enumeration at constant delay after linear precomputation, i.e. $A \in$ ConstD∘Lin iff $A = g \circ f$ with $g \in$ ConstD and $f \in$ Lin.

2 See also [GS89, Grä90, Reg93] for some discussions on linear time and quasi or nearly linear time complexity classes and their robustness properties.

Intuitively, this class consists in enumeration problems for which testing emptiness of the resulting set is done in linear time — which means optimal emptiness decision complexity — and generation of the resulting set has optimal delay, and starts just after decision time.

One particular aspect of this is that enumeration is exactly as easy as decision, since once decision is made (requiring only input reading), additional time is only devoted to writing the output. Therefore, no time is spent "thinking" without reading or writing, which gives its optimal aspect to this class.

To prove membership to ConstDoLin (directly, or with the reduction), we will only be needing the following complexity point:

▶ **Theorem 7 ([Gra96])**
Sorting is in Lin; therefore set operations (intersection, union, difference) and filtering/mapping with a linear function also are. ◀

Example 6 Given two sets E and F, producing $\left\{(a,b) \in (E \cap F)^2 \,\middle|\, a \neq b\right\}$ can be done in ConstDoLin:

$A(E,F)$:
    $G \leftarrow E \cap F$
    for $x \in G$:
        for $y \in G$:
            if $x \neq y$:
                yield (x,y)

Notice how simple and explicit this algorithm is, compared to the one of the previous example. We can check we match the definition:

$$A = \underbrace{E \mapsto \left\{(a,b) \in E^2 \mid a \neq b\right\}}_{g \in \text{ConstD}} \circ \underbrace{(E,F) \mapsto E \cap F}_{f \in \text{Lin}}$$

Definition 14 (optimal, easy, tractable) We define enumeration complexity classes other than ConstDoLin, that will allow for further classification. We introduce LogDoLin (logarithmic delay after linear precomp.), a variant of ConstDoLin where the precomputation still takes a linear time in the input size $|I|$, and the delay between production of successive objects a and b is in the form $\mathcal{O}((|a|+|b|)\log|I|)$. We say that an enumeration problem is in qLinD (quasi-linear delay) when the delay between two objects is $\mathcal{O}\big(|I|(\log|I|)^k\big)$ for some fixed k.

These complexity classes can be used to describe the complexity of problems other than enumeration: in fact any algorithm producing something can be thought as an enumeration algorithm; their complexity may admit a simpler description. We sum it up in the following table.

	Optimal ConstDoLin	Easy LogDoLin	Tractable qLinD
Decision	Lin	Lin	qLin
Enumeration	ConstDoLin	LogDoLin	qLinD
Counting	$\lvert I\rvert + \lvert O\rvert$	$\lvert I\rvert + \lvert O\rvert \log\lvert I\rvert$	$\lvert I\rvert.\lvert O\rvert(\log\lvert I\rvert)^k$
Jeth	ConstDoLin	LogDoLin	qLinD

Notice the implicit use of $\mathcal{O}$. In the cases we study, these will admit much simpler expressions, since the size of an output object can be bounded by $\mathcal{O}(\log|I|)$ in case of counting for example.

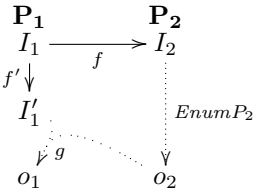
1.2 A Reduction

We now introduce the main tool for deriving complexity results: a reduction allowing to transfer membership to ConstDoLin between certain problems. This reduction is to enumeration what linear time reduction is to decision. This will be a little more complicated, but not much.

Definition 15 (nice reduction $\prec$) We call a *nice reduction* written $\prec$ from an enumeration problem P_1 to an enumeration problem P_2 a reduction in the following form, with $f, f' \in \text{Lin}$, $g \in \text{ConstD}$, and:

$$EnumP_1(I_1) = \biguplus_{o_2 \in EnumP_2(f(I_1))} g(f'(I_1), o_2)$$

where $g(f'(I_1), o_2)$ is always a non-empty set. or, more explicitly:




```
EnumP1(I1):
    I2 ← f(I1)
    I'1 ← f'(I1)
    for o2 ∈ EnumP2(I2):
        for o1 ∈ g(I'1, o2):
            yield o1
```

This algorithm is a valid enumeration algorithm, i.e. it generates all the solutions of P_1, each exactly once, from the solutions of P_2.

Notice that if f' is constant, the writings are much simpler. In the whole document, not mentioning f' means this is constant, and $g(I', o)$ is therefore written $g(o)$.

Example 7 We call a *(directed) graph* a set of ordered pairs, called edges. Edges are ordered pairs of elements called vertices. We define the problem EnumPath(k) which consists in enumerating the non-strict (directed) paths of length k of the graph. By non-strict, we mean a vertex can occur several times, and an edge can be used several times in the same path; that is to say a non-strict path of length k is a $k+1$-tuple of vertices $(x_0,...,x_k)$ such that (x_i, x_{i+1}) is an edge for every $i < k$. We will prove:

$$\text{EnumPath}(k) \prec \text{EnumPath}(k-1)$$

Proof. We just have to check the following is a nice reduction:

```
EnumPath(k)(G):
    G2 ← {(a,b) ∈ G | ∃c (b,c) ∈ G}
    G' ← G
    for (x0,...,xk−1) ∈ EnumPath(k−1)(G2):
        for (x0,...,xk) ∈ {(x0,...,xk) | (xk−1,xk) ∈ G'}:
            yield (x0,...,xk)
```

This writing makes it clear that:

$$\begin{aligned} f &= G &&\mapsto \{(a,b) \in G \mid \exists c\,(b,c) \in G\} \\ f' &= G &&\mapsto G \\ g &= G, (x_0,...,x_{k-1}) &&\mapsto \{(x_0,...,x_k) \mid (x_{k-1},x_k) \in G\} \end{aligned}$$

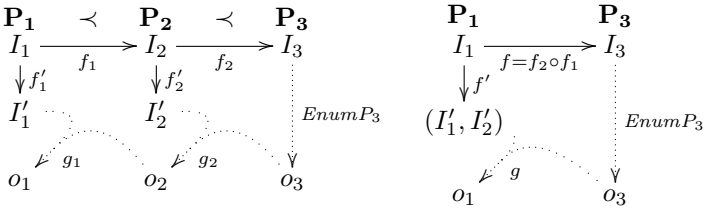
We can state expected properties, that are once more a transposition of those of the linear reductions for decision.

▶ ***Lemma 7*** The nice reduction is transitive: if $P_1 \prec P_2$ and $P_2 \prec P_3$, then $P_1 \prec P_3$.

The class ConstDoLin is closed under nice reduction:

$P_1 \prec P_2 \wedge P_2 \in \text{ConstDoLin} \Rightarrow P_1 \in \text{ConstDoLin}$

Proof. We will prove transitivity. We call f_1 and g_1 the parameters of the first reduction, and f_2 and g_2 those of the second one.



We can check the following is correct: $f = f_2 \circ f_1$, $f'(I) = (f'_1(I), f'_2(f_1(I)))$ and:

$g((I'_1, I'_2), o_3)$:
    for $o_2 \in g_2(I'_2, o_3)$:
        for $o_1 \in g_1(I'_1, o_2)$:
            yield o_1

Let us prove closure. Assume $P_2 \in \text{ConstDoLin}$ i.e. $P_2 = g_2 \circ f_2$ with $f_2 \in \text{Lin}$ and $g_2 \in \text{ConstD}$, and P_1 reduces to P_2 with parameters f_1, f'_1, g_1. The following proceeds in ConstDoLin.

$EnumP_1(I_1)$:
    $I'_1, I'_2 \leftarrow f'_1(I_1), f_2(f_1(I_1))$
    for $o_2 \in g_2(I'_2)$:
        for $o_1 \in g_1(I'_1, o_2)$:
            yield o_1

This concludes the proof. ◀

Example 8 We have seen $\text{EnumPath}(k) \prec \text{EnumPath}(k-1)$. By transitivity, $\text{EnumPath}(k) \prec \text{EnumPath}(1)$. Since $\text{EnumPath}(1) \in \text{ConstDoLin}$ (it consists in listing the edges) we finally have $\text{EnumPath}(k) \in \text{ConstDoLin}$.

This way of proving membership to ConstDoLin will be the *only one* we will use. First, prove successive reductions (recursive, in fact) up to a trivial problem; then prove directly membership of the trivial problem.

Note this incidentally gives *explicit* algorithms.

Definition 16 (nice reduction $\prec$ (cont.)) We write $P_1 \succ P_2$ to mean $P_2 \prec P_1$, we say in this case that P_1 *expresses* P_2.

If P_1 reduces *parsimoniously* to P_2 — i.e. for all I_1 and o_2, $g(f'(I_1), o_2)$ is a singleton set $\{o_1\}$ — we write $P_1 \preccurlyeq P_2$. If we also have $P_2 \preccurlyeq P_1$, then we write $P_1 \eqsim P_2$. We say in this case P_1 and P_2 are *equivalent*.

We call *closed class*, or *class* a set of problems closed by parsimonious reduction. In particular, reasonable time complexity classes are classes in this sense.

We call *equivalence class* a set of pairwise equivalent problems; in some sense elements of an equivalence class are different formulations of the same problem.

Example 9 Let us define the problem $\text{EnumColPath}(k)((G_1,...,G_k))$ which consists in enumerating every path of length k for which the first edge belongs to a given set of edges G_1, the second one to another set of edges G_2, etc. We have:

$\text{EnumPath}(2) \eqsim \text{EnumColPath}(2)$

Proof. Reduction of the first one to the second one is easy: $f = G \mapsto (G, G)$ and $g = x \mapsto \{x\}$. The other way is more tricky:

$$f = (G_1, G_2) \mapsto \{((x,1),(y,2)) \,|\, (x,y) \in G_1\} \uplus \{((x,2),(y,3)) \,|\, (x,y) \in G_2\}$$

and $g = ((x,1),(y,2),(z,3)) \mapsto \{(x,y,z)\}$. Let us prove the correctness of this reduction. For any input $(\mathcal{G}_1, \mathcal{G}_2)$ of EnumColPath(2) and its corresponding input $\mathcal{G} = f(\mathcal{G}_1, \mathcal{G}_2)$ of EnumPath(2), the following assertions are equivalent:

- $(X, Y, Z) \in \text{EnumPath}(2)(\mathcal{G})$;
- there exists (a unique) (x, y, z) such that $X = (x, 1)$, $Y = (y, 2)$, $Z = (z, 3)$, $(x, y) \in \mathcal{G}_1$, and $(y, z) \in \mathcal{G}_2$;
- the unique tuple (x, y, z) in the singleton $g((X, Y, Z))$ belongs to $\text{EnumColPath}(2)(\mathcal{G}_1, \mathcal{G}_2)$. ◀

We leave the proof of the following generalization as an exercise:

$$\text{EnumPath}(k) \eqsim \text{EnumColPath}(k)$$

This example is not innocent: a part of the proof of Lemma 12 below is a reformulation and generalization of the same trick.

Note that equivalent problems also have the same complexity for counting (and many other problems). We could say they are the *same* problem up to the encoding. Let us make it formal.

Définition 17 Let $\#P$ denote the problem of counting the output elements of the problem P: $\#P : I \mapsto \text{Card}(P(I))$ and $\star P$ denote the problem of producing the Jeth element of the output of the problem P (for some unspecified order). We formalize it as an enumeration problem : given a list of ranks $L = i_1, i_2, ...$, producing the list of the corresponding elements $P(I)[i_1], P(I)[i_2], ...$ of $P(I)$:

$$\star P : I \mapsto (L \mapsto [P(I)[i] \,|\, i \in L])$$

▶ ***Lemma 8***

$$P_1 \eqsim P_2 \;\Rightarrow\; \begin{cases} \#P_1 \underset{\text{Lin}}{\prec} \#P_2 \wedge \#P_2 \underset{\text{Lin}}{\prec} \#P_1 \\ \star P_1 \preccurlyeq \star P_2 \;\wedge\; \star P_2 \preccurlyeq \star P_1 \end{cases}$$

This means that the respective counting problems associated to P_1 and P_2 belong to the same time complexity class, provided that class is reasonable — stable under optimal (linear) time reduction; the same remark holds for the problems $\star P_1$ and $\star P_2$.

Proof. The functions f and g proving parsimonious reduction from P_1 to P_2 can be used to prove the reductions. The expression $\#P_1 = \#P_2 \circ f$ proves the first statement, and the second one is proved by the reduction itself:

$\star P_1(I_1)(L)$**:**
  for $o_2 \in \star P_2(f(I_1))(L)$**:**
    $\{o_1\} \leftarrow g(o_2)$
    yield o_1

which ends the proof. ◀

With these simplistic complexity tools, we will be able to prove easily dichotomic complexity results in the following sections.

Essentially, parsimonious reduction ($\preccurlyeq$) will allow to exhibit equivalence classes, and will be used to prove hardness results, while (non-parsimonious) reduction ($\prec$) is used to prove the easiness results.

2 Properties of Queries Classes

First of all, we introduce four classes of queries, from the most general to the most specific. In this section, we establish properties of the equivalence subclasses of these and in the next section, we will focus on giving explicit results about membership to what we call "easy classes": classes that are already optimal for decision, counting, and enumeration, but not necessarily for every problem.

	Equivalences Classes	Easy Class
Queries	sec.2.1	
Conjunctive Queries [3]	sec.2.2	sec.3.2.2
Existential C.Q.	sec.2.3.1	sec.3.2.1
Quantifier-Free C.Q.	sec.3.1	sec.3.1

2.1 Queries Definition and Basic Properties

2.1.1 Definitions and Examples

In this section, we define the notion of (first-order) query. This notion establishes a bridge between descriptive complexity — the logical expression describing a problem — and computational complexity of the problem thus defined. We will see that the class of problems definable by queries is rather natural.

In this section, instead of giving results of membership to a given class, we will give equivalence relations between problems, and mainly exhibit equivalence classes of problems defined by similar expressions. The main question we deal with in this section is "what in a given expression gives the related query its expressive power, and what does not matter?"

But first of all, we introduce a notion of structure in finite model theory, see [CK73, EF95, Imm99, Lib04, Kol03, GKL$^+$07].

Definition 18 (signature, structure) A *signature* σ is a set of relation symbols, together with an arity Ar that associates natural numbers to each symbol R of σ, therefore called its arity and written $\mathrm{Ar}(R)$. A *σ-structure* $\mathcal{S}$ or *structure of signature* σ consists in associating a set of $\mathrm{Ar}(R)$-tuples to each of the relation symbols R of σ which is called *interpretation of R in $\mathcal{S}$*, and written $R^{\mathcal{S}}$. Some relation symbols of arity 1 may be used in a particular way, and are then called *domain symbols*.

Example 10 If we define:

$$\mathcal{S} = \begin{cases} R \mapsto \{1, 3, 7\} \\ S \mapsto \{(1,4), (2,1), (5,5)\} \end{cases}$$

then we have $S^{\mathcal{S}} = \{(1,4), (2,1), (5,7)\}$ that is to say the interpretation of S in this structure $\mathcal{S}$ is $\{(1,4),...\}$. We have $\mathrm{Ar}(S) = 2$, meaning S has arity 2. Notice how we write tuples of length 1: we just give the element.

Definition 19 (query) A *first-order query* — *query* for short — ϕ is an expression describing a set, which is written in two parts.[4]

4 This is the same as writing sets with expressions like $\{f(a) \mid a$ satisfies a given criterion$\}$.

The first part is a tuple of variables–domain symbols pairs written $(x_1 \in D_1, ..., x_n \in D_n)$, with *distinct* domain symbols attached to different variables. It reflects that the defined set will be a subset of $D_1 \times ... \times D_n$.

The second part reflects the criterion that must be satisfied by a given element to be in the defined set. This criterion is given as what we call a relativized first-order formula $\psi(x_1, ..., x_n)$ over the variables $x_1, ..., x_n$ mentioned in the first part. By *relativized first-order formula*, we mean a first-order formula adapted to the finite world assumption; that is to say quantification is relativized with set membership like in $\exists x \in X \,|\, [...]$. To be formal, a formula $E(x_1, ..., x_n)$ belongs to the set of *relativized first-order formulas* we can build on a set of predicates σ — written $\mathrm{FO_r}(\sigma)$ — if, and only if it is in one of these forms:

- $\mathrm{Q}x \in P \,|\, E'(x_1, ..., x_n, x)$ with $\mathrm{Q} \in \{\exists, \forall\}$, $P \in \sigma$ and $E' \in \mathrm{FO_r}(\sigma \setminus \{P\})$;
- $\neg E'(x_1, ..., x_n)$ with $E' \in \mathrm{FO_r}(\sigma)$;
- $(E_1(x_1, ..., x_n))\ \mathrm{Op}\ (E_2(x_1, ..., x_n))$ with $\mathrm{Op} \in \{\vee, \wedge, \rightarrow, \leftrightarrow\}$ and $E_1, E_2 \in \mathrm{FO_r}(\sigma)$;
- $E'(x_{f(1)}, ..., x_{f(k)})$ with $f : \{1, ..., k\} \rightarrow \{1, ..., n\}$ and $E' \in \mathrm{FO_r}(\sigma)$.
- $R(x_1, ..., x_n)$ where $R \in \sigma$

The whole query is therefore in the form:

$$\phi = (x_1 \in D_1, ..., x_n \in D_n) \,|\, \psi(x_1, ..., x_n)$$

where $\psi \in \mathrm{FO_r}(\sigma)$ with $\{D_1, ..., D_i\} \cap \sigma = \emptyset$. Since ψ may be assumed in prenex form we can, without loss of generality, assume that the whole query is in the form:

$$\phi = (x_1 \in D_1, ..., x_n \in D_n) \,|\, Q_1\, x_{n+1} \in D_{n+1} ... Q_k\, x_{n+k} \in D_{n+k}\, \psi(x_1, ..., x_{n+k})$$

where ψ is a quantifier-free formula built on predicates in σ' where $\sigma' = \sigma \setminus \{D_1, ..., D_{n+k}\}$, which is written $\psi \in \mathrm{QF{-}FO}_r(\sigma')$.

Notice that we may in some cases manipulate queries whose domain symbols are non-pairwise distinct. We say in this case that the query is *odd*. We only do it when proving that, in fact, it does not matter. However, the statements of the theorems always assume domain symbols are pairwise distinct.

Example 11 This is a query:

$$\phi = (x_1 \in D_1, x_2 \in D_2) \,|\, \underbrace{R(x_1, x_2) \wedge \exists x_3 \in D_3\, S(x_2, x_3)}_{\psi(x_1, x_2)}$$

Notice that each variable has its own domain symbol. It can be put into prenex form:

$$\phi = (x_1 \in D_1, x_2 \in D_2) \,|\, \exists x_3 \in D_3\, \underbrace{R(x_1, x_2) \wedge S(x_2, x_3)}_{\psi'(x_1, x_2, x_3)}$$

Definition 20 (answering a query) Let:

$$\phi = (x_1 \in D_1, ..., x_n \in D_n) \,|\, Q_1 x_{n+1} \in D_{n+1} ... Q_k x_{n+k} \in D_{n+k} \psi(x_1, ..., x_n)$$

where $\psi \in \mathrm{QF{-}FO_r}(\sigma)$, and $\mathcal{S}$ be a $\sigma \uplus \{D_i\}$-structure. Answering the query ϕ on the structure $\mathcal{S}$ consists in producing the set defined by:

$$\left\{ (a_1 \in D_1^{\mathcal{S}}, ..., a_n \in D_n^{\mathcal{S}}) \;\middle|\; Q_1 x_{n+1} \in D_{n+1}^{\mathcal{S}} ... Q_k x_{n+k} \in D_{n+k}^{\mathcal{S}} \right.$$
$$\left. \mathcal{S} \vDash \psi\,(a_1, ...a_n, x_{n+1}, ..., x_{n+k}) \right\}$$

We write $\mathcal{Q}(\phi)$ the problem of answering a given query ϕ.

We consider this problem as a parameterized problem: given a fixed query, we have a particular enumeration problem.

Example 12 Many classical problems on graphs can be seen as instances of query answering. As an example, we consider the problem EnumTriangle which consists in enumerating each triangle in a given graph $\mathcal{G}$, i.e. producing:

$$\{\{x_1, x_2, x_3\} \mid \{x_1, x_2\} \in \mathcal{G} \wedge \{x_2, x_3\} \in \mathcal{G} \wedge \{x_3, x_1\} \in \mathcal{G}\}$$

We can see this expression is *not* a query, but it is equivalent to the *odd*[5] query $\mathcal{Q}\left((x_1 \in D, x_2 \in D, x_3 \in D) \mid R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_1, x_3)\right)$.

Proof. In one way:

$$f : \mathcal{G} \mapsto \mathcal{S} : \begin{cases} D \mapsto \mathcal{V}(\mathcal{G}) \\ R \mapsto \{(x, y) \mid \{x, y\} \in \mathcal{G} \wedge x < y\} \end{cases}$$

and $g : (a, b, c) \mapsto \{\{a, b, c\}\}$, where $\mathcal{V}(\mathcal{G})$ is the set of vertices of $\mathcal{G}$.

In the other way, we use the same kind of trick as we did for equivalence between EnumPath(k) and its colored version:

$$f : \mathcal{S} \mapsto \begin{cases} \{\{(a,1),(b,2)\} \mid (a,b) \in R^{\mathcal{S}}\} \\ \uplus \{\{(a,2),(b,3)\} \mid (a,b) \in R^{\mathcal{S}}\} \\ \uplus \{\{(a,1),(b,3)\} \mid (a,b) \in R^{\mathcal{S}}\} \end{cases}$$

and $g : \{(a,1),(b,2),(c,3)\} \mapsto \{(a,b,c)\}$ (Notice that this works only because the last atom is $R(x_1, x_3)$, with $R(x_3, x_1)$ it wouldn't be so easy.) ◀

In *this* case, we can make this query non-odd by proving its equivalence with the query:

$$\mathcal{Q}\left((x_1 \in V_1, x_2 \in V_2, x_3 \in V_3) \mid R(x_1, x_2) \wedge R(x_2, x_3) \wedge R(x_1, x_3)\right)$$

Proof. Reducing the first problem to the second one is obvious; the reverse reduction uses almost the same trick, except for the domains:

$$f : \mathcal{S} \mapsto \begin{cases} D \mapsto \begin{cases} \{(a,1) \mid a \in V_1^{\mathcal{S}}\} \\ \uplus \{(a,2) \mid a \in V_2^{\mathcal{S}}\} \\ \uplus \{(a,3) \mid a \in V_3^{\mathcal{S}}\} \end{cases} \\ R \mapsto \begin{cases} \{((a,1),(b,2)) \mid (a,b) \in R^{\mathcal{S}}\} \uplus \ldots \end{cases} \end{cases}$$

and $g : ((a,1),(b,2),(c,3)) \mapsto \{(a,b,c)\}$. ◀

Example 13 Parameterized problems can be described by parameterized queries. We can generalize the previous example to the problem consisting in enumerating cliques of a given size k present in a graph. We leave the proof as an exercise.

Now consider the slightly different problem of finding an independent set of a given size k. Open question: can we use the same trick to prove this is equivalent to the query:

$$\mathcal{Q}\left((x_1 \in V_1, \ldots, x_k \in V_k) \;\middle|\; \bigwedge_{i \leq j \leq k} \neg R(x_i, x_j)\right)$$

In the following, we will see two different kinds of properties: properties of *invariance*, showing that some aspects of query writing does not affect its expressive power; and *closure properties*, giving a transformation that changes a query into a query that is at least as easy. Such properties will be systematically studied for all considered classes of queries.

5 It is an odd query since x_1, x_2 and x_n have the same domain symbol D.

2.1.2 Class Invariance Property

Without loss of generality, any query considered in the rest of this document will be assumed to have pairwise distinct relation symbols, and every tuple of variables will be presented in increasing order of subscripts, with no repetition in the tuple. This is justified by the following invariance property.

Definition 21 (nice, simple, sorted) We say a query $\mathcal{Q}(\phi)$ is *nice* if, for any two different atoms A and B of ϕ, they hold a different tuple [6] of variables.

We say a query is *simple* (see[BDG07]) if every relation symbol occurs in at most one atom. Given a query $\mathcal{Q}(\phi)$, we say $\mathcal{Q}(\phi')$ is the *simple form* of $\mathcal{Q}(\phi)$ if ϕ' is obtained from ϕ by changing only relation symbols, such that $\mathcal{Q}(\phi')$ is simple.

We say a query is *sorted* is every tuple of variables is sorted with no repeating. Given a query $\mathcal{Q}(\phi)$, we say $\mathcal{Q}(\phi')$ is the *sorted form* of $\mathcal{Q}(\phi)$ if ϕ' is obtained from ϕ by replacing every tuple of variables by a tuple holding the same variables, in increasing order with no repetitions.

▶ ***Lemma 9*** A nice query is equivalent to the sorted form of its simple form. Another formulation is: let $\phi = (x_1,...,x_n)Q_1y_1...Q_my_m\, F(A_1,...,A_k)$ with:

$$A_i = R_{f(i)}(v_i) \text{ with } \begin{cases} v_i \in \{x_1,...,x_n,y_1,...,y_m\}^\star \\ f : \{1,...,k\} \to \mathbb{N} \end{cases}$$

where F is a propositional formula, i.e. $F(A_1,...,A_k)$ is built using *any* connectives, and atoms in $\{A_1,...,A_n\}$, and such that the v_i are *pairwise distinct*. We call s_i the set of variables occurring in v_i i.e. in A_i. The equivalence class of ϕ only depends on the "expression":

$$\{x_1,...,x_n\}\, Q_1y_1...Q_my_m\, F(s_1,...,s_k)$$

Proof. Let $\mathcal{Q}(\phi_1)$ a nice query. We call $\mathcal{Q}(\phi_2)$ the simple form of $\mathcal{Q}(\phi_1)$. We call $\mathcal{Q}(\phi_3)$ the sorted form of $\mathcal{Q}(\phi_2)$. We first prove $\mathcal{Q}(\phi_1) \backsimeq \mathcal{Q}(\phi_2)$ and then we prove $\mathcal{Q}(\phi_2) \backsimeq \mathcal{Q}(\phi_3)$.

It is obvious that $\mathcal{Q}(\phi_1) \preccurlyeq \mathcal{Q}(\phi_2)$. Let us prove $\mathcal{Q}(\phi_2) \preccurlyeq \mathcal{Q}(\phi_1)$. In ϕ_1, we call $v(i,j)$ the subscript of the $j$eth variable hold by the $i$eth atom, $r(i)$ its relation symbol, $n(i)$ its length. For example, in a formula whose Jeth atom is $S(x_7,x_1,x_3,x_3)$, $v(i,1) = 7$, $r(i) = S$, $n(i) = 4$. Given a structure $\mathcal{S}_2$, we need to build a structure $\mathcal{S}_1$ such that $\mathcal{Q}(\phi_1)(\mathcal{S}_1) = \mathcal{Q}(\phi_2)(\mathcal{S}_2)$. To do so, we define:

$$R^{\mathcal{S}_1} = \bigcup_i \left\{ \left((a_1,v(i,1)),...,(a_{n(i)},v(i,n(i)))\right) \,\middle|\, (a_1,...,a_{n(i)}) \in r(i)^{\mathcal{S}_1} \right\}$$

and we convert the domains: $D_i^{\mathcal{S}_2} = D_i^{\mathcal{S}_1} \times \{i\}$. Since $\mathcal{Q}(\phi_1)$ is nice, we have a "superposition" of relations without "interferences". Defining g as $((a_1,b_1),...,(a_n,b_n)) \mapsto \{(a_1,...,a_n)\}$ concludes.

In order to prove $\mathcal{Q}(\phi_2) \backsimeq \mathcal{Q}(\phi_3)$, we prove two steps : First, we prove that $\mathcal{Q}(\phi_2)$ is equivalent to the query $\mathcal{Q}(\phi_2')$ obtained from ϕ_2 by removing multiple occurrences of a variable. Then we prove this last query is equivalent to $\mathcal{Q}(\phi_3)$. In order to prove that $\mathcal{Q}(\phi_2)$ and $\mathcal{Q}(\phi_2')$ are equivalent, we only have to prove that a *simple* query where a variable occurs twice in a given atom is equivalent to the same query where the second occurrence of the variable has been removed.

6 The tuples (x,y) and (y,x) are different, so are (x,y) and (x,x,y).

Assume we have any simple query ϕ whose quantifier-free part is ψ, with pairwise distinct relation symbols, and let $R(x_{i(1)},...,x_{i(n)})$ be an atom of ψ. Assume we have two variables that are the same, i.e. $i(a) = i(b)$, $a < b$, for some a and b. We call ϕ' the query obtained from ϕ by substituting the atom by $R(x_{i(1)},...,x_{i(b-1)},x_{i(b+1)},...,x_{i(n)})$. Let us prove $\mathcal{Q}(\phi)$ and $\mathcal{Q}(\phi')$ are equivalent (by $\eqsim$). To do so, we just have to prove the left to right reduction:

$$R^{\mathcal{S}'} = \left\{ \left(a_{i(1)},...,a_{i(b-1)},a_{i(b+1)},...,a_{i(n)}\right) \;\middle|\; \left(a_{i(1)},...a_{i(n)}\right) \in R^{\mathcal{S}} \wedge a_{i(a)} = a_{i(b)} \right\}$$

Applying repeatedly this fact allows to prove $\mathcal{Q}(\phi_2) \eqsim \mathcal{Q}(\phi'_2)$.

Now we prove $\mathcal{Q}(\phi'_2) \eqsim \mathcal{Q}(\phi_3)$. In order to do so, we also proceed to a series of equivalences. Now let π be a permutation of $\{i(1),...,i(n)\}$, and ϕ any simple query. We call ϕ' the query obtained from ϕ by substituting the atom by $R\left(x_{\pi(i(1))},...,x_{\pi(i(n))}\right)$. Let us prove $\mathcal{Q}(\phi)$ and $\mathcal{Q}(\phi')$ are equivalent:

$$R^{\mathcal{S}'} = \left\{ \left(a_{\pi(i(1))},...,a_{\pi(i(n))}\right) \;\middle|\; \left(a_{i(1)},...a_{i(n)}\right) \in R^{\mathcal{S}} \right\}$$

and $g : (a_1,...,a_n) \mapsto \left\{\left(a_{\pi^{-1}(1)},...,a_{\pi^{-1}(n)}\right)\right\}$ does it. Applying this fact for each atom proves $\mathcal{Q}(\phi'_2) \eqsim \mathcal{Q}(\phi_3)$. ◀

Example 14 Vertex-colored and edge-colored versions of the triangle enumeration problem are equivalent to the basic version of this problem. All these are equivalent (in sense $\eqsim$):

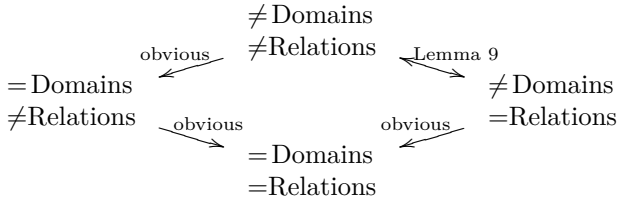
1. EnumTriangle(G)
2. $\mathcal{Q}((x_1 \in D\ , x_2 \in D\ , x_3 \in D\ )\ R\ (x_1,x_2) \wedge R\ (x_2,x_3) \wedge R\ (x_1,x_3))$
3. $\mathcal{Q}((x_1 \in D_1, x_2 \in D_2, x_3 \in D_3)\ R\ (x_1,x_2) \wedge R\ (x_2,x_3) \wedge R\ (x_1,x_3))$
4. $\mathcal{Q}((x_1 \in D_1, x_2 \in D_2, x_3 \in D_3)\ R_1(x_1,x_2) \wedge R_2(x_2,x_3) \wedge R_3(x_1,x_3))$
5. $\mathcal{Q}((x_1 \in D_1, x_2 \in D_2, x_3 \in D_3)\ R_1(x_1,x_2) \wedge R_2(x_2,x_3) \wedge R_3(x_3,x_1))$
6. $\mathcal{Q}((x_1 \in D_1, x_2 \in D_2, x_3 \in D_3)\ R\ (x_1,x_2) \wedge R\ (x_2,x_3) \wedge R\ (x_3,x_1))$

Proof.

- (1)$\eqsim$(2): By definition, justified in previous example.
- (2)$\eqsim$(3): This was proved in previous example. This is not easily generalized: we use the asymmetric aspect of $\{(1,2),(2,3),(1,3)\}$ which would not work for: $\{(1,2),(2,3),(3,1)\}$.
- (3)$\eqsim$(4)$\eqsim$(5)$\eqsim$(6): By previous lemma.

Notice that this proves that finding a triangle in an undirected graph and finding  in a directed graph are equivalent, and that they both express finding a directed triangle, i.e.  , in a directed graph, while the converse probably does not hold.

Remark 15 (about symbol diversity) The case of the triangle illustrates that in some cases, symbol diversity brings no expressive power. In the general case, all we know is that:



where the arrow means "expresses parsimoniously" i.e. $\succcurlyeq$; and that, when a query with one domain and one relation symbol is able to express

the same with different domains, they are all equivalent, which is the case for the clique-of-size-k problem, in particular the triangle problem.

From now on, in most cases we will omit to mention the domains of the variables of a query in most cases. Typically, the expression $(x_1,...,x_n)\,\psi$ stands for the query $(x_1 \in D_1,...,x_n \in D_n\,\psi)$.

2.1.3 Closure Property

Here we give a first idea of what a subquery is. We prove (easily) that, if we take any query $\mathcal{Q}(\phi)$, and remove some variable x, i.e. we remove *any* occurrence of x from ϕ, then the obtained query $\mathcal{Q}(\phi')$ is *easier* than $\mathcal{Q}(\phi)$. This means variable removal reflects the idea of subquery.

▶ ***Lemma 10*** Let ϕ be a *nice* query, and x be a variable appearing in the formula of ϕ. Let ϕ' be the query obtained by removing every occurrence of x in ϕ, that is to say removing the $Q_x x \in D_x$ if x is quantified, or removing x from the output part, and removing x in atoms where it occurs. Then $\mathcal{Q}(\phi)$ expresses parsimoniously $\mathcal{Q}(\phi')$.

Proof. By virtue of Lemma 9, we can assume that every relation appears once in ϕ, with associated variables in increasing order. The variable x is assumed to be the minimal element for the order. To define f, we will just define how a given relation R is transposed between $\mathcal{S}_1$ and $\mathcal{S}_2$. If x does not appear in the corresponding atom, there is no difference i.e. $R^{\mathcal{S}_2} = R^{\mathcal{S}_1}$.

In the other case, R appears as $R(x, x_{a(1)},..., x_{a(k)})$ in ϕ, with $a : \{1,...,k\} \to \{1,...n\}$. R appears as $R(x_{a(1)},..., x_{a(k)})$ in ϕ'. From now on, defining the reduction is easy:

$$R^{\mathcal{S}_2} = \left\{(1, s_1,..., s_k) \,\middle|\, (s_1,..., s_k) \in R^{\mathcal{S}_1}\right\}$$

Defining $D_x^{\mathcal{S}_2} = \{1\}$ ends the definition of f. If x was quantified, take $g : y \mapsto \{y\}$, else $g : (a, b_1,..., b_n) \mapsto \{(b_1,..., b_n)\}$. ◀

2.2 Conjunctive Queries

A well-studied class of queries are *conjunctive queries*. We define them now in a generalized version with quantifiers $\exists$ and $\forall$.

2.2.1 Definition

Definition 22 (Conjunctive Queries) A *conjunctive query* — CQ for short — is a query in prenex form for which the quantifier-free formula part is a conjunction of atoms. A conjunctive query is said *existential* — ECQ for short — when the quantification part is only composed of existential quantifiers.[7] A conjunctive query is said *quantifier-free* — QFCQ for short — when the quantification part is empty.

We will prove the complexity of a conjunctive query is only determined by what we call the hypergraph of the query, which reflects only the way variables are shared between atoms of the conjunction.

Example 16 This is a simple example of a quantifier-free conjunctive query (QFCQ):

7 Note that our class of conjunctive queries is larger than ECQ which is the classical class of conjunctive queries.

$$\phi = (x_1, x_2, x_3, x_4) \mid R_1(x_1, x_3) \wedge R_1(x_2, x_2) \wedge R_2(x_4, x_1, x_3) \wedge R_3(x_1, x_3, x_4)$$

Notice that, in the above example, the third and the fourth atoms hold the same variables, but in different order; that R_1 appears twice with different variables, and finally that a variable may appear several times in the same atom. We will see later that none of those points matters.

Example 17 Some previously defined problems can be formulated as conjunctive queries:

$$\text{EnumPath}(k) = \mathcal{Q}\left((x_1, ..., x_k) \;\middle|\; \bigwedge_{1 \leq i < k} R(x_i, x_{i+1})\right)$$

$$\text{EnumColPath}(k) = \mathcal{Q}\left((x_1, ..., x_k) \;\middle|\; \bigwedge_{1 \leq i < k} R_i(x_i, x_{i+1})\right)$$

Remark 18 (on domains) Conjunctive queries are traditionally defined without domains, because domains can be easily inferred from relations — provided every variable appears in the conjunction. We have three major objections to this way of doing — in addition to its uselessness:

- Variables not appearing in the conjunction stay undefined.
- We cannot generalize this with $\forall$ quantifiers, since they do not make sense anymore.[8]
- Our Lemma 12 (see below) does not hold anymore, even if we could fix it dirtily, causing its statement, and the statement of all its consequences to become really unclear.

Using this not-so-handy shorthand does finally not seem to be a so good idea.

► ***Lemma 11*** A conjunctive query $\mathcal{Q}(\phi)$ is always equivalent to another conjunctive query $\mathcal{Q}(\phi')$ where all the variables explicitly appear in the conjunction.

Proof. We call ψ the quantifier-free part of ϕ. Build ϕ' as follows: make it a copy of ϕ for now. In ϕ', for each variable x_i of domain D_i not in ψ, append $\wedge D_i(x_i)$ to ψ. Equivalence is then obvious. ◄

From now on, we always assume variables all appear in the conjunction.

2.2.2 Class Invariant: Formula Hypergraph

We will see that the expressive power of a conjunction is neither dependent on the order of variables in atoms, nor on the order of atoms. Moreover, duplicates do not matter. To express this, we introduce an object reflecting all that matters: the way variables are shared between atoms. This object is called a hypergraph.

Definition 23 (hypergraph notations) We call *hypergraph* a set of non-empty sets, namely the *edges*. We call $\mathcal{V}(\mathcal{H})$ the union of edges; its elements are called *vertices*.

Let ϕ be a conjunctive formula whose conjunction is:

$$\bigwedge_{i \in I} R_{f(i)}(x_{g(1,i)}, ..., x_{g(n(i),i)})$$

We call *hypergraph of this query*, written $\mathcal{H}(\phi)$, the hypergraph:

$$\left\{\{x_{g(1,i)}, ...x_{g(n(i),i)}\} \;\middle|\; i \in I\right\}$$

8 This point is strongly related to the discussion in [AHV95], chapter 5 about the so-called sane queries in the relational calculus of databases.

Example 19 Let:

$$\phi = (x_1, x_2, x_3) \mid \exists x_4 \; R_1(x_1, x_3) \wedge R_1(x_2, x_2) \wedge R_2(x_4, x_1, x_3) \wedge R_3(x_1, x_3, x_4)$$
$$\phi' = (x_1, x_2, x_3) \mid \exists x_4 \; R_1(x_1, x_3) \wedge R_3(x_2) \wedge R_2(x_1, x_3, x_4)$$

Then $\mathcal{H}(\phi) = \mathcal{H}(\phi') = \{\{x_1, x_3\}, \{x_2\}, \{x_1, x_3, x_4\}\}$. The following lemma will establish that consequently they are equivalent.

▶ ***Lemma 12*** Let $\phi = (x_1,...,x_n)Q_1y_1...Q_my_m\psi(x_1,...,x_n,y_1,...,y_m)$ be a conjunctive query. The equivalence class of $\mathcal{Q}(\phi)$ depends only on the quantifier string $Q_1y_1...Q_my_m$ and on the hypergraph $\mathcal{H}(\psi)$.

Proof. Let:

$$\phi = (x_1,...,x_n)\, Q_1y_1...Q_my_m\, \psi(x_1,...,x_n,y_1,...,y_m)$$

with $\psi(x_1,...,x_n,y_1,...,y_m) = \bigwedge_i A_i$ where:

$$A_i = R_{f(i)}(v_i) \text{ with } \begin{cases} v_i \in \{x_1,...,x_m,y_1,...,y_m\}^\star \\ f : \{1,...,k\} \to \mathbb{N} \end{cases}$$

We can assume w.l.o.g. that the v_i are pairwise distinct (i.e. the query is nice): if two atoms A_i and A_j hold the same tuple of variables, then this query is trivially equivalent to the same query where A_j has been removed.

We therefore can apply Lemma 9, the equivalence class of $\mathcal{Q}(\phi)$ only depends on:

$$\{x_1,...,x_n\}\, Q_1y_1...Q_my_m\, (s_1,...,s_k)$$

where each s_i is the set of variables of v_i (i.e. of A_i). Notice the formula doesn't have to be mentioned, because we know it is a conjunction. Since the conjunction is commutative, the equivalence class of $\mathcal{Q}(\phi)$ finally only depends on:

$$\{x_1,...,x_n\}\, Q_1y_1...Q_my_m\, \underbrace{\{s_1,...,s_n\}}_{\mathcal{H}(\psi)}$$

Furthermore, all variables appear in the conjunction by Lemma 11. Consequently, we have $\{x_1,...,x_n\} = \mathcal{V}(\mathcal{H}(\psi)) \setminus \{y_1,...,y_m\}$, the set $\{x_1,...,x_n\}$ therefore does not need to be mentioned (it can be inferred), which concludes. ◀

Remark 20 Applying Lemma 8 to the previous results gives:

$$\mathcal{H}(\phi_1) = \mathcal{H}(\phi_2) \;\Rightarrow \begin{cases} \#\text{Query}(\phi_1) \underset{\text{Lin}}{\prec} \#\text{Query}(\phi_2) \\ \text{JethQuery}(\phi_1) \preccurlyeq \text{JethQuery}(\phi_2) \end{cases}$$

The basic fact that same equivalence class implies same complexity class for any problem gives the equivalence its interest. Every equivalence result must therefore be seen through this consequence.

2.2.3 Class Properties Written with Hypergraphs

First of all, we re-write previously defined class closure with hypergraphs.

Definition 24 (hypergraph induction) We call induction of a hypergraph $\mathcal{H}$ on $S \subseteq \mathcal{V}(\mathcal{H})$ written $\mathcal{H}[S]$, the following transformation:

$$\mathcal{H}[S] = \{e \cap S \mid e \in \mathcal{H}\} \setminus \{\emptyset\}$$

$\mathcal{H}[S]$ is the hypergraph obtained by keeping only the vertices in S. We write $\mathcal{H}[\setminus S]$ to mean $\mathcal{H}[\mathcal{V}(\mathcal{H}) \setminus S]$. The hypergraph $\mathcal{H}[\setminus S]$ is obtained from $\mathcal{H}$ by removing the vertices in S. The two first columns of figure 4.3 illustrate induction.

From previous lemma, it becomes relevant to extend hypergraph notations directly to conjunctive formulas.

Example 21 Definition of induction gives:

$$\{\{x\},\{x,y\},\{y,z\}\}\,[\{x,z\}] \;=\; \{\{x\},\{z\}\}$$

Extending hypergraph notations to conjunctive queries consists in writing, in the same spirit:

$$((x,y)\,|\,\exists z\; R(x)\wedge S(x,y)\wedge T(y,z))[\{x,z\}] \;=\; (x)\,|\,\exists z\; R(x)\wedge S(x)\wedge T(z)$$

► ***Lemma 13*** A conjunctive query expresses parsimoniously every induced sub-query:

$$\forall S\; \mathcal{Q}(\phi[S]) \preccurlyeq \mathcal{Q}(\phi)$$

Proof. This is proved by applying Lemma 10 to all variables that are not in S. ◄

We have seen that a query expresses parsimoniously every induced subquery, but note that, even in the case of conjunctive queries, $\mathcal{Q}(\phi')$ with $\phi' \subset \phi$ (the conjunction of ϕ' is a subset of the conjunction of ϕ) does *not* necessarily reduces to $\mathcal{Q}(\phi)$. This shows an induced subquery is expressed in a query, but a subquery which is just a subset of the given query is not. Induction therefore best describes what a subquery is.

Example 22 Applying this lemma to Example 21 leads to:

$$\mathcal{Q}\,((x)\,|\,\exists z\; R(x)\wedge S(x)\wedge T(z)) \;\preccurlyeq\; \mathcal{Q}\,((x,y)\,|\,\exists z\; R(x)\wedge S(x,y)\wedge T(y,z))$$

We introduce a class invariant property, also based on hypergraphs.

Definition 25 (minimization $\mathrm{r}_\mathrm{e}^\star$) We call r_e the operation consisting in removing an edge included in another edge, that is to say $\mathcal{H} \leftarrow \mathcal{H} \setminus \{e\}$ where $\exists e' \in \mathcal{H} \mid e \subset e'$. We call *minimization*, written $\mathrm{r}_\mathrm{e}^\star$ (or M in chapters 6 and 7) the operation consisting in applying r_e until not possible. We define these operations similarly for conjunctive queries.

The following lemma proves that this operation does not affect corresponding query complexity.

► ***Lemma 14 (minimization equivalence)*** For every ECQ $\mathcal{Q}(\phi)$, we have (1) $\mathcal{Q}(\phi) \eqsim \mathcal{Q}(\mathrm{r}_\mathrm{e}(\phi))$ and (2) $\mathcal{Q}(\phi) \eqsim \mathcal{Q}(\mathrm{r}_\mathrm{e}^\star(\phi))$.

Proof.

1 We have to prove:

$$\mathcal{Q}(...\wedge R(x_1,...,x_n)) \eqsim \mathcal{Q}(...\wedge R(x_1,...,x_n)\wedge S(x_{i(1)},...,x_{i(k)}))$$

with $\{i(j)\,|\,j\in\{1,...,k\}\}\subseteq\{1,...n\}$. In both directions, $g = x \mapsto \{x\}$, and $f = \mathcal{S}_1 \mapsto \mathcal{S}_2$ with $R_i^{\mathcal{S}_2} = R_i^{\mathcal{S}_1}$ for all relations R_i except for R and S. These are treated differently: in the case of $\succcurlyeq$, we perform filtering:

$$R^{\mathcal{S}_2} = \{(a_1,...,a_n)\in R^{\mathcal{S}_1}\,|\,(a_{i(1)},...,a_{i(k)})\in S^{\mathcal{S}_1}\}$$

while in the case of $\preccurlyeq$, we compute a projection:

$$S^{\mathcal{S}_2} = \{(a_{i(1)},...,a_{i(k)})\,|\,\exists ...\,(a_1,...,a_n)\in R^{\mathcal{S}_1}\}$$
$$R^{\mathcal{S}_2} = R^{\mathcal{S}_1}$$

2 this is a direct consequence of (1) and of reduction transitivity (Lemma 7). ◄

2.3 Additional Properties and Summary

2.3.1 Existential Conjunctive Queries Pseudo-Minor Closure

We have seen that conjunctive queries properties rely on a notion of hypergraph. We therefore express closure properties of these queries as closure properties of their associated hypergraphs. The main point consists in defining the widest possible notion of hypergraph minor such that any conjunctive query complexity class is minor-closed. We have seen induction is an operation satisfying this condition. We will see another one, corresponding to edge contraction for graphs.

This leads to a simple definition of subquery making the following statement true: "A query expresses any of its subqueries".

This will prove to be quite an essential tool for deriving hardness results. We will give in section 3 a family of "canonical hard queries" such that any "hard" query expresses one of these.

Definition 26 (edge contraction) We call *edge contraction* of a hypergraph $\mathcal{H}$ the operation consisting in replacing all occurrences of some vertex y by another fixed vertex x, provided x and y both belong to some *common* edge. We extend this definition to ECQ.

Exemple 23 The contraction of the atom $R_1(x,y,z)$ in the ECQ

$$(x,y,t)\,|\,\exists z R_1(x,y,z)\wedge R_2(y,t)$$

by replacing y by x (resp. z by y) gives:

$$(x,t)\,|\,\exists z R_1(x,z)\wedge R_2(x,t) \quad \big(\text{resp. } (x,y,t)\,|\,R_1(x,y)\wedge R_2(y,t)\big)$$

▶ ***Lemma 15*** Existential Conjunctive Query complexity classes are closed under edge contraction, i.e. if some ECQ $\mathcal{Q}(\phi')$ is obtained from some ECQ $\mathcal{Q}(\phi)$ by edge contraction, then $\mathcal{Q}(\phi) \succcurlyeq \mathcal{Q}(\phi')$.

Proof. Assume ϕ' is obtained by contracting the atom $R(x,y,z_1,...,z_n)$ in ϕ by the replacement of y by x in ϕ. Then the reduction only consists in setting $D_x^{\mathcal{S}_2} = D_y^{\mathcal{S}_1}$, setting $R^{\mathcal{S}_2} = \big\{(a,a,b_1,...,b_n)\,\big|\,(a,b_1,...,b_n)\in R^{\mathcal{S}_1})\big\}$ and keep other relations unchanged. ◀

Definition 27 (pseudo-minor) We say that $\mathcal{H}'$ is a *pseudo-minor* (or, for short, *minor*) of $\mathcal{H}$ if $\mathcal{H}' = \mathcal{H}$ or if we can build $\mathcal{H}'$ by:

- removal of an arbitrary vertex of some pseudo-minor of $\mathcal{H}$,
- edge contraction of some pseudo-minor of $\mathcal{H}$,
- or removal of an arbitrary edge of some pseudo-minor of $\mathcal{H}$, *provided some other edge includes it.*

Note that the "only" difference with the standard minor definition on graphs (see [Die10, Ber69, BM08, CLZ10, Lov06]) consists of the "provided some other edge includes it" condition.

We have just (Feb. 2013) discovered that almost the same notion of hypergraph minor has been defined and studied by [AGK12].

▶ ***Lemma 16 (pseudo-minor)*** A query of hypergraph $\mathcal{H}$ expresses parsimoniously every query of hypergraph a pseudo-minor of $\mathcal{H}$.

Proof. By Lemma 13, Lemma 14, and Lemma 15. ◀

2.3.2 Summary

In this section, we have shown how certain aspects of formula expression do not affect the membership to a given equivalence class; another way of saying is that only few aspects of a formula define the equivalence class of the associated query:

▶ **Theorem 8 (equivalence classes)**

Let $\phi = (x_1,...,x_n)Q_1y_1...Q_my_mF(A_1,...,A_k)$ be any query with $A_i = R_{f(i)}(v_i)$ with $v_i \in \{x_1,...,x_n,y_1,...,y_m\}^\star$ and $f : \{1,...,k\} \to \mathbb{N}$. We call s_i the set of variables appearing in v_i (in A_i). If ϕ is a [...] then its equivalence class depends only on [...]:

- general Query $\{x_1,...,x_n\}Q_1y_1...Q_my_mF(s_1,...,s_n)$
- Conjunctive Query $Q_1y_1...Q_my_m\{s_1,...,s_n\}$
- Existential Conjunctive Query $\{x_1,...,x_n\}\{s_1,...,s_n\}$
- Quantifier-Free Existential Conjunctive Query $\{s_1,...,s_n\}$

Proof. We justify each line :

- Line 1 is given by Lemma 9.
- Line 2 is given by Lemma 12.
- Line 3 is the same as the previous one, except that quantifiers are existential, so their order does not matter. We therefore just need to know which variable is quantified and which is not; we choose to give those that are not.
- Line 4 is the same except every variable is in the output part. ◀

Additionally, we have seen invariance and closure properties:

▶ **Theorem 9 (invariance and closure)**

The following query families exhibit invariance properties (for $\approx$) and closure properties (for $\preccurlyeq$). These are summed up here: ◀

	Invariance	Closure
general Queries		variable removal [Lemma 10]
Conjunctive Queries	minimization $\mathrm{r}_{\mathrm{e}}^\star$ [Lemma 14]	hypergraph induction [Lemma 13]
Existential C.Q.		edge contraction [Lemma 15] pseudo-minor [Lemma 16]

2.3.3 Queries Complexity Classes

Recall that Definition 14 defines *optimal* as the complexity class ConstDoLin (constant delay after a linear precomputation), *easy* as LogDoLin (logarithmic delay after a linear precomputation), and *tractable* as qLinD (quasi-linear $|\mathcal{S}|(\log|\mathcal{S}|)^{\mathcal{O}(1)}$ delay). In the particular case of queries, we can simplify complexities of related problems accordingly.

	Optimal	Easy	Tractable
Decision, Counting	Lin	Lin	qLin
Enumeration, Jeth	ConstDoLin	LogDoLin	qLinD

Notice that the differences for certain problems may become slight — Optimal Decision Vs Tractable Decision — and even nonexistent — e.g. Optimal Count Vs Easy Count. As we will see, the distinction between easy Jeth and optimal Jeth will have importance.

3 Conjunctive Queries and Acyclicity

3.1 Quantifier-Free Conjunctive Queries and Acyclicity

We now try to answer the following question: which queries are optimal, i.e. in ConstDoLin ? We have seen the equivalence class of a quantifier-free conjunctive query (QFCQ) relies only on the hypergraph of its formula.

In a first time, we give a known (c.f. [BDG07]) hypergraph property that is both necessary and sufficient for the corresponding queries to be easy: this is called α-*acyclicity*, and is also known as the criterion such that decision is optimal or easy — meaning linear.

In a second time, we extend and refine this result: we give a similar dichotomy for the existential case, we introduce the issue of lexicographically–ordered enumeration, and finally introduce the so-called *trivial* class, that can't express anything interesting but is the only one allowing optimal complexity for every associated problem.

3.1.1 Acyclicity: Definition and Properties

First, let us define several notions of reduction of a hypergraph by edge or vertex removal.

Definition 28 ($\mathrm{r_e}$, $\mathrm{r_v}$, $\mathrm{r}^\star$) We call the *remainder* of a hypergraph $\mathcal{H}$, written $\mathrm{r}^\star$, the hypergraph obtained after applying the following operations until none can be performed:

- $\mathrm{r_v}$ Remove an *isolated* vertex, that is to say $\mathcal{H} \leftarrow \mathcal{H}[\diagdown\{x\}]$ where x only belongs to *one edge* of $\mathcal{H}$.
- $\mathrm{r_e}$ — as previously defined — remove an edge included in another edge, that is to say $\mathcal{H} \leftarrow \mathcal{H} \diagdown \{e\}$ where $\exists e' \in \mathcal{H} \mid e \subset e'$.

$\mathrm{r_e}^\star$ was defined as the operation consisting in applying $\mathrm{r_e}$ until not possible; similarly, we define $\mathrm{r_v}^\star$. We *admit* (for now) the remainder is the same regardless the order in which transformations $\mathrm{r_v}$ and $\mathrm{r_e}$ are performed. We therefore could write $\mathrm{r}^\star = (\mathrm{r_e}^\star \mathrm{r_v}^\star)^\star$.

This is the original GYO (Graham, Yu, Ozsoyoglu) non deterministic algorithm described by [Gra79, YO79] (see also [AHV95] ex. 5.29 p 151 and [FMU82]).

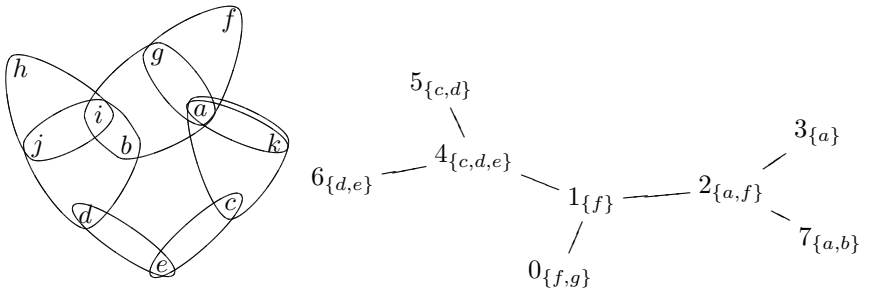
In order to reflect the order in which vertices are removed during this process, we introduce a notion we call *elimination order*, defined inductively as follows.

Definition 29 (elimination order) The empty tuple is the only elimination order of the empty hypergraph. If $\mathcal{H}$ is non-empty, $(x_1,...,x_n)$ is a *reverse elimination order* of $\mathcal{H}$ iff one of the following hold:

- $(x_1,...,x_n)$ is a reverse elimination order of $\mathrm{r_e}(\mathcal{H})$, or
- x_n is isolated in $\mathcal{H}$ and $(x_1,...,x_{n-1})$ is a reverse elimination order of $\mathcal{H}[\diagdown\{x_n\}]$.

Note that we may simply say *elimination order* for *reverse* elimination order. Notice that, with this definition, an elimination order of $\mathcal{H}$ is an elimination order of $\mathrm{r_e}(\mathcal{H})$ and conversely. Furthermore, it is easy to see that a hypergraph is acyclic iff it admits an elimination order (see Theorem 10).

We introduce notation required to state the standard result we use as a starting point. This result will consist in three equivalent statements describing acyclicity.



● **Figure 4.1:** Illustration of Definition 30. The first figure represents a hypergraph, where (c, e, d, b, a) is a chordless cycle. The second one is a join tree, whose nodes are written $x_{l(x)}$.

Definition 30 (standard definitions) We call *join tree* a tree with a labeling function l which maps vertices of the tree to sets which satisfies the following so-called *join property*: for any pair of vertices of the tree (a, b), any vertex c on the path between the two is such that $l(a) \cap l(b) \subseteq l(c)$. We call *join tree associated with a hypergraph* $\mathcal{H}$ a join tree for which the set of images of vertices through l is $\mathcal{H}$, and l is injective. We say that a hypergraph is acyclic when it has a join tree.

We say that two vertices x and y are *neighbours* in $\mathcal{H}$ when $\exists e \in \mathcal{H}$ such that $\{x, y\} \subseteq e$. We call *clique* of $\mathcal{H}$ a set K of pairwise neighbours in $\mathcal{H}$. A hypergraph is said to be *conformal* when every clique is contained in some edge. We call *(chordless) cycle* of $\mathcal{H}$ a tuple $(x_1,...,x_n)$ such that the set of neighbours pairs is exactly $\{\{x_i, x_{i+1}\}\} \cup \{\{x_1, x_n\}\}$. A hypergraph is said to be *cycle-free* when no chordless cycle can be found.

An example is provided figure 4.1.

The following result will be used as a starting point to prove every result on hypergraphs; some results may therefore appear as consequences of this result, while a proof from scratch would rather use these results to prove the following one.

▶ **Theorem 10 (standard result)**

The following statements are equivalent for any hypergraph $\mathcal{H}$:

- $\mathcal{H}$ is acyclic (i.e. has a join tree).
- $\mathrm{r}^\star(\mathcal{H}) = \emptyset$
- $\mathcal{H}$ is conformal and cycle-free. ◀

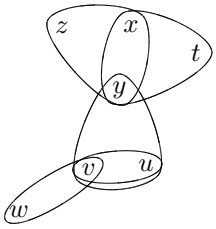
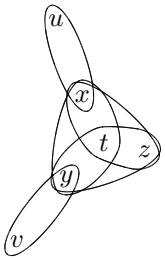
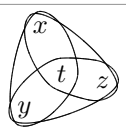
Note that the equivalence of the first two is easy to prove, and that equivalence of the second and the third will be proved in chapter 7.

An example is provided figure 4.2.

First of all, we give a simple lemma.

▶ ***Lemma 17 (cleaning)*** Let t be a join tree t with labeling l. For any vertex x of t such that there exists a vertex $y \neq x$ of t such that $l(x) \subseteq l(y)$, we can manage to build (in linear time) a join tree without the vertex x.

Proof. The vertex x has one neighbour leading to y that we call z. By join property, $l(x) \subseteq l(z)$. Every edge $\{x, u\}$ with $u \neq z$ may be replaced by $\{z, u\}$, the tree would remain a join tree. Now x is a leaf, and can be safely removed. ◀

Hypergraph	$\{\{x,y,z\},\{x,y,t\},\{y,u,v\},$ $\{u,v\},\{v,w\}\}$ 	$\{\{x,y,t\},\{y,z,t\},$ $\{z,x,t\},\{x,u\},\{y,v\}\}$ 
Join Tree	$\{x,t,y\}$ $\{v,u\}$ $\{z,x,y\} - \{y,v,u\} - \{v,w\}$	none
Remainder	$\emptyset$	
Elimination	w, $\{v\}$, $\{v,u\}$, z, t, x, $\{y\}$, u, v	$u, \{x\}, v, \{y\}$
Conformity and Cycles	conformal and cycle-free	(x,y,z) is a chordless cycle

● **Figure 4.2:** The left one is acyclic while the right one isn't.

This lemma can be used to prove that from any join tree t, we can build another join tree t' having the same set of images except $\emptyset$, with an injective labeling function.

► ***Lemma 18*** Acyclicity is minor-closed.

Proof. We have three points to prove: acyclicity is closed by induction, minimization, and edge contraction. In each, assume $\mathcal{H}$ is acyclic, and t is a join tree of $\mathcal{H}$, with labeling l.

Take S a subset of $\mathcal{V}(\mathcal{H})$. Then the tree t with labeling $l' : x \mapsto l(x) \cap S$ is a join tree of $\mathcal{H}[S]$ except for some vertices mapping to empty set, and several vertices mapping to the same set. This is not a problem due to Lemma 17.

The same lemma can be used to build a join tree of $\mathrm{r_e}(\mathcal{H})$.

Assume we have two vertices a and b such that some edge of $\mathcal{H}$ contains them both. Then, by the join property, the set of vertices v such that $a \in l(v)$ or $b \in l(v)$ is a subtree of t. Therefore, the modified labeling:

$$l' = x \mapsto \begin{cases} l(x) & \text{if } b \notin l(x) \\ l(x) \setminus \{b\} \cup \{a\} & \text{if } b \in l(x) \end{cases}$$

preserves the join property, and (t, l') is a join tree of the hypergraph obtained by the edge contraction, once again up to some duplicate images, which is solved by Lemma 17. ◄

We now re-write the standard result (Theorem 10) as a slightly different triple definition of acyclicity: one statement being used as its definition, and allowing easy reasoning on acyclicity in the next sections, one being a "positive" statement used to prove the easiness part of the dichotomy theorem, and the last one being a "negative" statement used to prove the hardness part of the dichotomy. Most hypergraph theorems we will give will have a similar structure.

The negative statement is about the presence of certain induced hypergraphs; we introduce them now.

Definition 31 (Tetra– and Cycle–graphs) We define, for any $k \geq 3$:

$$\text{Cycle}(k) = \{\{i, i+1\} \mid 1 \leq i < k\} \cup \{\{1, k\}\}$$
$$\text{Tetra}(k) = \{\{1,...,k\} \setminus \{x\} \mid x \in \{1,...,k\}\}$$

Hypergraphs of the first kind may also be called *cycle-graphs* and those of the second kind *tetra-graphs*.

In particular, $\text{Tetra}(3) = \{\{1,2\},\{2,3\},\{3,1\}\} = \text{Cycle}(3)$.

Remark 24 The name "Tetra" comes from "tetrahedron": the hypergraph Tetra(3) describes a triangle (a set of three edges, where two edges always share one vertex) and the hypergraph Tetra(4) describes a regular tetrahedron (a set of four faces, where two faces always share an edge). This can be generalized to spaces on k dimensions.

▶ **Theorem 11 (acyclicity)**

The following statements are equivalent:

- **(def)** $\mathcal{H}$ is acyclic (i.e. has a join tree or, equivalently, is conformal and cycle-free)
- **(+)** $\text{r}^\star(\mathcal{H}) = \emptyset$ (or, equivalently, $\mathcal{H}$ admits some elimination order)
- **(–)** We cannot find S such that $\text{r}^\star_\text{e}(\mathcal{H}[S])$ is either isomorphic to $\text{Cycle}(\text{Card}\,(S))$ or to $\text{Tetra}(\text{Card}\,(S))$.
- **(–)** $\mathcal{H}$ does not admit any $\text{Tetra}(k)$, $k \geq 3$, as a pseudo-minor.

An example is provided at figure 4.3.

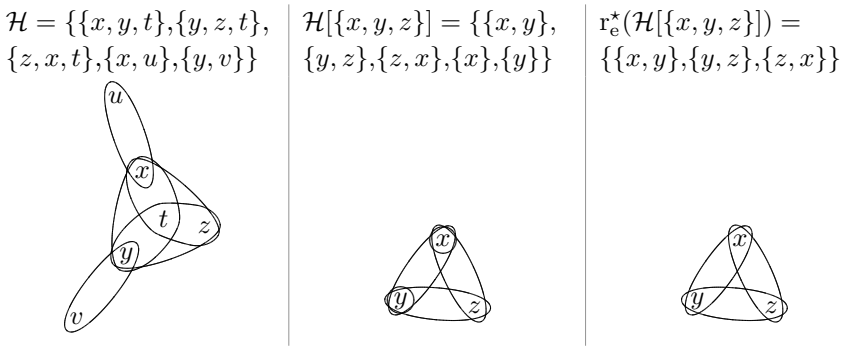
Proof. We have to prove the equivalence between "$\mathcal{H}$ is conformal and cycle-free" and the first negative formulation. Equivalently, by Theorem 10, we have to prove a hypergraph is non-conformal or has a chordless cycle if and only if there is a set S of vertices such that the hypergraph $\text{r}^\star_\text{e}(\mathcal{H}[S])$ is *either* a cycle-graph of order ≥ 3 *or* a tetra-graph of order > 3.

The "if" part is obvious: if there is a set S of vertices such that the hypergraph $\text{r}^\star_\text{e}(\mathcal{H}[S])$ is a cycle-graph then $\mathcal{H}$ cannot be cycle-free hence not acyclic by Theorem 10; and if it is a tetra-graph of order > 3, then $\mathcal{H}$ is non-conformal hence not acyclic.

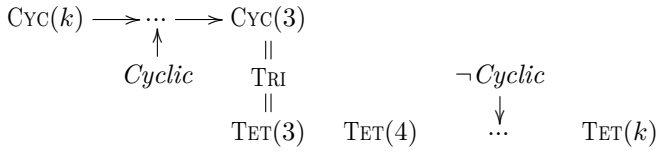
We prove the "only if" part. This can be done in two points: first of all prove that if there is a cycle, then for some S, $\text{r}^\star_\text{e}(\mathcal{H}[S])$ is $\text{Cycle}(\text{Card}\,(S))$, and then, as a second point, prove that if a cycle-free hypergraph $\mathcal{H}$ is non-conformal, then we can find S such that $\text{r}^\star_\text{e}(\mathcal{H}[S]) = \text{Tetra}(\text{Card}\,(S))$.

First point: if there is a chordless cycle $(x_1,...,x_n)$ in $\mathcal{H}$, then $\mathcal{H}[\{x_1,...,x_n\}]$ is a cycle with additional singleton edges, therefore the hypergraph $\text{r}^\star_\text{e}(\mathcal{H}[\{x_1,...,x_n\}])$ is isomorphic to $\text{Cycle}(n)$, which is the first point.

Second point: assume $\mathcal{H}$ is a cycle-free and non-conformal hypergraph. Take K the smallest non-conformal clique of $\mathcal{H}$, that is to say it is not included in an edge of $\mathcal{H}$. Assume we can find $x \in K$ such that $K \setminus \{x\}$ is not an edge of $\mathcal{H}[K]$. Then $K \setminus \{x\}$ is a non-conformal clique smaller than K, contradiction. Therefore $\{K \setminus \{x\} \mid x \in K\} \subseteq \mathcal{H}[K]$. Since every other edge of $\mathcal{H}[K]$ is contained in some edge $K \setminus \{x\}$, we have $\text{r}_\text{e}(\mathcal{H}[K]) = \text{Tetra}(\text{Card}\,(K))$. By definition, K contains at least three elements, but assuming it contains three elements exactly contradicts absence of cycles in $\mathcal{H}$.



● **Figure 4.3:** Illustration of Theorem 11.



● **Figure 4.4:** Illustration of Theorem 12. A non-acyclic query is either cyclic or not. The first one — *Cyclic* in the figure — expresses some $\mathcal{Q}(\mathrm{Cycle}(i))$ with $i \geq 3$, and therefore $\mathcal{Q}(\mathrm{Tetra}(3))$; the second one — $\neg$*Cyclic* — some $\mathcal{Q}(\mathrm{Tetra}(i))$ with $i \geq 4$. Cyc(k) is a shorthand for $\mathcal{Q}(\mathrm{Cycle}(k))$. Arrows mean $\succcurlyeq$ and double arcs means $\eqsim$. Notice that horizontal arrows mean reduction by edge contraction, while the other ones mean reduction by induction and/or minimization.

Obviously, if $\mathcal{H}$ does not contain any $\mathrm{Tetra}(k)$, $k \geq 3$, as a minor, in particular, we cannot find S such that $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[S])$ is either isomorphic to $\mathrm{Cycle}(\mathrm{Card}\,(S))$ — since it would admit Tetra(3) as a minor — or to $\mathrm{Tetra}(\mathrm{Card}\,(S))$. The converse is a direct consequence of Lemma 18. ◀

3.1.2 Consequences on Quantifier-Free Conjunctive Queries

Recall that, in the rest of the document, relation symbols are assumed to always occur *once* in formulas.

First of all, we can show how to decide an acyclic ECQ in linear time. We will establish a linear-time reduction from the decision problem $?\mathcal{Q}(\phi)$ to the decision problem $?\mathcal{Q}(\mathrm{r_v}(\phi))$; and also from the decision problem $?\mathcal{Q}(\phi)$ to the decision problem $?\mathcal{Q}(\mathrm{r_e}(\phi))$; finally, we notice $?\mathcal{Q}(R())$ is trivially decidable in linear time by some algorithm A_0. When ϕ is acyclic, this will give a linear-time algorithm deciding $?\mathcal{Q}(\phi)$:

$?\mathcal{Q}(\phi)$:
- **if** ϕ only contains $R()$ (i.e. has no variable left):
  - **return** A_0
- **elsif** ϕ contains something like $R(\bar{x},\bar{y}) \wedge S(\bar{x})$:
  - Reduce to $?\mathcal{Q}(\mathrm{r_e}(\phi))$
- **elsif** ϕ contains $R(\bar{y},x)$ with x isolated:
  - Reduce to $?\mathcal{Q}(\mathrm{r_v}(\phi))$
- **else** ϕ is not acyclic !!!

Algorithm 4.1 shows how to proceed to the reductions, and it is easy to see they are in Lin, i.e. performed in linear time. The next lemma is a generalisation of this.

```
?Q(φ)(S):
    if φ contains just R():
        if R^S ≠ ∅: return True
        else return False
    elsif φ contains something like R(x̄, ȳ)∧S(x̄):
        R^S ← {(ā, b̄) ∈ R^S | ā ∈ S^S}
        Remove S(X̄) from φ
        return ?Q(φ)(S)
    elsif φ contains R(ȳ, x) with x isolated:
        R^S ← {(ā) | (ā, b) ∈ R^S}
        Replace R(ȳ, x) by R(ȳ) in φ
        return ?Q(φ)(S)
```

● **Algorithm 4.1:** Decision of acyclic existential conjunctive queries.

► ***Lemma 19*** Let ϕ be an acyclic QFCQ. Then $\mathcal{Q}(\phi)$ and $\#\mathcal{Q}(\phi)$ therefore $?\mathcal{Q}(\phi)$ are optimal (ConstDoLin, resp. Lin) and $\star\mathcal{Q}(\phi)$ is easy (LogDoLin).

Proof. Algorithm 4.2 shows how to count and enumerate the solutions of an acyclic QFCQ, in linear time in the first case, at constant delay after linear precomputation in the second one. The correctness of both algorithms is justified by Theorem 11; it is basically the same idea as in the decision case. The reader, if not familiar with this kind of "functional-flavoured imperative programming style", is invited to consider the execution on a very simple example, this alone should be enough to make things clear.

In the case of the counting algorithm, we deal with a slightly more general problem : the sum of weighted solutions. The idea is to associate, to every tuple $\bar{a}$ belonging to some relation $R^{\mathcal{S}}$ a *weight* noted $w(\bar{a}) = f_R^{\mathcal{S}}(\bar{a})$, which is initially set to 1. The weight $w(\bar{a})$ reflects the number of "continuations" of a given tuple $\bar{a}$ in the relations that were previously removed. It is easy to see how the storage of this function can be efficiently implemented: we can implement each relation R as a list of ordered pairs $(\bar{a}, w(\bar{a}))$ for $\bar{a} \in R^{\mathcal{S}}$. This way, we can consider simultaneously every tuple $\bar{a} \in R^{\mathcal{S}}$ together with its image by $w(\bar{a}) = f_R^{\mathcal{S}}(\bar{a})$. This is easy to maintain.

In the case of the enumeration algorithm, $f_R^{\mathcal{S}}$ is now the set of continuations, but the way things work is the same as for counting. The main new feature of this algorithm is that, instead of returning the calculated value, we proceed to enumeration of ordered pairs (m, f), where each m is a partial model, and f is a function that associates to some relation symbol R, the value of $f_R^{\mathcal{S}}(m')$ where m' is the partial model m projected onto variables that R holds (i.e. the atom holding R).

Finally, algorithm 4.3 shows the easiness of the Jeth problem. Here L is the list of respective ranks of desired solutions, that we expect no to be read until the precomputation is done, then, we want the delay before the reading of an element of L and the production of the corresponding solution to be $\mathcal{O}(\log|\mathcal{S}|)$. We chose to *ignore silently (in constant time)* ranks that are greater than the count of the query.

In this algorithm, the f_R functions correspond to those of counting, while g_R functions correspond to f_R functions of enumeration with the difference that, instead of saving the set of continuations, it saves it together with a cumulative count. ◄

$\#\mathcal{Q}(\phi)(\mathcal{S})$:
- **if** ϕ only contains $R()$:
  - **if** $R^{\mathcal{S}} \neq \emptyset$:
    - **return** $f_R^{\mathcal{S}}(())$
  - **return** 0
- **elsif** ϕ contains something like $R(\bar{x},\bar{y}) \wedge S(\bar{x})$:
  - $f_R^{\mathcal{S}} \leftarrow (\bar{a},\bar{b}) \mapsto f_R^{\mathcal{S}}(\bar{a},\bar{b}) \times f_S^{\mathcal{S}}(\bar{a})$
  - $R^{\mathcal{S}} \leftarrow \left\{(\bar{a},\bar{b}) \in R^{\mathcal{S}} \;\middle|\; \bar{a} \in S^{\mathcal{S}}\right\}$
  - Remove $S(\bar{x})$ from ϕ
  - **return** $\#\mathcal{Q}(\phi)(\mathcal{S})$
- **elsif** ϕ contains $R(\bar{y},x)$ with x isolated:
  - $f_R^{\mathcal{S}} \leftarrow (\bar{a}) \mapsto \sum_{(\bar{a},b)\in R^{\mathcal{S}}} f_R^{\mathcal{S}}(\bar{a},b)$
  - $R^{\mathcal{S}} \leftarrow \left\{\bar{a} \;\middle|\; \exists b\, (\bar{a},b) \in R^{\mathcal{S}}\right\}$
  - Replace $R(\bar{y},x)$ by $R(\bar{y})$ in ϕ
  - **return** $\#\mathcal{Q}(\phi)(\mathcal{S})$

$\mathcal{Q}(\phi)(\mathcal{S})$:
- **if** ϕ only contains $R()$:
  - **if** $R^{\mathcal{S}} \neq \emptyset$:
    - **yield** $\left((), R \mapsto f_R^{\mathcal{S}}(())\right)$
- **elsif** ϕ contains something like $R(\bar{x},\bar{y}) \wedge S(\bar{x})$:
  - $f_R^{\mathcal{S}} \leftarrow (\bar{a},\bar{b}) \mapsto \left(f_R^{\mathcal{S}}(\bar{a},\bar{b}), f_S^{\mathcal{S}}(\bar{a})\right)$
  - $R^{\mathcal{S}} \leftarrow \left\{(\bar{a},\bar{b}) \in R^{\mathcal{S}} \;\middle|\; \bar{a} \in S^{\mathcal{S}}\right\}$
  - Remove $S(\bar{x})$ from ϕ
  - **for** $(m,f) \in \mathcal{Q}(\phi)(\mathcal{S})$:
    - $(f(R), f(S)) \leftarrow f(R)$
    - **yield** (m,f)
- **elsif** ϕ contains $R(\bar{y},x)$ with x isolated:
  - $f_R^{\mathcal{S}} \leftarrow (\bar{a}) \mapsto \left\{\left(b, f_R^{\mathcal{S}}(\bar{a},b)\right) \;\middle|\; (\bar{a},b) \in R^{\mathcal{S}}\right\}$
  - $R^{\mathcal{S}} \leftarrow \left\{\bar{a} \;\middle|\; \exists b\, (\bar{a},b) \in R^{\mathcal{S}}\right\}$
  - Replace $R(\bar{y},x)$ by $R(\bar{y})$ in ϕ
  - **for** $(m,f) \in \mathcal{Q}(\phi)(\mathcal{S})$:
    - **for** $(b,f') \in f(R)$:
      - $f(R) \leftarrow f'$
      - **yield** $((m,b),f)$

● **Algorithm 4.2:** Count and enumeration for acyclic quantifier-free conjunctive queries.

Definition 32 (Tetra problem) We call $\textsc{Tet}(k)$ the problem of, given a hypergraph $\mathcal{H}$, producing the set:

$$\{E \subseteq \mathcal{V}(\mathcal{H}) \,|\, \forall x \in E\ E \smallsetminus \{x\} \in \mathcal{H}\}$$

In particular, we call $\textsc{Tri} = \textsc{Tet}(3)$, which is the triangle problem.

▶ ***Lemma 20*** $\textsc{Tet}(k) \asymp \mathcal{Q}(\text{Tetra}(k))$.

Proof. Analog to the one of Example 14. ◀

▶ **Theorem 12 (QFCQ conditional dichotomies)**

Under and only under the complexity hypothesis that for every $k \geq 3$, $\textsc{Tet}(k)$ (resp.$?\textsc{Tet}(k)$, $\#\textsc{Tet}(k)$, $\star\textsc{Tet}(k)$) is not easy (resp. optimal, tractable), a quantifier-free conjunctive query enumeration (resp. decision, count, Jeth) problem is easy (resp. optimal, tractable) only if the hypergraph of the query is acyclic.

As a corollary, (and by previous lemma) under the reasonable hypothesis that deciding $?\textsc{Tet}(k)$ is not tractable for any $k \geq 3$, the following are equivalent for any QFCQ ϕ:

$\star\mathcal{Q}(\phi)(\mathcal{S})(L)$:
- **if** ϕ only contains $R()$:
  - **if** $R^{\mathcal{S}} \neq \emptyset$:
    - **for** $i \in L$:
      - **if** $i < f_R^{\mathcal{S}}(())$:
        - **yield** $\left((), R \mapsto \left(i, g_R^{\mathcal{S}}(())\right)\right)$
- **elsif** ϕ contains something like $R(\bar{x}, \bar{y}) \wedge S(\bar{x})$:
  - $f_R^{\mathcal{S}} \leftarrow (\bar{a}, \bar{b}) \mapsto f_R^{\mathcal{S}}(\bar{a}, \bar{b}) \times f_S^{\mathcal{S}}(\bar{a})$
  - $g_R^{\mathcal{S}} \leftarrow (\bar{a}, \bar{b}) \mapsto \left(g_R^{\mathcal{S}}(\bar{a}, \bar{b}), g_S^{\mathcal{S}}(\bar{a}), f_S^{\mathcal{S}}(\bar{a})\right)$
  - $R^{\mathcal{S}} \leftarrow \left\{(\bar{a}, \bar{b}) \in R^{\mathcal{S}} \;\middle|\; \bar{a} \in S^{\mathcal{S}}\right\}$
  - Remove $S(\bar{x})$ from ϕ
  - **for** $(m, f) \in \star\mathcal{Q}(\phi)(\mathcal{S})(L)$:
    - $\left(n, (g_R, g_S, f_S)\right) \leftarrow f(R)$
    - $f(S) \leftarrow \left((n \mod f_S), g_S\right)$
    - $f(R) \leftarrow \left(\left\lfloor \frac{n}{f_S} \right\rfloor, g_R\right)$
    - **yield** (m, f)
- **elsif** ϕ contains $R(\bar{y}, x)$ with x isolated:
  - $f_R^{\mathcal{S}} \leftarrow (\bar{a}) \mapsto \sum_{\bar{a}, b \in R^{\mathcal{S}}} f_R^{\mathcal{S}}(\bar{a}, b)$
  - $g_R^{\mathcal{S}} \leftarrow (\bar{a}) \mapsto \left\{\left(b, g_R^{\mathcal{S}}(\bar{a}, b), \sum_{\bar{a}c < \bar{a}b} f_R^{\mathcal{S}}(\bar{a}, c)\right) \;\middle|\; (\bar{a}, b) \in R^{\mathcal{S}}\right\}$
  - $R^{\mathcal{S}} \leftarrow \left\{\bar{a} \;\middle|\; \exists b\, (\bar{a}, b) \in R^{\mathcal{S}}\right\}$
  - Replace $R(\bar{y}, x)$ by $R(\bar{y})$ in ϕ
  - **for** $(m, f) \in \star\mathcal{Q}(\phi)(\mathcal{S})(L)$:
    - $(n, g) \leftarrow f(R)$
    - Look for $(b, g_R, w) \in g$ such that $w \leq n$ with w maximal
    - $f(R) \leftarrow (n - w, g_R)$
    - **yield** $((m, b), f)$

● **Algorithm 4.3:** Jeth for acyclic quantifier-free conjunctive queries.

- ϕ is acyclic
- $\mathcal{Q}(\phi)$ is optimal/easy/tractable
- $?\mathcal{Q}(\phi)$ is optimal/easy/tractable
- $\#\mathcal{Q}(\phi)$ is optimal/easy/tractable
- $\star\mathcal{Q}(\phi)$ is easy/tractable

We call easy such a query.

Proof. By Theorem 11, Lemma 19, and Lemma 16. ◀

3.2 Existential Conjunctive Queries and Acyclicity-based Properties

We now introduce acyclicity-based properties allowing to give an analog of the previous dichotomy theorem but in the more general case of *existential* conjunctive queries (ECQ), then to refine this result with the concern of enumerating in *some* lexicographical order, and finally introduce a class we call *trivial* that is shown to be the only one allowing enumeration in *every* lexicographical order, and getting the Jeth element in constant time (versus logarithmic time in the easy case). As often as possible, we will give extensions of easiness results to the general (First-Order) conjunctive queries.

The presented properties are defined once again as three equivalent statements on hypergraphs: one is simple, used for defining and checking the property in reasonable time, another one is based on the existence of some variable elimination order, and is used to show how such a

property allows for answering certain queries in ConstDoLin, and the last is defined as the absence of a certain induced pattern, that would express something "hard".

3.2.1 Starring Property

The following is a simple lemma, which allows a much simpler proof of Bagan's result [BDG07].

▶ ***Lemma 21 (glue)*** Let $\mathcal{H}$ be a hypergraph; if both hypergraphs $\mathcal{H}[S]$ and $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$ are acyclic, then we can build a join tree of $\mathcal{H} \cup \mathcal{H}[S]$ with a join tree of $\mathcal{H}[S]$ as a subtree.

Proof. By Theorem 10, we have t_1 and t_2 two join trees resp. of $\mathcal{H}[S]$ and $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$. Consider the neighbours x_i of the vertex labeled S in t_2. Remove the vertex labeled S, we get a forest, and for any $i \neq j$, $l(x_i) \cap l(x_j) \subseteq S$. Each x_i can be connected to the vertex labeled $l(x_i) \cap S$ in t_1. We have $l(x_i) \cap (l(x_i) \cap S) = l(x_i) \cap S$, the join property is therefore preserved. ◀

Definition 33 (starred hypergraph) We say that a hypergraph $\mathcal{H}$ is *starred with respect to* S, or *S-starred*, with $S \subseteq \mathcal{V}(\mathcal{H})$ if both $\mathcal{H}$ and $\mathcal{H} \cup \{S\}$ are acyclic.

Example 25 $\{\{x, y, v\}, \{t, u, v\}, \{y, z\}\}$ is $\{x, t, v\}$-starred but not $\{x, z\}$-starred.

Remark 26 With this definition, it should be clear that, if some hypergraph $\mathcal{H}$ is S-starred, then $\mathcal{H}[S']$ is $S \cap S'$-starred, and that $\mathcal{H}$ is S-starred if and only if $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H})$ is.

We now give an alternative characterisation of the starring property.

▶ ***Lemma 22*** The following statements are equivalent:
- Both hypergraphs $\mathcal{H}$ and $\mathcal{H} \cup \{S\}$ are acyclic.
- Both hypergraphs $\mathcal{H}[S]$ and $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$ are acyclic.

Proof. By Lemma 21, if $\mathcal{H}[S]$ and $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$ are acyclic, then so is $\mathcal{H} \cup \mathcal{H}[S]$; since $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}) = \mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H} \cup \mathcal{H}[S])$, $\mathcal{H}$ is then also acyclic. By Lemma 18, $\mathcal{H}$ is acyclic implies that so do $\mathcal{H}[S]$. Since $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H} \cup \{S\}) = \mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\})$, $\mathcal{H} \cup \{S\}$ is acyclic iff $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$ is. ◀

▶ **Theorem 13 (S-starred)**

The following are equivalent:

(def) Both $\mathcal{H}$ and $\mathcal{H} \cup \{S\}$ are acyclic.

(+) $\mathcal{H}$ admits an elimination order that eliminates vertices of $\mathcal{V}(\mathcal{H}) \setminus S$ before those of S.

(−) $\mathcal{H}$ is acyclic and $\mathcal{H} \cup \{S\}$ does not admit the triangle as a minor.

Proof (+). In order to prove equivalence between the first two points, by Lemma 22, we only need to prove equivalence between:
- Both $\mathcal{H}[S]$ and $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$ are acyclic.
- $\mathcal{H}$ admits an order of elimination of variables that eliminates vertices of $\mathcal{V}(\mathcal{H}) \setminus S$ before those of S.

We prove the second point implies the first one. Assume $S = \{x_1, ..., x_k\}$, $\mathcal{V}(\mathcal{H}) = \{x_1, ..., x_n\}$, and $(x_1, ..., x_n)$ is an elimination order of $\mathcal{H}$. Then it is easy to check that $(x_1, ..., x_n)$ is an elimination order of $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$, and that $(x_1, ..., x_k)$ is an elimination order of $\mathcal{H}[S]$.

The first one implies by previous lemma that we have a join tree of $\mathcal{H} \cup \mathcal{H}[S]$ in which vertices mapping to subsets of S are a subtree. We show that we can use the tree to get an order of elimination of variables. We proceed by induction on the size (i.e. $|\mathcal{H}| = \sum_{e \in \mathcal{H}} \mathrm{Card}\,(e).$) of $\mathcal{H}$.

Assume this is the case for all hypergraphs of size n or less. Take $\mathcal{H}$ of size $n+1$, $S \subseteq \mathcal{V}(\mathcal{H})$ such that both $\mathcal{H}[S]$ and $\mathcal{H} \setminus \mathcal{H}[S] \cup \{S\}$ are acyclic. If $\mathcal{V}(\mathcal{H}) = S$, the result is obvious; we assume $\mathcal{V}(\mathcal{H}) \neq S$. If $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}) \neq \mathcal{H}$, then the induction hypothesis concludes. We assume $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}) = \mathcal{H}$.

By Lemma 21, $\mathcal{H} \cup \mathcal{H}[S]$ has a join tree which contains a join tree of $\mathcal{H}[S]$. Therefore, the join tree of $\mathcal{H} \cup \mathcal{H}[S]$ contains a leaf a such that $l(a) \setminus S \neq \emptyset$. Since $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}) = \mathcal{H}$, the neighbour b of a is not such that $l(a) \subseteq l(b)$. Therefore $l(a) \setminus l(b)$ is not empty. Assuming $l(a) \setminus l(b) \subseteq S$ contradicts the join property. We therefore have a vertex $x \in l(a) \setminus l(b)$ that is not in S. By join property, x is isolated in $\mathcal{H}$. By the induction hypothesis, we have some elimination order $(x_1, ..., x_n)$ of $\mathcal{H}[\setminus\{x\}]$ that eliminates vertices of $\mathcal{V}(\mathcal{H}[\setminus\{x\}]) \setminus S$ before those of S. Therefore $(x_1, ..., x_n, x)$ is the desired elimination order of $\mathcal{H}$.

Proof (–). Assume $\mathcal{H}$ is acyclic. We want to prove equivalence between:

- $\mathcal{H} \cup \{S\}$ is acyclic.
- $\mathcal{H} \cup \{S\}$ does not admit the triangle as a minor.

By Theorem 10, we already know the first implies the second. Assume $\mathcal{H} \cup \{S\}$ is not acyclic. We know there must be some S' such that the hypergraph $\mathcal{H}' = \mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H} \cup \{S\}[S']) = \mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[S] \cup \{S \cap S'\})$ is isomorphic to:

- either $\{\{1, ..., k\} \setminus \{x\} \mid x \in \{1, ..., k\}\}$ with $k > 3$,
- or $\mathrm{Cycle}(k) = \{\{i, i+1\} \mid 1 \leq i < k\} \cup \{\{1, k\}\}$ with $k > 3$.

If the edge $S \cap S'$ does not belong to $\mathcal{H}'$, then $\mathcal{H}' = \mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[S'])$, therefore $\mathcal{H}$ is not acyclic, contradiction. Therefore $S \cap S' \in \mathcal{H}'$. Assume $\mathcal{H}'$ is in the first form. Then $(\mathcal{H}' \setminus \{S \cap S'\})[S]$ is also in the same [9] form, with $k > 2$. Hence this latter hypergraph is not acyclic, therefore $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[S' \cap S])$ is not acyclic, so $\mathcal{H}$ is not acyclic, contradiction. Then $\mathcal{H}'$ is a cycle, which proves that $\mathcal{H} \cup \{S\}$ admits the triangle as a minor. ◀

▶ ***Lemma 23*** Let $\phi = (x_1, ..., x_k) \mid \exists x_{k+1} ... \exists x_n \ \psi(x_1, ..., x_n)$ be an existential conjunctive query, with ψ its quantifier-free part. If the hypergraph $\mathcal{H}(\phi) \cup \{\{x_1, ..., x_k\}\}$ is acyclic, then $\mathcal{Q}(\phi[\{x_1, ..., x_k\}]) \approx \mathcal{Q}(\phi)$.

Proof. We proceed by induction on the number of quantifiers $n - k$. If $\mathcal{H}(\phi) \cup \{\{x_1, ..., x_k\}\}$ is acyclic, then, obviously, it is $\{x_1, ..., x_k\}$-starred, therefore there is an elimination order that eliminates every x_i with $i > k$ before the x_i with $i \leq k$. In particular, there is some x_i that is isolated in $\mathrm{r}_{\mathrm{e}}(\mathcal{H}(\phi))$. By Lemma 14, we have both:

- $\mathcal{Q}(\phi) \approx \mathcal{Q}(\mathrm{r}_{\mathrm{e}}(\phi))$
- $\mathcal{Q}(\mathrm{r}_{\mathrm{e}}(\mathrm{r}_{\mathrm{e}}(\phi)[\setminus\{x_i\}])) = \mathcal{Q}(\mathrm{r}_{\mathrm{e}}(\phi[\setminus\{x_i\}])) \approx \mathcal{Q}(\phi[\setminus\{x_i\}])$

We therefore just have to prove that, if x_i is isolated in ϕ, then $\mathcal{Q}(\phi) \approx \mathcal{Q}(\phi[\setminus\{x_i\}])$. The right-left reduction is Lemma 13. We prove the reduction in the other way. We have:

$$\phi = (x_1, ..., x_k) \mid \exists x_{k+1} ... \exists x_{i-1} \exists x_{i+1} ... \exists x_m \underbrace{\exists x_i \ \psi(x_1, ..., x_n)}_{\psi'(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n)}$$

We can put ψ' in the following form:

$$\psi'(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) = \psi''(x_1, ..., x_{i-1}, x_{i+1}, ..., x_n) \wedge \exists x_i R(, ..., x_i, ...)$$

9 Take $\mathcal{H}_1 = \{K_1 \setminus \{x\} \mid x \in K_1\}$. Choose $e = K_1 \setminus \{y\} \in \mathcal{H}_1$. Consider $\mathcal{H}_2 = \mathcal{H}_1 \setminus \{e\}[e]$. We have $\mathcal{H}_2 = \{K_1 \setminus \{y\} \setminus \{x\} \mid x \in K_1 \setminus \{y\}\}$. If we define $K_2 = K_1 \setminus \{y\}$, we have $\mathcal{H}_2 = \{K_2 \setminus \{x\} \mid x \in K_2\}$.

where R is the only relation holding x_i, say in $j$eth position, for a total of k variables. Therefore, defining:

$$R^{\mathcal{S}_2} = \{(a_1,...,a_{j-1},a_{j+1},...,a_k) \mid (a_1,...,a_k) \in R^{\mathcal{S}_1}\}$$

proves $\mathcal{Q}(\phi) \preccurlyeq \mathcal{Q}(\phi[\diagdown\{x_i\}])$. Finally, the hypergraph $(\mathcal{H}\cup\{x_1,...,x_k\})[\diagdown\{x_i\}]$ which is the same as $\mathcal{H}[\diagdown\{x_i\}] \cup \{x_1,...,x_k\}$ is also acyclic, hence, by induction hypothesis, $\mathcal{Q}(\phi[\diagdown\{x_i\}]) \eqsim \mathcal{Q}(\phi[\diagdown\{x_i\}][\{x_1,...,x_k\}])$. This last query is exactly $\mathcal{Q}(\phi[\{x_1,...,x_k\}])$. ◀

Definition 34 (decision hardness hypotheses) We call the *triangle hypothesis* the complexity hypothesis that checking the presence of a triangle in a graph with n vertices cannot be done in $\mathcal{O}(n^2)$. We call the *tetrahedron hypothesis* the (non-standard) complexity hypothesis that, for all $k > 3$, $?\mathcal{Q}(\text{Tetra}(k))$ cannot be decided in linear time.

We now define a notion of starred query that is equivalent to Bagan's notion of free-connex query, that were defined in a complicated fashion.[10]

Definition 35 (starred query) We say that an ECQ $\mathcal{Q}(\phi)$ with F the set of free variables in ϕ is *starred* when $\mathcal{H}(\phi)$ is F-starred.

▶ **Theorem 14 (ECQ dichotomy)**

Under the triangle and the tetrahedron hypotheses, an ECQ is in ConstDoLin if and only if it is starred.

Proof. We strongly rely on Theorem 13.

By previous lemma, if $\phi = (x_1,...,x_k) \mid \exists x_{k+1}...\exists x_n\ \psi(x_1,...,x_n)$, with $\mathcal{H}(\psi)$ $\{x_1,...,x_k\}$-starred, then $\mathcal{Q}(\phi) \preccurlyeq \mathcal{Q}(\phi[\{x_1,...,x_k\}]) \in$ ConstDoLin.

A non-acyclic ECQ expresses by Lemma 16 the $?\mathcal{Q}(\text{Tetra}(k))$ problem, for some $k \geq 3$; if $k > 3$, the tetrahedron hypothesis concludes, else the triangle hypothesis does.

By Lemma 16 again, an acyclic but non starred ECQ expresses the query $\mathcal{Q}\,((x,y)\ \exists z \mid R(x,z)\wedge R(y,z))$ which cannot be done in linear time, by the triangle hypothesis. ◀

3.2.2 Ordering Property

We introduce a notion of ordering, that will be proved equivalent to the notion of elimination order, but which is more convenient.

Definition 36 (ordered hypergraph) We say that a hypergraph $\mathcal{H}$ where $\mathcal{V}(\mathcal{H}) = \{x_1,...,x_n\}$ is $(x_1,...,x_n)$-*ordered* when $\mathcal{H}$ is $\{x_1,...,x_k\}$-starred for all $k \in \{1,...,n\}$.

▶ **Theorem 15 (ordering)**

Let $\mathcal{H}$ be a hypergraph where $\mathcal{V}(\mathcal{H}) = \{x_1,...,x_n\}$. The following statements are equivalent:

- **(def)** $\mathcal{H}$ is $(x_1,...,x_n)$-ordered, i.e. $\{x_1,...,x_k\}$-starred for all k$\in \{1,...,n\}$.
- **(+)** $\mathcal{H}$ admits $(x_1,...,x_n)$ as a reverse elimination order.
- **(−)** $\mathcal{H}$ is acyclic and we can't find $i < j < k$ such that, in $\mathcal{H}$, x_i and x_k are neighbours, x_j and x_k are neighbours, but x_i and x_j are not, i.e. such that $\mathrm{r}^\star_{\mathrm{e}}(\mathcal{H}[\{x_i,x_j,x_k\}]) = \{\{x_i,x_k\},\{x_j,x_k\}\}$.

10 Precisely: a query $\mathcal{Q}(\phi)$ with F the set of free variables in ϕ is *free-connex* when $\mathcal{H}(\phi)$ is F-connex, i.e. admits a join tree t that is F-connex by $\subseteq$-extension, i.e. which is a join tree of $\mathcal{H}(\phi)\cup\mathcal{H}(\phi)[F]$ where the set of all nodes labeled with edges included in F form a connex part (a subtree) of t.

Proof. If $\mathcal{H}$ is not acyclic, all three statements are false. From now on, we assume $\mathcal{H}$ is acyclic. We prove separately the equivalence of the statement based on the definition with the positive statement, and the equivalence of the statement based on the definition with the negative statement.

Proof (+). If $\mathcal{H}$ admits $(x_1,...,x_n)$ as a reverse elimination order, then it admits an elimination order that removes $x_{k+1},...,x_n$ before $x_1,...,x_k$ for all $k < n$. Then, by Theorem 13, it is $\{x_1,...,x_k\}$-starred for every $k \leq n$.

Now assume $\mathcal{H}$ is $\{x_1,...,x_k\}$-starred for all $k \in \{1,...,n\}$, i.e. every $\mathcal{H} \cup \{\{x_1,...,x_k\}\}$ is acyclic. As a consequence, $\mathcal{H}[\{x_1,...,x_{k+1}\}]$ is $\{x_1,...,x_k\}$-starred for all $k \in \{1,...,n-1\}$.

We prove by induction that for all $k < n$, $\mathcal{H}[\{x_1,...,x_{k+1}\}]$ admits $(x_1,...,x_k)$ as an elimination order. If $k = 1$, the result is obvious. Assume this is the case for a given k. The induced hypergraph $\mathcal{H}' = \mathcal{H}[\{x_1,...,x_{k+1}\}]$ is $\{x_1,...,x_k\}$-starred, therefore, by Theorem 13, it admits an elimination order that eliminates $\{x_{k+1}\}$ before $\{x_1,...,x_k\}$, that is to say that eliminates x_{k+1} in first place. Since $\mathcal{H}'[\setminus\{x_{k+1}\}] = \mathcal{H}'[\{x_1,...,x_k\}]$ admits $(x_1,...,x_k)$ as an elimination order by induction hypothesis, finally, by definition of an elimination order, $\mathcal{H}'$ admits $(x_1,...,x_{k+1})$ as an elimination order, which concludes the proof.

Proof (–). Recall that $\mathcal{H}$ is supposed to be acyclic.

We prove that, if $\mathcal{H}$ is acyclic but not $(x_1,...,x_n)$-ordered, then we can find $i < j < k$ such that $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[\{x_i,x_j,x_k\}]) = \{\{x_i,x_k\},\{x_j,x_k\}\}$. Assume $\mathcal{H}$ is not $\{x_1,...,x_k\}$-starred for some k. Then, by Theorem 13, $\mathcal{H} \cup \{\{x_1,...,x_k\}\}$ contains the triangle as a minor. This means we can find $S \subseteq \mathcal{V}(\mathcal{H})$ such that $\mathcal{H}' = \mathrm{r}_{\mathrm{e}}^{\star}((\mathcal{H} \cup \{\{x_1,...,x_k\}\})[S])$ is a chordless cycle. Take $x_m \in S$, such that m is maximal. In $\mathcal{H}'$, x_m has (exactly) two neighbours x_i and x_j, we assume $i < j$. In $\mathcal{H}$, x_i (resp x_j) and x_m are neighbours.[11] Assume some edge of $\mathcal{H}$ contains x_i, x_j and x_m. Then so does some edge of $\mathcal{H}'$, contradiction with the assumption that the cycle is chordless. Assume some edge of $\mathcal{H}$ contains x_i and x_j but not x_m. Then $\mathcal{H}$ contains a triangle, and is therefore not acyclic, contradiction. Therefore, x_i and x_j are not neighbours in $\mathcal{H}$. We have found $i < j < k$ such that $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[\{x_i,x_j,x_k\}]) = \{\{x_i,x_k\},\{x_j,x_k\}\}$.

Now we prove that $\mathcal{H}$ cannot be $(x_1,...,x_n)$-ordered if we can find $i < j < k$ such that $\mathrm{r}_{\mathrm{e}}^{\star}\mathcal{H}[\{x_i,x_j,x_k\}] = \{\{x_i,x_k\},\{x_j,x_k\}\}$. Assume we can find $i < j < k$ such that, in $\mathcal{H}$, x_i and x_k are neighbours, x_j and x_k neighbours, but no edge includes all three. In this case, they are pairwise neighbours in $(\mathcal{H} \cup \{\{x_1,...,x_j\}\})$, but not included all three in the same edge, therefore $\mathcal{H} \cup \{\{x_1,...,x_j\}\}$ is not acyclic, hence $\mathcal{H}$ is not $\{x_1,...,x_j\}$-starred, which concludes. ◀

Definition 37 (order hypothesis) We call *order hypothesis* the statement that the query $\mathcal{Q}((x_1,x_2,x_3) \mid R(x_1,x_3) \wedge R(x_2,x_3))$ cannot be enumerated in ConstDoLin in the lexicographical order (x_1,x_2,x_3).

▶ **Theorem 16 (ordered enumeration dichotomies)**

Under the order hypothesis, A QFCQ is in ConstDoLin for the lexicographical order $(x_1,...,x_n)$ if and only if it is $(x_1,...,x_n)$-ordered.

Under the order hypothesis, the triangle hypothesis, and the tetrahedron hypothesis (see Theorem 14), an existential conjunctive query

11 This is because induction and minimization do not affect neighbourhood (except, of course, for vertices that were lost during induction).

$\mathcal{Q}(\phi)$ is in ConstDoLin for the lexicographical order $(x_1,...,x_n)$ if and only if $\mathcal{H}(\phi)$ is $\{x_1,...,x_n\}$-starred and $\mathcal{H}(\phi)[\{x_1,...,x_n\}]$ is $(x_1,...,x_n)$-ordered.

Proof. We rely on Theorem 15.

A QFCQ that is $(x_1,...,x_n)$-ordered admits this order as a reverse elimination order, therefore our enumeration algorithm can be used to enumerate in the corresponding lexicographical order.

A QFCQ that is not $(x_1,...,x_n)$-ordered expresses parsimoniously the problem of the enumeration in the (x_1, x_2, x_3) lexicographical order of $\mathcal{Q}((x_1, x_2, x_3) \mid R(x_1, x_3) \wedge R(x_2, x_3))$ which is presumably hard.

The second point is a direct consequence of the first one and of Theorem 14. ◀

3.3 The Whole Picture

We have defined two classes of existential conjunctive queries: those which can be decided fast — characterized by their acyclic hypergraph — and a subset of those that can be enumerated at constant delay — characterized by their starring property.

We now see a last class which is the most favourable case: the class of queries that can be enumerated at constant delay in *every* lexicographical order, and also those admitting an optimal algorithm for the Jeth problem. Unfortunately, such queries are equivalent to tautological queries.

3.3.1 The Trivial Class

We are interested in the class of queries that allow to enumerate optimally fast *in every lexicographical order*. We introduce the corresponding hypergraph notion.

Definition 38 (trivial hypergraph) We say that a hypergraph is *trivial* when each connected component is included in an edge.

We characterize this notion in terms of starring property (starred w.r.t. any set), in terms of ordering property (ordered for any permutation) and in terms of excluded minor.

▶ **Theorem 17 (trivial)**

Let $\mathcal{H}$ be a hypergraph with $\mathcal{V}(\mathcal{H}) = \{x_1,...,x_n\}$. The following statements are equivalent:

1. Each connected component of $\mathcal{H}$ is included in an edge ($\mathcal{H}$ is trivial).
2. The hypergraph $\mathcal{H}$ is S-starred for any $S \subseteq \mathcal{V}(\mathcal{H})$.
3. The hypergraph $\mathcal{H}$ is $(x_{\pi(1)},...,x_{\pi(n)})$-ordered for any permutation π.
4. We cannot find three vertices x, y and z such that $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[\{x, y, z\}]) = \{\{x, y\}, \{y, z\}\}$.

Proof. Notice that the statement 4 implies trivially that $\mathcal{H}$ is acyclic. It must be therefore be thought as "4 and $\mathcal{H}$ is acyclic".

The equivalence between 2 and 3 is by definition of ordering; the equivalence between 3 and 4 is a straightforward consequence of Theorem 15; and the equivalence between 4 and 1 is rather obvious. ◀

Definition 39 (trivial query) We call *trivial* a query equivalent to a *tautological query*, i.e. a query in the form $\mathcal{Q}((x_1 \in D_1,...,x_n \in D_n) \mid \top)$.

We establish the equivalence, for QFCQ, between being trivial and having a trivial hypergraph.

▶ ***Lemma 24*** A conjunctive query $\mathcal{Q}(\phi_1)$ where $\phi_1 = (x_1 \in D_1, ..., x_n \in D_n) \mid \top$ cannot express parsimoniously the query $\mathcal{Q}(\phi_2)$ where $\phi_2 = (x, y, z) \mid R_1(x, y) \wedge R_2(y, z)$.

A QFCQ query $\mathcal{Q}(\phi)$ is trivial iff its hypergraph $\mathcal{H}(\phi)$ is trivial.

Proof. Assume $\mathcal{Q}(\phi_1)$ could express parsimoniously $\mathcal{Q}(\phi_2)$. In this case, every structure $\mathcal{S}_2$ can be reduced in linear time to some structure $\mathcal{S}_1$, therefore such that $|\mathcal{S}_1| = \Theta(|\mathcal{S}_2|)$ by linearity of the reduction and $\#\mathcal{Q}(\phi_1)(\mathcal{S}_1) = \#\mathcal{Q}(\phi_2)(\mathcal{S}_2)$ by parsimony of the reduction.

We define a family of structures $\mathcal{S}_2(i)$, $i \in \mathbb{N}$. Let $q(i) = \lfloor \sqrt{i} \rfloor$ and $r(i) = i - q(i)^2$. Notice $r(i) = (\sqrt{i} - \lfloor \sqrt{i} \rfloor)(\sqrt{i} + \lfloor \sqrt{i} \rfloor) = \mathcal{O}(\sqrt{i})$. Let:

$$R_1^{\mathcal{S}_2(i)} = \{(x, 1) \mid x \in \{1, ..., q(i)\}\} \uplus \{(x, 2) \mid x \in \{1, ..., r(i)\}\}$$
$$R_2^{\mathcal{S}_2(i)} = \{(1, z) \mid z \in \{1, ..., q(i)\}\} \uplus \{(2, 1)\}$$

We have $|\mathcal{S}_2(i)| = \Theta(\sqrt{i})$ and $\#\mathcal{Q}(\phi_2)(\mathcal{S}_2(i)) = q(i)^2 + r(i) = i$. In particular, this is true when i is a prime number. In this case, the only possible way for $\#\mathcal{Q}(\phi_1)(\mathcal{S}_1(i)) = \prod_{j \leq n} \mathrm{Card}\left(D_j^{\mathcal{S}_1(i)}\right)$ to be i, is that one $D_j^{\mathcal{S}_1(i)}$ exactly has cardinality i, i.e. $|\mathcal{S}_1(i)|$ is of size $\Theta(i)$, therefore $\Theta(|\mathcal{S}_2(i)|^2)$, contradictory. This proves the first point of the lemma.

Now we prove that a QFCQ query ϕ is trivial iff its hypergraph $\mathcal{H}(\phi)$ is trivial. Assume we have a QFCQ ϕ such that $\mathcal{H} = \mathcal{H}(\phi)$ is not trivial. By Theorem 17, we know we can find x, y, z such that:

$$\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[\{x, y, z\}]) = \{\{x, y\}, \{y, z\}\}$$

Then $\mathcal{Q}(\phi)$ expresses parsimoniously a query in the form of $\mathcal{Q}(\phi_2)$. Therefore, by the first point of the lemma, $\mathcal{Q}(\phi)$ cannot be equivalent to some query in the form $\mathcal{Q}(\phi_1)$, therefore $\mathcal{Q}(\phi)$ cannot be trivial.

Now assume $\mathcal{H}(\phi)$ is trivial with l connected components. This means $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}(\phi))$ is in the form $\{\{x_1^1, ..., x_{k_1}^1\}, ..., \{x_1^l, ..., x_{k_l}^l\}\}$, i.e. has l pairwise disjoint edges. That is to say $\mathcal{Q}(\phi)$ is equivalent to the query:

$$(...) \;\Big|\; \bigwedge_{1 \leq j \leq l} R_j\left(x_1^j, ..., x_{k_j}^j\right)$$

which is the Cartesian product $\prod_{1 \leq j \leq l} \mathcal{Q}((...) \mid R_j(x_1^j, ..., x_{k_j}^j))$ that is equivalent to $\prod_{1 \leq j \leq l} \mathcal{Q}((x_j \in D_j) \mid \top)$ which is the tautological query:

$$(x_1 \in D_1, ..., x_l \in D_l) \mid D_1(x_1) \wedge ... D_l(x_l)$$

So $\mathcal{Q}(\phi)$ is trivial, which achieves the proof of the second point of the lemma.

Assume $\mathcal{H} = \mathcal{H}(\phi)$ is not trivial. Then, by Theorem 17, we can find x, y and z such that $\mathrm{r}_{\mathrm{e}}^{\star}(\mathcal{H}[\{x, y, z\}]) = \{\{x, y\}, \{y, z\}\}$. This implies $\mathcal{Q}(\phi) \succcurlyeq \mathcal{Q}((x, y, z) \mid R_1(x, y) \wedge R_2(y, z))$. This completes the proof. ◀

▶ **Theorem 18 (trivial conjunction)**

Let ψ be a quantifier-free conjunction. Let $\phi = (...) \mid \psi$, i.e. $\mathcal{Q}(\phi)$ is *the* QFCQ having ψ as conjunction. Under the triangle, tetrahedron, and order hypotheses, the following statements are equivalent:

- $\mathcal{Q}(\phi)$ is trivial, i.e. equivalent to a tautological query.
- $\mathcal{Q}(\phi)$ can be enumerated in any lexicographical order in ConstDoLin.
- Every existential conjunctive query having ψ as its quantifier-free part is in ConstDoLin.
- $\star\mathcal{Q}(\phi)$ is in ConstoLin, under hypothesis $\star\mathcal{Q}((x, y, z) \mid R_1(x, y) \wedge R_2(y, z))$ is not in ConstoLin.

Proof. By Theorem 17, Lemma 24, Theorem 14 and Theorem 16, we have equivalence of the first three points. They are all equivalent to the fact that $\mathcal{Q}(\phi)$ cannot express parsimoniously the query $\mathcal{Q}\big((x,y,z) \,\big|\, R_1(x,y) \wedge R_2(y,z)\big)$.

We prove the first implies this last statement. Assume $\star\mathcal{Q}(\phi) \in$ ConstoLin and $\mathcal{Q}(\phi)$ expresses parsimoniously $\mathcal{Q}\big((x,y,z) \,\big|\, R_1(x,y) \wedge R_2(y,z)\big)$. Then we contradict the fact that $\star\mathcal{Q}\big((x,y,z) \,\big|\, R_1(x,y) \wedge R_2(y,z)\big)$ is not in ConstoLin.

Assume $\mathcal{H}(\phi)$ is trivial. Then $\mathcal{Q}(\phi)$ is equivalent to some tautological query. Getting the Jeth solution of a tautological query consists in getting the Jeth element of a Cartesian product:

$$\star\mathcal{Q}\big((x_1 \in D_1, ..., x_n \in D_n) \,|\, \top\big)[i] = \big(D_1[\sigma_1(i)], ..., D_n[\sigma_n(i)]\big)$$
$$\text{with } \sigma_k(i) = \left\lfloor \frac{i}{\prod_{j>k} \text{Card}\,(D_j)} \right\rfloor \mod \text{Card}\,(D_k) \quad 1 \leq k \leq n$$

We can get the Jeth element of $\mathcal{Q}(\phi)$ in the same (constant) time. ◀

This can be thought as "a conjunction whose corresponding QFCQ enumeration can be done easily in any lexicographical order is also a conjunction for which any ECQ is easy; such a conjunction has no expressive power at all". As a corollary, any "interesting" conjunction can be used to build hard ECQ, and QFCQ that have lexicographical orders for which the best possible enumeration algorithm relies on the decision algorithm.

▶ **Theorem 19 (trivial query)**

Let $\mathcal{Q}(\phi)$ be an existential conjunctive query whose set of free variables is $x_1,...,x_n$. $\mathcal{Q}(\phi)$ is trivial if and only if $\mathcal{H}(\phi)$ is $\{x_1,...,x_n\}$-starred and $\phi[\{x_1,...,x_n\}]$ is a trivial quantifier-free conjunctive query.

Proof. If $\mathcal{H}(\phi)$ is $\{x_1,...,x_n\}$-starred, then by Theorem 14, $\mathcal{Q}(\phi) \eqsim \mathcal{Q}(\phi[\{x_1,...,x_n\}])$; if this last query is trivial, then $\mathcal{Q}(\phi)$ is also trivial.

Now assume $\mathcal{Q}(\phi)$ is trivial. Then it can be enumerated at constant delay after linear precomputation, therefore it is $\{x_1,...,x_n\}$-starred. By Theorem 14, $\mathcal{Q}(\phi)$ is therefore equivalent to $\mathcal{Q}(\phi[\{x_1,...,x_n\}])$, which is a trivial QFCQ, which has therefore a trivial hypergraph, therefore $\mathcal{H}(\phi)[\{x_1,...,x_n\}]$ is a trivial hypergraph. ◀

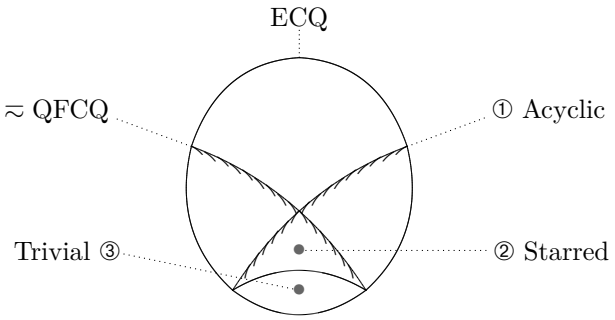
Remark 27 We have seen trivial queries are optimal for *all* mentioned problems: decision, counting, enumeration in any lexicographical order and Jeth in any lexicographical order, while starred ECQ are optimal for decision, counting, enumeration in some lexicographical order, but are only easy for the Jeth problem and tractable for the all-lexicographical order enumeration problem.

3.3.2 The Whole Picture

Now all classes are defined, we can draw the full picture. Before that, a last lemma.

▶ ***Lemma 25*** An acyclic ECQ is tractable for enumeration in *any* lexicographical order.

Proof. By reduction to the decision problem (self-reducibility principle) through Creignou and Hébrard algorithm, see algorithm 2. ◀



	① Acyclic	② Starred	③ Trivial
$\mathcal{H}$, free vars. F	$\mathcal{A}_\alpha(\mathcal{H})$	$\mathcal{A}_\alpha(\mathcal{H}) \wedge$ $\mathcal{A}_\alpha(\mathcal{H} \cup \{F\})$	$\mathcal{A}_\alpha(\mathcal{H}) \wedge$ $\forall E\, \mathcal{A}_\alpha(\mathcal{H}[F] \cup \{E\})$
Decision	Tract., Opt.		
Enumeration	Tractable[a]	Easy, Optimal	
Enum.$\forall <_{\text{lex}}$	Tractable[a]		Easy, Optimal
Counting		Tract.[b] , Opt.	
Jeth		Tract.[c] , Easy	Optimal

a By Lemma 25.
b (Not so reasonable) If counting ones in boolean matrix product can't be qLin.
c (Weaker) If getting the Jeth one in boolean matrix product can't be qLin.

● **Figure 4.5:** The whole picture of ECQ problems complexity.

We could prove enumeration in any lexicographical order can be done at linear delay (instead of quasi-linear) but it would require a (very slightly) more complicated proof, that can be found in [Bag09].

We finally have a simple classification (under complexity hypotheses) of existential conjunctive queries *up to equivalence*. That is why, on figure 4.5, the class ≃QFCQ is the set of ECQ that are equivalent to some QFCQ. On this figure, we can see three classes of growing tractability.

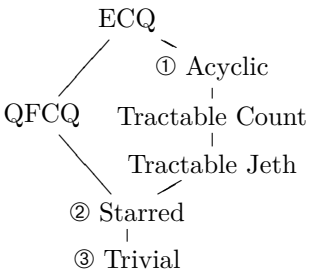
The largest one is *acyclic* ECQ, which can be characterised by one of the following equivalent statements: ECQ whose hypergraph is acyclic, ECQ that are *tractable* (or, at choice, *easy*) for decision, ECQ that are *easy* for decision and ECQ that are tractable for enumeration in *some* (or, at choice, in *all*) lexicographical order. This is summed up by the column "Acyclic" in the figure.

The second class is *starred* queries, characterised by each of the following: queries that are both acyclic and in the equivalence class of some QFCQ, ECQ whose hypergraph is starred w.r.t. the set of free variables, ECQ that are *easy* (or, at choice, *optimal*) for enumeration in some lexicographical order, ECQ that are *tractable* (or, at choice, *easy* or *optimal*) for counting, ECQ that are *tractable* (or, at choice, *easy*) for the Jeth problem. This is summed up by the column "Starred" in the figure.

Finally, the third class is *trivial* queries, that are ECQ that are in the equivalence class of a QFCQ whose hypergraph is trivial. They can also be characterised, among others, by the following equivalent properties: ECQ that are *easy* (or, at choice, *optimal*) for enumeration in *all* lexicographical order, and ECQ that are *optimal* for the Jeth problem.

Now we make some comments on this picture. Distinguishing QFCQ and ECQ looks relevant for decision, but decision ignores the quantified aspect. The only difference is that some ECQ that are tractable for enumeration are not easy. We can merely consider only QFCQ. In that case, tractable and easy are surprisingly synonyms for every problem. Easy and optimal also are, which is less surprising, *except* for the Jeth problem. This exception together with the problem of enumerating in any lexicographic order defines a subclass of queries optimal for every problem, that seriously lacks expressive power, since they are equivalent to queries of the tautological formula.

In the (pessimistic) case where the two complexity hypotheses (b) and (c) mentioned in figure 4.5 fail, the picture becomes:



3.3.3 Conjunctive Queries

A natural question is the following: can the classification results obtained for existential conjunctive queries be generalized to all conjunctive queries, i.e. for any quantification?

▶ ***Lemma 26*** Let ϕ be an $(x_1,...,x_n)$-ordered conjunctive query having its quantification part of the form $Q_k x_k ... Q_n x_n$. Then we have the equivalence $\mathcal{Q}(\phi[\{x_1,...,x_{k-1}\}]) \backsimeq \mathcal{Q}(\phi)$. In particular $\phi[\{x_1,...,x_{k-1}\}]$ is trivial if and only if ϕ is. ◀

Remark 28 The following three queries are easy, but the first two have defects that would make them hard if quantifiers where just $\exists$, and the third one exhibits a query can be easy even if no elimination order matches the nesting order of quantifiers.

$$() \mid \forall x \forall y \forall z \; R(x,y) \wedge R(z,y) \wedge R(x,z) \qquad \text{acyclicity defect}$$
$$(x,z) \mid \forall y \; R(x,y) \wedge R(z,y) \qquad \text{starring defect}$$
$$() \mid \forall x \exists y \forall z \; R(x,z) \wedge R(y,z) \qquad \text{ordering defect}$$

We can conclude the condition given in the previous statement is sufficient but not necessary. A tighter result would be:

▶ **Conjecture**

A conjunctive query of hypergraph $\mathcal{H}$ and of the form:

$$(z_1,...,z_m) \mid \forall x_1 \exists y_1 ... \forall x_n \exists y_n \; \psi$$

where ψ a quantifier-free conjunction belongs to the equivalence class of $\mathcal{H}[\{z_1,...,z_m\}]$ when:

$$\forall k \; \mathcal{A}_\alpha(\mathcal{H}\,[\{z_1,...,z_m,x_1,y_1,...,x_k,y_k\}] \cup \{x_1,y_1,...,x_k,z_1,...,z_m\}) \qquad ◀$$

Remark 29 This statement regards as easy the first two above counter-examples. What about the third?

Chapter 5

A Negative Conjunctive Query is Easy iff it is Beta-Acyclic

Contents

1 Preliminaries and Results 91
1.1 Preliminaries 91
1.2 Acyclicity Notions 93
1.3 Statement of the Results 94
2 Davis-Putnam Resolution w.r.t a Nest Point 95
2.1 Definition and Properties 95
2.2 Algorithm and Complexity 96
3 Easiness Result 97
3.1 NCQ on the Boolean Domain 98
3.2 Easiness Result for SCQ on the Boolean Domain . . 99
3.3 Final Easiness Result 101
4 Hardness Result 102
4.1 A Technical Point 102
4.2 Hardness Result 103

Abstract It is known that the data complexity of a Conjunctive Query (CQ) is determined *only* by the way its variables are shared between atoms, reflected by its hypergraph. In particular, Yannakakis [Yan81, BFMY83] proved that a CQ is decidable in linear time when it is α-acyclic, i.e. its hypergraph is α-acyclic; Bagan et al. [BDG07] even state:

- *Any CQ is decidable in linear*[1] *time iff it is α-acyclic (under certain hypotheses).*

A natural question is the following: since the complexity of a *Negative* Conjunctive Query (NCQ), a conjunctive query where *all* atoms are negated, also only depends on its hypergraph, can we find a similar dichotomy in this case?

To answer this question, we revisit a result of Ordyniak et al. [OPS10] — that states that satisfiability of a β-acyclic CNF formula is decidable in polynomial time — by proving that some part of their procedure can be done in linear time. This implies, under an algorithmic hypothesis [2] that is likely true:

1 We mean a query on a structure $\mathcal{S}$ can be decided in time $\mathcal{O}(|\mathcal{S}|)$.

2 Precisely: one cannot decide whether a graph is triangle-free in time $\mathcal{O}\left(n^2 \log n\right)$ where n is the number of vertices.

- *Any NCQ is decidable in quasi-linear*[3] *time iff it is β-acyclic.*

We extend the easiness result to *Signed* Conjunctive Query (SCQ) where *some* atoms are negated. This has great interest since using some negated atoms is natural in the frameworks of databases and CSP. Furthermore, it implies straightforwardly the following:

- *Any β-acyclic existential first-order query is decidable in quasi-linear*[3] *time.*

Introduction

This paper ([BB12a]) gives descriptive complexity results in a finite model theory framework. According to [Lib04], "Finite model theory studies the expressive power of logic on finite relational structures". Here we emphasize the complexity aspect of expressive power, in relation with the considered (first-order) logic fragment.

A fragment of great interest is the primitive positive fragment, i.e. the set of sentences one can build using only atoms — relations between variables —, the conjunction $\wedge$ and the existential quantifier $\exists$. This fragment, also known as Conjunctive Queries (CQ), is fundamental in database theory (see [CM77] for example) and in Constraint Satisfaction Problems (CSP) (see [DP89]). While general queries (first-order sentences) are PSpace-complete in terms of combined complexity, conjunctive queries are "only" NP-complete in terms of combined complexity, or W[1]-Complete ([GSS01]) in terms of parametrized complexity.

An important tractable class of conjunctive queries are the α-acyclic (or *acyclic*, for short) conjunctive queries. They are tractable in a very strong sense: Yannakakis [Yan81] proved that an α-acyclic conjunctive query ϕ on a relational structure $\mathcal{S}$ can be decided in time $\mathcal{O}(|\phi|.|\mathcal{S}|)$; in particular, it is fixed-parameter linear, i.e. it has linear data complexity. Bagan et al.([BDG07]) even states (partially) that α-acyclicity is a necessary condition of linear data complexity. Another result giving relevance to this class is that Gottlob et al. ([GLS01]) proved α-acyclic CQ to be LogCFL-complete for combined complexity.

Acyclic CQ [Kol03] have met a great interest essentially because they are the basis of important tractable classes of queries: they have been extended to several notions of queries not-too-far from α-acyclicity. In particular, this notion was extended to bounded treewidth CQ, a notion of bounded distance from being a tree, leading to classification results, see [GSS01]; and to the notion of bounded hyper-treewidth CQ [GLS02], for example. These classes allow polynomial reduction to acyclic CQ.

Nevertheless, in a database context, polynomiality is not a sufficient notion of tractability: a quadratic dependency on the database size is often considered intractable. A more reasonable notion would be quasi-linear time $n(\log n)^{\mathcal{O}(1)}$ (see [GS89]), which we take as a definition of tractability. Notice that, according to this definition of tractability, tractable CQ are still exactly CQ that have an α-acyclic hypergraph.

One can naturally ask oneself which queries, besides α-acyclic CQ, are tractable. Notice that there are some obviously tractable queries that are not in CQ, in particular some sets operations; e.g., the query $\exists x\, R(x) \wedge \neg S(x)$ that can be interpreted as "is $R \setminus S$ non-empty?". This example leads to a quite natural question: which extensions of CQ where

3 We mean a query on a structure $\mathcal{S}$ can be decided in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$.

some atoms are negated, which we call *Signed Conjunctive Queries* (SCQ), are tractable?

In order to answer this question, we investigate a simpler question: Which queries that are conjunctions of *only* negated atoms, that we call *Negative Conjunctive Queries* (NCQ), are tractable? The advantage of this simpler case is that we can use the same tool as the known CQ case: the notion of hypergraph. The hypergraph of a query is quite a simple object reflecting the way variables are shared between atoms. It is a widespread intuitive idea that the complexity of a CQ only depends on its hypergraph, and in fact it is easy to prove this also holds in the case of NCQ.

One might think that a NCQ can be immediately reduced to a CQ by complementing the relations. However, computing the complement is clearly not in FPT, and *a fortiori* not in quasi-linear time.

We will prove for this case a dichotomy similar to the one known for CQ: a NCQ is tractable iff it is β-acyclic, which means that its hypergraph is β-acyclic. We finally extend the easiness result by proving that β-acyclic SCQ are tractable.

Structure of this Paper This paper is organized as follows:

- Section 1 introduces the main definitions and states the results;
- Section 2 refines a result from Ordyniak et al. [OPS10] by proving Davis-Putnam resolution w.r.t. a variable that is a nest point is done in linear time;
- Section 3 proves the easiness result;
- Section 4, after introducing some technical points, establishes the hardness result.

1 Preliminaries and Results

We introduce here the respective statements of the two results with all necessary definitions.

1.1 Preliminaries

Definition 40 (sentence, structure, query, CQ, NCQ, SCQ)
A *signature* σ is:

- a set of relation symbols,
- and an arity Ar that associates a number to each symbol R, denoted $\mathrm{Ar}(R)$.

A *σ-structure* $\mathcal{S}$ consists in associating a set of $\mathrm{Ar}(R)$-tuples to each of the relation symbols R of σ which is called the *interpretation of R in $\mathcal{S}$*, and denoted $R^{\mathcal{S}}$. Some relation symbols of arity 1 may be used in a particular way, and are then called *domain symbols*.

An *existential first-order sentence*, or *sentence* for short, is a usual existential first-order sentence where each variable has a *distinct* associated domain symbol D_i. More formally, a σ-sentence has the form $\exists x_1 \in D_1 ... \exists x_n \in D_n\ \psi$ where D_i is a domain symbol belonging to σ and ψ is a usual quantifier-free σ'-formula where σ' is σ without the $D_1, ..., D_n$ previously mentioned. In the whole document, ψ will refer to the quantifier-free part of ϕ.

We call query of a sentence ϕ, denoted $?\mathcal{Q}(\phi)$, the problem of, given a structure $\mathcal{S}$, deciding whether ϕ holds in $\mathcal{S}$. Depending on the quantifier-free formula ψ, we define classes of queries of interest:

- when $\psi = \bigwedge_i R_i(x_{i_0},...,x_{i_{\mathrm{Ar}(R_i)}})$, $?\mathcal{Q}(\phi) \in$ CQ the *existential Conjunctive Queries*;
- when $\psi = \bigwedge_i \neg R_i(x_{i_0},...,x_{i_{\mathrm{Ar}(R_i)}})$, $?\mathcal{Q}(\phi) \in$ NCQ the *existential Negative Conjunctive Queries*;
- when $\psi = \bigwedge_i \sigma_i R_i(x_{i_0},...,x_{i_{\mathrm{Ar}(R_i)}})$, where σ_i is either $\neg$ or ϵ (nothing), $?\mathcal{Q}(\phi) \in$ SCQ the *existential Signed Conjunctive Queries*;
- when ψ is unrestricted, i.e. written using any connectives ($\wedge$, $\vee$, $\rightarrow$, $\neg$, etc.) $?\mathcal{Q}(\phi) \in$ EQ the *existential Queries*.

Considering the particular form of the first three classes, we may consider these formulas as conjunctions of so-called *conjuncts*, i.e. when ψ is in the form: $\psi = \bigwedge_i \sigma_i R_i(x_{i_0},...,x_{i_{\mathrm{Ar}(R_i)}})$ each $C_i = \sigma_i R_i(x_{i_0},...,x_{i_{\mathrm{Ar}(R_i)}})$ is a conjunct. When $\sigma(i)$ is ϵ (resp. $\neg$), we say that C_i is a *positive* (resp. *negative*) *conjunct*.

Remark 30 Defining sentences with distinguished domains symbols attached to variables is a bit unusual. Let us justify it briefly through a simple example. Consider:

$$\phi_1 = \exists x_1 \exists x_2 \exists x_3 \exists x_4 \; R(x_1,x_2) \wedge R(x_3,x_2) \wedge R(x_1,x_4) \wedge R(x_3,x_4)$$
$$\phi_2 = \exists x_1 \exists x_2 \exists x_3 \exists x_4 \; R(x_1,x_2) \wedge R(x_2,x_3) \wedge R(x_4,x_1) \wedge R(x_3,x_4)$$

The sentences ϕ_1 and ϕ_2 obviously have the same hypergraph (see below) but $?\mathcal{Q}(\phi_1)$ is obviously easy (it consists in deciding whether a directed graph has at least one edge), while $?\mathcal{Q}(\phi_2)$ is presumably not as easy (it consists in deciding whether a directed graph has a circuit of size 4). Using our formalism guarantees that:

- the complexity of a CQ (or a NCQ) depends only on its hypergraph, and
- the easiness results (of α-acyclic CQ and β-acyclic SCQ) still hold with the usual definition, i.e. when the variables all have the same domain.

Definition 41 (complexity classes) These definitions are based on the paper [Gra96], see it for more details. We call qLin (resp. Lin) the set of problems decidable in time $\mathcal{O}(n \log n)$ (resp. $\mathcal{O}(n)$) on a RAM machine where n is the size of the input. In particular, sorting is in Lin by a result of [AHU74], and [Gra96].

Remark 31 This definition of qLin looks quite restrictive: $n(\log n)^{\mathcal{O}(1)}$ would be much more reasonable, as suggested by [GS89]. In fact, all our results hold in both cases. We chose the restrictive one in order to put the emphasis on the easiness result.

The main object of our discourse will be how the restriction of the way variables are shared in a formula allows easy decision. The way variables are shared can be described by a very simple yet powerful notion: the hypergraph.

Definition 42 (hypergraph of a query) We call *hypergraph* $\mathcal{H}$ a set of non-empty sets, called the *edges* of $\mathcal{H}$. We call $\mathcal{V}(\mathcal{H})$ the union of its edges; the elements of $\mathcal{V}(\mathcal{H})$ are called the *vertices* of $\mathcal{H}$.

We write $\mathcal{H}(\phi)$ the hypergraph of the sentence ϕ defined as the set of variables sets appearing in atoms of ϕ:

$$\mathcal{H}(\phi) = \{\text{Vars}(A) \mid A \text{ is an atom of } \phi\}$$
$$\text{where Vars}\left(R_i\left(x_{i_1},...,x_{i_{\text{Ar}(R_i)}}\right)\right) = \left\{x_{i_1},...,x_{i_{\text{Ar}(R_i)}}\right\}$$

Example 32 If we consider the SCQ $?\mathcal{Q}(\phi)$ where:

$$\phi = \exists x_1 \in D_1\, \exists x_2 \in D_2\, \exists x_3 \in D_3\, \exists x_4 \in D_4\ \psi$$
$$\psi = R_1(x_1,x_2) \wedge R_2(x_2,x_3) \wedge R_3(x_1,x_2,x_3) \wedge \neg R_4(x_4,x_3) \wedge \neg R_5(x_4,x_4)$$

then we have $\mathcal{H}(\phi) = \{\{x_1,x_2\},\{x_2,x_3\},\{x_1,x_2,x_3\},\{x_3,x_4\},\{x_4\}\}$.

In this case, $R_1(x_1,x_2)$ is a positive conjunct. By contrast, $R_5(x_4,x_4)$ is an atom but not a conjunct in this formula, while $\neg R_5(x_4,x_4)$ is a negative conjunct.

We will see that the complexity of $?\mathcal{Q}(\phi)$ depends only on $\mathcal{H}(\phi)$. We now define the criterion that discerns easy queries from hard queries.

1.2 Acyclicity Notions

We will see that some interesting properties of hypergraphs are several notions of acyclicity, which are different extensions of the (classical) graph acyclicity property.

Definition 43 (induced hypergraph, nest point) Let $\mathcal{H}$ be a hypergraph. We define its induced hypergraph w.r.t. $S \subseteq \mathcal{V}(\mathcal{H})$, denoted $\mathcal{H}[S]$, as follows:

$$\mathcal{H}[S] = \{e \cap S \mid e \in \mathcal{H}\} \setminus \{\emptyset\}$$

We write $\mathcal{H}[\!\setminus\! S]$ as a shorthand for $\mathcal{H}[\mathcal{V}(\mathcal{H}) \setminus S]$. In particular, $\mathcal{H}[\!\setminus\!\{x\}]$ is the hypergraph obtained by weak vertex removal of x.

We say that x is a *nest point* of $\mathcal{H}$ when for any two distinct edges e_1 and e_2 containing x, either $e_1 \subset e_2$ or $e_2 \subset e_1$. In other words, the set $\{e \in \mathcal{H} \mid x \in e\}$ is linearly ordered w.r.t. set inclusion.

Fagin's originally defined ([Fag83]) α-acyclicity of a hypergraph $\mathcal{H}$, here denoted $\mathcal{A}_\alpha(\mathcal{H})$. We assume the reader is familiar with the notion of α-acyclicity, that is the most classical notion of acyclicity for hypergraphs. He also defined β-acyclicity as follows:

Definition 44 (β-acyclicity) We say that a hypergraph $\mathcal{H}$ is β-acyclic, denoted $\mathcal{A}_\beta(\mathcal{H})$, if each of its subsets, i.e. every $\mathcal{H}' \subseteq \mathcal{H}$, is α-acyclic.

This characterisation says that β-acyclicity is the "hereditary" closure of α-acyclicity.

We give a second characterisation of β-acyclicity. This inductive characterisation from [Dur08, Dur09], based on a result of [BK80], is useful for the algorithm we give for β-acyclic queries (easiness result).

► ***Lemma 27 (β-acyclicity inductive characterisation)*** A hypergraph $\mathcal{H}$ is β-acyclic iff *either* it is empty, *or* such that:

- we can find $x \in \mathcal{V}(\mathcal{H})$ such that x is a nest point of $\mathcal{H}$ *and*
- $\mathcal{H}[\!\setminus\!\{x\}]$ is also β-acyclic.

We say that the ordered list $(x_1,...,x_n)$ of the vertices of an hypergraph $\mathcal{H}$ is a *Reverse Elimination Order* (*REO*) of $\mathcal{H}$ when, for all i in $\{1,...,n\}$, x_i is a nest point of $\mathcal{H}[\{x_1,...,x_{i-1}\}]$. By this characterisation, a hypergraph is β-acyclic iff it has a REO. ◄

Here is the third characterisation of β-acyclicity. It will be used to obtain our hardness result. This characterisation uses the notion of β-cycle defined here.

Definition 45 (β-cycle) A *chordless cycle* is a graph isomorphic to:

$$\{\{x_i, x_{i+1}\} \mid 1 \leq i \leq k\} \cup \{\{x_k, x_1\}\}$$

for some k. A β-cycle of a hypergraph $\mathcal{H}$ is a subset of some induced sub-hypergraph of $\mathcal{H}$ that is a chordless cycle. More formally, we say that C is a β-cycle of $\mathcal{H}$ when C is a chordless cycle and:

$$\exists S \subseteq \mathcal{V}(\mathcal{H})\ C \subseteq \mathcal{H}[S]$$

▶ ***Lemma 28 (β-acyclicity as absence of β-cycles)*** An hypergraph $\mathcal{H}$ is β-acyclic iff it does not have a β-cycle. ◀

1.3 Statement of the Results

With all these notations, we are now able to state our results:

▶ **Theorem 20 (dichotomy)**

Under hypothesis that the presence of a triangle in a graph of n vertices cannot be decided in time $\mathcal{O}(n^2 \log n)$, we have:

$$\forall \phi \in \text{NCQ} \quad ?\mathcal{Q}(\phi) \in \text{qLin} \quad \Leftrightarrow \quad \mathcal{A}_\beta(\mathcal{H}(\phi))$$

Proof. Lemma 36 proves the implication $\Leftarrow$ (easiness result) and Lemma 41 proves the implication $\Rightarrow$ (hardness result). ◀

Remark 33 This is to be compared to the positive conjunctive queries dichotomy contained in [BDG07]: for any $\phi \in$ CQ such that $\mathcal{H}(\phi)$ is 4-conformal,[4] we have:

$$\forall \phi \in \text{CQ} \quad ?\mathcal{Q}(\phi) \in \text{Lin} \quad \Leftrightarrow \quad \mathcal{A}_\alpha(\mathcal{H}(\phi))$$

under hypothesis we cannot decide the presence of a triangle in a graph $\mathcal{G}$ in time $\mathcal{O}(|\mathcal{G}|)$.

Notice that this result also holds if we replace Lin by qLin; in that case the hypothesis becomes "deciding the presence of a triangle in a graph is not in qLin".

▶ **Theorem 21 (easiness)**

We have:

$$\forall \phi \in \text{EQ} \quad ?\mathcal{Q}(\phi) \in \text{qLin} \quad \Leftarrow \quad \mathcal{A}_\beta(\mathcal{H}(\phi))$$

That is to say any β-acyclic existential first-order sentence is decidable in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$.

Let P be a property on hypergraphs. Under the previously mentioned complexity hypothesis (about the triangle problem), we have:

$$(\forall \phi \in \text{EQ}\ P(\mathcal{H}(\phi)) \Rightarrow ?\mathcal{Q}(\phi) \in \text{qLin}) \quad \Leftrightarrow \quad (\forall \mathcal{H}\ P(\mathcal{H}) \Rightarrow \mathcal{A}_\beta(\mathcal{H}))$$

This means: any property of the hypergraph of a query grants it quasi-linear decision time if and only if this property implies β-acyclicity.

Proof. Put ϕ in Disjunctive Normal Form, distribute (existential) quantification over disjunctive clauses which is correct since $\exists x(A(x) \vee B(x)) \Leftrightarrow (\exists x A(x)) \vee (\exists x B(x))$. Each clause is a SCQ whose hypergraph is a subset

4 A hypergraph is said to be k-conformal when every clique of cardinality $\geq k$ is contained in an edge.

of $\mathcal{H}(\phi)$, each clause is therefore a β-acyclic SCQ; by Lemma 36, it has a QLIN decision time, therefore so has their disjunction ϕ.

Second point is a direct corollary of this easiness result (part $\Leftarrow$) and the hardness part of the NCQ dichotomy (part $\Rightarrow$). ◀

2 Davis-Putnam Resolution w.r.t a Nest Point

This section gives an algorithmic result needed for the easiness result; it can be independently read for its own, or may be skipped by admitting it. This part consists in exploiting a particular property of a variable in a CNF formula to perform efficient Davis-Putnam resolution with respect to this variable.

2.1 Definition and Properties

Definition 46 (CNF formula–related) A *CNF formula* $F(x_1,...,x_n)$ is a classic propositional formula on the variables $x_1,...,x_n$ that is in Conjunctive Normal Form, i.e. $F(x_1,...,x_n) = \bigwedge_i C_i$ where C_i are *clauses*, i.e. sub-formulae in the form $C_i = \bigvee_{j=1}^{n(i)} l_i^j$ where l_i^j are *literals*. Literals are either *positive* literals — variables taken in $\{x_1,...,x_n\}$ — or *negative* literals — negated variables $\neg x_i$ where $x_i \in \{x_1,...,x_n\}$. A formula in CNF can be thought of as a set of clauses, which are sets of literals.

We say that a clause C *holds* a variable x when either $x \in C$ or $\neg x \in C$ (or both, but the clause is tautological in this case). The set of variables held by a clause C is denoted $\text{Vars}(C) = \{x_i \mid x_i \in C \text{ or } \neg x_i \in C\}$. We write F_x the set of clauses of F holding a variable x. We say a variable x_i is a *nest point* of $F(x_1,...,x_n)$ (see [OPS10]) when for all C_1 and C_2 in F_{x_i}, either $\text{Vars}(C_1) \subseteq \text{Vars}(C_2)$ or $\text{Vars}(C_2) \subseteq \text{Vars}(C_1)$.

We define the resolvent of $F(x_1,...,x_n)$ w.r.t. the variable x_1:

$$\text{Res}(F,x_1)(x_1,...,x_n) = \left\{ C_1 \vee C_2 \;\middle|\; \begin{array}{l} (C_1 \vee \ x_1) \in F(x_1,...,x_n) \text{ and} \\ (C_2 \vee \neg x_1) \in F(x_1,...,x_n) \end{array} \right\}$$

The following results are from [OPS10].

▶ ***Lemma 29 (resolvent properties)*** The following propositions hold:

- the Davis-Putnam resolution is correct:

$$\forall x_1,...,x_n \ \text{Res}(F,x_1)(x_1,...,x_n) \ \Leftrightarrow \ \exists x_1 \ F_{x_1}(x_1,...,x_n)$$

- if x_1 is a nest point of a formula F, then:

$$\text{Res}(F,x_1) \text{ is a subset of } \{C \setminus \{x_1, \neg x_1\} \mid C \in F_{x_1}\}$$

Proof. Correctness of the resolution is classical [DP60] and easy.

Let x_1 be a nest point of F. We just have to consider F_{x_1}. Since x_1 is a nest point of F_{x_1}, in F_{x_1} we can rename variables as x_1, ..., x_n by growing size of the smallest clause containing them. Obviously, $x = x_1$, and all clauses C are such that $\text{Vars}(C) = \{x_1,...,x_k\}$ with $k \leq n$. When the computation is over, we just have to give their original names back to the variables.

Take any two clauses $C_1 \vee x$ and $C_2 \vee \neg x$ involved in resolvent computation. We know $\text{Vars}(C_1) \subseteq \text{Vars}(C_2)$ or the converse. Assume, w.l.o.g, $\text{Vars}(C_1) \subseteq \text{Vars}(C_2)$. If some variable is present with different signs in C_1 and C_2, then the resolvent is tautological. Assume this is not the case. Therefore $C_1 \vee C_2 = C_2$. ◀

```
ComputeResolvent(S_−, S_+):
    Res ← ∅
    i_− ← 1; i_+ ← 1
    while i_− < n_− and i_+ < n_+:
        if S_−[i_−] is a prefix of S_+[i_+]:
            Res ← Res ∪ {S_+[i_+]}
            Increment i_+
        elsif S_+[i_+] is a prefix of S_−[i_−]:
            Res ← Res ∪ {S_−[i_−]}
            Increment i_−
        elsif S_+[i_+] < S_−[i_−]:
            Increment i_+
        elsif S_+[i_+] > S_−[i_−]:
            Increment i_−
    return Res
```

● **Algorithm 5.1:** Resolvent computation.

2.2 Algorithm and Complexity

Here we prove that the resolution can be done in linear time, which is our addition to the main result of Ordyniak et al. ([OPS10])

▶ ***Lemma 30 (resolvent computation)*** Let $F(x_1,...,x_n)$ be a CNF sentence whose x_1 is a nest point. We can compute the resolvent of $F(x_1,...,x_n)$ w.r.t. x_1 in time $\mathcal{O}(|F|)$.

Proof. The main algorithmic point consists in adopting a handy representation of the formula. Like in proof of Lemma 29, since x_1 is a nest point of F, we can rename variables in F_{x_1} by growing size of the smallest clause containing it, therefore all clauses C are such that $\mathrm{Vars}\,(C) = \{x_1,...,x_k\}$ with $k \leq n$. Clauses can therefore be encoded as *words* over the alphabet $\{-,+\}$. For example, $x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4$ is encoded as $+--+$.

Let us consider subsumed clauses, i.e., clauses C such that we can find C' such that $C' \subseteq C$. With this encoding, C_1, encoded be w_1, subsumes C_2, encoded by w_2, if and only if w_1 is a prefix of w_2. Getting rid of subsumed clauses is done by applying the following simple algorithm, where L is the list of words encoding the set of clauses.

```
RemovePrefixed(L):
    Sort lexicographically L
    n ← 1
    for i from 2 to Card (L):
        if L[n] not prefix of L[i]:
            L[n + 1] ← L[i]
            n ← n + 1
    L[n + 1] ← End-Of-List
```

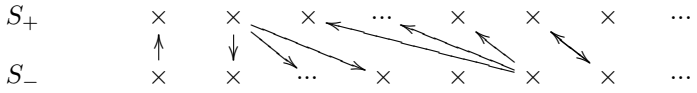
It is easy to see that this algorithm is correct, i.e. eliminates (in-place) any word that is prefixed by another. Sorting in lexicographical order, here denoted $<$, can be done in linear time using algorithm 3.2 page 80 in [AHU74], see also definition 41. Therefore previous algorithm is linear.

In order to compute the resolvent of F with respect to x_1, we just need to consider pairs of clauses such that one is in the form $+...$ and the other in the form $-...$ Let us construct the lexicographically ordered sub-lists S_- and S_+ of words beginning with $-$, resp. $+$, without their

leading $-$, resp. $+$. Two clauses C_1 and C_2 respectively encoded by words w_1 and w_2 have a non-tautological resolvent on x_1 if and only if $w_1 \in S_-$ and $w_2 \in S_+$ (or the converse), and w_1 is a prefix of w_2: in this case, their "resolvent" is w_2 itself, i.e. $C_2 \setminus \{x_1\}$.

Algorithm 5.1, where S_- (resp. S_+) is represented by a sorted list L_1 (resp. L_2) of n_1 (resp. n_2) elements, is a variant of the fusion algorithm of two sorted lists. It is obviously linear.

In order to prove Algorithm 5.1 is correct, we consider the bipartite directed graph where $a \to b$ means a is a prefix of b, and where words are represented in growing lexicographical order (from left to right).

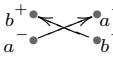
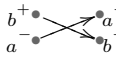


The constraints of this graph are given by the assertions 1-4 given below. As a preliminary, we let the reader convince himself the following fact holds: if a word a is a prefix of a word b, then for all words w and W such that $w < a < W$ and none is a prefix of another, $w < b < W$.

1 The in-degree of the graph is at most one: assuming the contrary leads to $\exists u, v \in S_-$ (resp. S_+) with u prefix of v.

2 The graph has no path of length 2: exactly the same proof.

3 Edges do not cross each other. By symmetry, only two cases have to be considered:

- Edges do not cross each other in this fashion:  i.e. we can't have a^- prefix of a^+ and b^- prefix of b^+ with $a^- < b^-$ and $a^+ > b^+$. Assume this is the case. a^- is prefix of a^+ and inferior and not prefix of b^-. Therefore $a^+ < b^-$. Since b^- is a prefix of b^+, $b^- < b^+$, therefore $a^+ < b^+$ (by the preliminary fact), contradictory.
- Edges don't cross each other in this fashion:  i.e. we can't have a^- prefix of a^+ and b^+ prefix of b^- with $a^- < b^-$ and $a^+ > b^+$: essentially the same proof, but using the preliminary fact twice.

4 The neighbourhood of a vertex is contiguous, i.e. if a^- is a prefix of both b^+ and c^+, then for all $b^+ < d^+ < c^+$, a^- is a prefix of d^+.

All these facts together prove that no pair (a, b) where a is a prefix of b is "forgotten" by the given algorithm, which is therefore correct. ◀

3 Easiness Result

In this section, we prove that one can decide a β-acyclic SCQ with n variables on a structure $\mathcal{S}$ in time $\mathcal{O}(n|\mathcal{S}| \log |\mathcal{S}|)$. We prove it a bottom-up fashion: each step reduces to or generalizes the previous one:

- first step uses results of the first section (Davis-Putnam resolvent computation w.r.t. a nest point in linear time) to decide β-acyclic NCQ-BoolD fast,
- second step generalizes it to β-acyclic SCQ-BoolD, and
- third step reduces β-acyclic SCQ to β-acyclic SCQ-BoolD.

For the sake of simplicity, we always assume a sentence is simple in the following sense: relation symbols all appear once, and in an atom, each variable appears at most once, and in any order of our convenience. This is justified by Corollary 22, page 103.

3.1 NCQ on the Boolean Domain

Here we make use of the results of previous section in order to get the announced complexity.

Definition 47 (NCQ-BoolD, SCQ-BoolD) We say a NCQ (resp. SCQ) is *over the boolean domain* when all its domains are fixed to $\{0,1\}$ instead of being defined with domain symbols; we write it NCQ-BoolD (resp. SCQ-BoolD).

As an example, if $\phi = \exists x_1 \in D_1 ... x_n \in D_n\, \psi(x_1,...,x_n)$ is a NCQ, then $\phi' = \exists x_1 ... x_n \in \{0,1\}^n \psi(x_1,...,x_n)$ is a NCQ-BoolD.

The main result of this subsection, Lemma 32, gives an easiness result for NCQ-BoolD having a β-acyclic hypergraph. This result is obtained using inductively the following lemma jointly with the inductive characterisation of β-acyclicity by nest points.

▶ ***Lemma 31 (NCQ-BoolD nest point)*** Let ϕ be a NCQ-BoolD, $\mathcal{S}$ a structure, and x a nest point of $\mathcal{H}(\phi)$. We can build another NCQ-BoolD ϕ' and another structure $\mathcal{S}'$ in time $\mathcal{O}(|\mathcal{S}|)$ (*independent* of $|\phi|$) such that $?\mathcal{Q}(\phi)(\mathcal{S}) \Leftrightarrow ?\mathcal{Q}(\phi')(\mathcal{S}')$ and $\mathcal{H}(\phi') = \mathcal{H}(\phi)\lfloor\{x\}\rfloor$.

Proof (Sketch). Take $\phi = \exists x_1 ... x_n \in \{0,1\}^n \psi(x_1,...,x_n)$ with:

$$\psi(x_1,...,x_n) = \bigwedge_i \neg R_i\left(x_{f(i,1)},...,x_{f(i,n(i))}\right)$$

The main point of the proof is transforming the assertion $\mathcal{S} \vDash \psi$ into an equivalent CNF formula, and then applying CNF nest point results. For all $(x_1,...,x_n) \in \{0,1\}^n$, we have:

$$\begin{aligned}
\mathcal{S} \vDash \psi(x_1,...,x_n) &\Leftrightarrow \bigwedge_i \neg R_i^{\mathcal{S}}\left(x_{f(i,1)},...,x_{f(i,n(i))}\right) \\
&\Leftrightarrow \bigwedge_i \bigwedge_{(a_1,..,a_{n(i)}) \in R_i^{\mathcal{S}}} \underbrace{(x_{f(i,1)},...,x_{f(i,n(i))}) \neq (a_1,...,a_{n(i)})}_{C_i(a_1,..,a_{n(i)})} \\
C_i(a_1,...,a_{n(i)}) &\Leftrightarrow (x_{f(i,1)} \neq a_1) \vee ... \vee (x_{f(i,n(i))} \neq a_{n(i)}) \\
&\Leftrightarrow \sigma(a_1)x_{f(i,1)} \vee ... \vee \sigma(a_{n(i)})x_{f(i,n(i))}
\end{aligned}$$

where σ maps 1 to $\neg$ and 0 to ϵ. Finally, for all x_1, ..., x_n we have the equivalence:

$$\mathcal{S} \vDash \psi(x_1,...,x_n) \Leftrightarrow \underbrace{\bigwedge_i \bigwedge_{(a_1,..,a_{n(i)}) \in R_i^{\mathcal{S}}} \sigma(a_1)x_{f(i,1)} \vee ... \vee \sigma(a_{n(i)})x_{f(i,n(i))}}_{F(x_1,..,x_n)}$$

Clearly, x is a nest point of $F(x_1,...,x_n)$ (see the note at the end of previous definition). The transformation $(\phi, \mathcal{S}) \mapsto F$ is done in time $\mathcal{O}(|\mathcal{S}|)$ — i.e. linear time [5]. It is rather easy to see that the reverse transformation (from the propositional formula back to the corresponding NCQ-BoolD) can be done in linear time.

Lemma 29 states that $\mathrm{Res}\,(F,x)$ is a subset of F_x where x was removed, and that Davis-Putnam resolution is correct:

$$\forall x_1,...,x_n\ \mathrm{Res}\,(F,x)\,(x_1,...,x_n) \Leftrightarrow \exists x\ F_x(x_1,...,x_n)$$

Notice that $x \in \{x_1,...,x_n\}$. Furthermore, Lemma 30 states that this can be done in linear time.

5 Not exactly: the variables of the CNF are "bigger" than the 0/1 present in the structure. Nevertheless, when applying resolution, they will be encoded as signs, i.e. either + or −. In fact, the CNF form *explains* correctness but is not an actual step.

The following algorithm uses previous notations and is justified by the arguments above.

RemoveNestPoint(ϕ,x,$\mathcal{S}$):
    // ψ is in the following form: $\bigwedge_i \neg R_i\left(x_{f(i,1)},...,x_{f(i,n(i))}\right)$
    // Recall that $\sigma(1) = \neg$, $\sigma(0) = \epsilon$
    $F \leftarrow \bigwedge_i \bigwedge_{(a_1,..,a_{n(i)}) \in R^{\mathcal{S}}{}_i} \sigma(a_1)x_{f(i,1)} \vee ... \vee \sigma(a_{n(i)})x_{f(i,n(i))}$
    Replace $\neg R(x_1,...,x_n,x)$ by $\neg R(x_1,...,x_n)$ in ϕ
    $F \leftarrow Res(F,x)$
    Rebuild $\mathcal{S}$ from F
    return (ϕ,$\mathcal{S}$)

This algorithm clearly uses linear time, i.e. $\mathcal{O}(|\mathcal{S}|)$. ◀

We now state the NCQ easiness result. This lemma is stated and proved in order to give a simplified view of Lemma 34 below.

▶ ***Lemma 32 (NCQ-BoolD easiness)*** A NCQ-BoolD ϕ with n variables such that $\mathcal{H}(\phi)$ is β-acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}|)$.

Proof. The following does it:

?NCQ – BoolD(ϕ)($\mathcal{S}$):
    We know $(x_1,...,x_n)$ is a REO of ϕ.
    for i from n to 1:
        $(\phi, \mathcal{S}) \leftarrow$ RemoveNestPoint(ϕ,x_i,$\mathcal{S}$)
        while ϕ contains some negative conjunct $\neg R()$:
            if $R^{\mathcal{S}} \neq \emptyset$: **return** False
            Remove $\neg R()$ from ϕ
    return True

Let us justify its correctness. By previous β-acyclicity characterisation, we know we can apply nest point removing until there is no variable left.

While nest points are inductively removed, some relations become of arity 0. The interpretation of these relations R is either empty — in this case, the assertion $\neg R()$ is a tautology and can be removed from the conjunction — or non-empty, i.e. contains only the empty tuple. In this case, the assertion $\neg R()$ is a contradiction, and the query should return "false" to mean this conjunction is not satisfiable (i.e. the sentence is not satisfied).

Since resolvent computation is done in linear time, each loop turn takes linear time, which happens n times. ◀

3.2 Easiness Result for SCQ on the Boolean Domain

We now refine previous result, instead of using it. It is barely more complicated.

Instead of proving easiness of general β-acyclic NCQ, we directly treat the case of SCQ. A first reason is that domains will be positive relations, therefore a NCQ is already a kind of SCQ; treating directly SCQ avoids treating domains specifically. The other reason is discussed after the following lemma.

This trick is a (weak) variant of the one presented in [ZH02], and was inspired by it. It looks simple but is the key point allowing extension of the NCQ easiness result to wider classes.

?SCQ − BoolD(ϕ)($\mathcal{S}$):
    We know $(x_1,...,x_n)$ is a REO of ϕ.
    for i from n to 1 :
        while there is more than one positive conjunct containing x_i in ϕ :
            Take $R(\bar{y},\bar{z},x_i)$ and $S(\bar{z},x_i)$ among them
            $R^{\mathcal{S}} \leftarrow \{(\bar{a},\bar{b},c) \in R^{\mathcal{S}} \mid (\bar{b},c) \in S^{\mathcal{S}}\}$
            Remove $S(\bar{z},x_i)$ from ϕ
        if x_i in contained in a positive conjunct $R_a^+(\bar{y},x)$:
            Build $(R_{a-1}^+)^{\mathcal{S}}$, $(R_a^-)^{\mathcal{S}}$ from $(R_a^+)^{\mathcal{S}}$
            Replace $R_a^+(\bar{y},x_i)$ by $R_{a-1}^+(\bar{y}) \wedge \neg R_a^-(\bar{y},x_i)$ in ϕ
            $(\phi,\mathcal{S}) \leftarrow$ RemoveNestPoint(ϕ,x_i,$\mathcal{S}$)
            $(R_{a-1}^+)^{\mathcal{S}} \leftarrow (R_{a-1}^+)^{\mathcal{S}} \setminus (R_a^-)^{\mathcal{S}}$
        else
            $(\phi,\mathcal{S}) \leftarrow$ RemoveNestPoint(ϕ,x_i,$\mathcal{S}$)
        while ϕ contains a negative conjunct $\neg R()$ (resp. positive $R()$) :
            if $R^{\mathcal{S}} \neq \emptyset$ (resp. $R^{\mathcal{S}} = \emptyset$) :
                return False
            Remove $\neg R()$ (resp. $R()$) from ϕ
    return True

● **Algorithm 5.2:** SCQ-BoolD decision algorithm.

▶ ***Lemma 33*** Let R_a^+ be a boolean relation of arity a. We can build in linear time the boolean relations R_{a-1}^+ of arity $a-1$ and R_a^- of arity a such that, for all $x_1,...,x_a \in \{0,1\}^a$:

- $R_a^+(x_1,...,x_a) \Leftrightarrow R_{a-1}^+(x_1,...,x_{a-1}) \wedge \neg R_a^-(x_1,...,x_a)$
- $(x_1,...,x_a) \in R_a^- \Rightarrow (x_1,...,x_{a-1}) \in R_{a-1}^+$

Proof. Let:

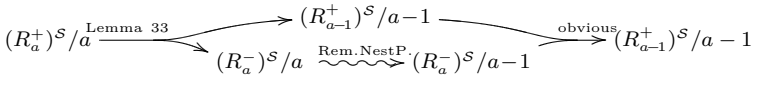
$R_{a-1}^+ = \{(e_1,...,e_{a-1}) \mid \exists e_a \; R_a^+(e_1,...,e_a)\}$
$R_a^- = \left(R_{a-1}^+ \times \{0,1\}\right) \setminus R_a^+$ ◀

We could transform a SCQ-BoolD into a NCQ-BoolD, at the cost of an additional n factor, that would affect the complexity of general SCQ. However, by making a somewhat *ad hoc* algorithm, we can treat the SCQ-BoolD case *directly* and get the expected complexity $\mathcal{O}(n|\mathcal{S}|)$.

▶ ***Lemma 34 (SCQ-BoolD easiness)*** A SCQ-BoolD ϕ with n variables, and whose hypergraph $\mathcal{H}(\phi)$ is β-acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}|)$.

Proof. Notice the statements of the proof of Lemma 32 still hold. Apply Algorithm 5.2. If several positive conjuncts hold a given nest point, then we can proceed to filtering as follows: take any two positive conjuncts. The set of variables hold by one includes the set of variables held by the other, we can sort them in a lexicographical order such that the set of shared variables have the strong weight of the lexicographical order; we can finally proceed to filtering in linear time. β-acyclicity is preserved, and the REO is maintained by edge removal. In the end, there is only one positive conjunct holding the nest point.

Now the positive conjunct can be managed with Lemma 33, summed up in the following, where R/a means the relation R has arity a.



We build $(R^+_{a-1})^{\mathcal{S}}$, $(R^-_a)^{\mathcal{S}}$ from $(R^+_a)^{\mathcal{S}}$ in linear time. After linear time resolvent computation, $(R^-_a)^{\mathcal{S}}$ has arity $a-1$ (the nest point has been removed) and we can proceed to a simplification in linear time justified by the equivalence:

$$\begin{aligned} & (R^+_{a-1})^{\mathcal{S}}(x_1,...,x_{a-1}) \wedge \neg (R^-_a)^{\mathcal{S}}(x_1,...,x_{a-1}) \\ \Leftrightarrow \quad & ((R^+_{a-1})^{\mathcal{S}} \diagdown (R^-_a)^{\mathcal{S}})(x_1,...,x_{a-1}) \end{aligned}$$

Since resolvent computation is done in linear time, each loop turn takes linear time, which happens n times. ◀

3.3 Final Easiness Result

Here is the last (technical) result on hypergraphs we need.

▶ ***Lemma 35*** Let $\mathcal{H}_1$, $x \in \mathcal{V}(\mathcal{H}_1)$, $y \notin \mathcal{V}(\mathcal{H}_1)$ and:

$$\mathcal{H}_2 = \{e \cup \{x,y\} \mid e \cup \{x\} \in \mathcal{H}_1\} \cup \{e \in \mathcal{H}_1 \mid x \notin e\}$$

This means $\mathcal{H}_2$ is the same hypergraph as $\mathcal{H}_1$ except that every edge holding x also holds y.

We have: $(a_1,...,a_n,x,b_1,...,b_m)$ is a REO of $\mathcal{H}_1$ if and only if both $(a_1,...,a_n,x,y,b_1,...,b_m)$ and $(a_1,...,a_n,y,x,b_1,...,b_m)$ are REOs of $\mathcal{H}_2$. In particular, $\mathcal{A}_\beta(\mathcal{H}_1) \Leftrightarrow \mathcal{A}_\beta(\mathcal{H}_2)$.

Proof. Easy considering the inductive definition of β-acyclicity (definition 27). ◀

▶ ***Lemma 36 (SCQ easiness)*** A SCQ ϕ with n variables, and whose hypergraph $\mathcal{H}(\phi)$ is β-acyclic can be decided in time $\mathcal{O}(n|\mathcal{S}|\log|\mathcal{S}|)$.

Proof. We transform a β-acyclic SCQ with n variables into a β-acyclic SCQ-BoolD having $n\log|\mathcal{S}|$ variables. Let ϕ be a β-acyclic SCQ admitting $(x_1,...,x_n)$ as a REO. Finding this elimination order takes a time *that only depends on* ϕ.

We assume domains are reasonably encoded: we suppose there is a constant k such that, for any domain D_i, $\lceil \log_2 \max_i(D_i^{\mathcal{S}}) \rceil < k \log|\mathcal{S}|$. This assumption is reasonable: we always can sort (in linear time) the union of the domains and associate each element with its number in this order. Then we can re-encode (in-place) the whole structure in time $\mathcal{O}(|\mathcal{S}|\log|\mathcal{S}|)$ by, for each occurring element, looking for its number, and replacing it by its number. Each element has therefore a size $\mathcal{O}(\log|\mathcal{S}|)$.

Let $s_i = \lceil \log_2 \max(D^{\mathcal{S}}{}_i) \rceil$. By the previous point, $s_i = \mathcal{O}(\log|\mathcal{S}|)$. We define:

$$\phi' = \exists x_1 \in \{1,...,2^{s_1}\}...\exists x_n \in \{1,...,2^{s_n}\}\ D_1(x_1) \wedge ... \wedge D_n(x_n) \wedge \psi$$

We have

$$\mathcal{H}(\phi') = \mathcal{H}(\phi) \cup \{\{x_i\} \mid x_i \in \mathcal{V}(\mathcal{H}(\phi))\}$$

therefore $\mathcal{H}(\phi')$ is β-acyclic by previous lemma.

Now we can transform ϕ' into a SCQ-BoolD:

$$\phi'' = \exists (x_1^1,...,x_1^{s_1},...,x_n^{s_n}) \in \{0,1\}^{\Sigma s_i} D_1(x_1^1,...,x_1^{s_1}) \wedge ... \wedge D_n(x_n^1,...,x_n^{s_n}) \wedge \psi'$$

with ψ' the same as ψ with every $R(x_a,x_b,...)$ replaced by:

$$R(x_a^1,...,x_a^{s_a},x_b^1,...,x_b^{s_b},...)$$

ϕ'' is a SCQ-BoolD, also β-acyclic due to Lemma 35.

Moreover, $(x_1^1,...,x_1^{s_1},...,x_n^1,...,x_n^{s_n})$ is a REO of $\mathcal{H}(\phi'')$. Noting that the number of variables $\Sigma_{i=1}^n s_i = \mathcal{O}(n\log|\mathcal{S}|)$ and applying Lemma 34 concludes. ◀

4 Hardness Result

We introduce a basic notion of reduction together with a trivial lemma that will be useful to prove Corollary 22, and makes simple the proof of the hardness result.

Definition 48 (linear reduction $\prec$) We say a problem P_1 *reduces linearly* to a problem P_2, denoted $P_1 \prec P_2$, when we can find $f \in \text{LIN}$ such that, given an algorithm A_2 deciding P_2, $A_2 \circ f$ decides P_1 that is to say the following algorithm decides P_1:

$?P_1(I_1)$:
    $I_2 \leftarrow f(I_1)$
    return $?P_2(I_2)$

We also say P_2 *expresses* P_1. When P_1 both reduces linearly to and expresses linearly P_2, we say P_1 and P_2 are *equivalent*, denoted $P_1 \sim P_2$.

The following lemma is obvious:

▶ ***Lemma 37 (linear reduction properties)*** Linear reduction is transitive, i.e. if $P_1 \prec P_2$ and $P_2 \prec P_3$, then $P_1 \prec P_3$; and QLIN is closed by linear reduction, i.e. if $P_1 \prec P_2$ and $P_2 \in \text{QLIN}$ then $P_1 \in \text{QLIN}$. ◀

4.1 A Technical Point

In order to prove our hardness result on NCQ, we need to introduce a normalized form of NCQ; it is used to show that the complexity of a NCQ is exactly determined by its hypergraph.

Definition 49 (simple, sorted, normalized NCQ) One says that a NCQ $\phi = \exists x_1 \in D_1 ... \exists x_n \in D_n\ \psi$ is *simple* if each relation symbol occurs only once in ψ. ϕ is *sorted* if the tuple of variables of each atom of ψ occurs in increasing order of subscripts with no repetition, i.e. every atom of ψ is in the form $R(x_{i_1},...,x_{i_k})$ where $i_1 < ... < i_k$.

A NCQ is *normalized* if it is both simple and sorted.

The following lemma is obvious:

▶ ***Lemma 38*** Let ϕ_1, ϕ_2 be two normalized *NCQ* with the same signature and $\mathcal{H}(\phi_1) = \mathcal{H}(\phi_2)$. Then ϕ_1 and ϕ_2 are identical up to relation symbols permutation. In particular $?\mathcal{Q}(\phi_1) \sim ?\mathcal{Q}(\phi_2)$. ◀

▶ ***Lemma 39*** Any NCQ ϕ is equivalent to some normalized NCQ ϕ'' with $\mathcal{H}(\phi) = \mathcal{H}(\phi'')$, i.e. $?\mathcal{Q}(\phi) \sim ?\mathcal{Q}(\phi'')$.

Proof (Sketch). Here we give the only non-trivial point of the proof, in an easy to generalize example. Let $\phi_1 = \exists x_1 \in D_1 \exists x_2 \in D_2 \exists x_3 \in D_3\ \neg R_1(x_1,x_2) \wedge \neg R_2(x_2,x_3)$ and ϕ_2 built by replacing *both* R_1 and R_2 by the same symbol R in ϕ_1. We prove $?\mathcal{Q}(\phi_1) \prec ?\mathcal{Q}(\phi_2)$. We define, for $i \in \{1,2,3\}$, $D_i^{\mathcal{S}_2} = D_i^{\mathcal{S}_1} \times \{i\}$ and:

$$R^{\mathcal{S}_2} = \{((a_1,1),(a_2,2)) \mid (a_1,a_2) \in R_1^{\mathcal{S}_1}\} \cup \{((a_2,2),(a_3,3)) \mid (a_2,a_3) \in R_2^{\mathcal{S}_1}\}$$

The key point is that $R^{\mathcal{S}_2}$ is a *disjoint* union of relations corresponding to relations $R_1^{\mathcal{S}_1}$ and $R_2^{\mathcal{S}_1}$; further, the tuples originating from $R_1^{\mathcal{S}_1}$ (resp. $R_2^{\mathcal{S}_1}$) are *identified* by their specific form $((a_1,1),(a_2,2))$ (resp. $((a_2,2),(a_3,3))$). ◀

▶ **Corollary 22 (hypergraph as only relevant aspect)**
For all NCQ ϕ_1 and ϕ_2, we have:

$$\mathcal{H}(\phi_1) = \mathcal{H}(\phi_2) \quad \Rightarrow \quad ?\mathcal{Q}(\phi_1) \sim ?\mathcal{Q}(\phi_2)$$

Proof. Obvious corollary of Lemma 38 and Lemma 39. ◀

4.2 Hardness Result

▶ ***Lemma 40*** Let ϕ be a NCQ, and x a variable appearing in ϕ. Let ϕ' be the query obtained by removing every occurrence of x in ϕ, that is to say removing the $\exists x \in D_x$, and removing x in atoms where it occurs. Then $?\mathcal{Q}(\phi)$ expresses linearly $?\mathcal{Q}(\phi')$.

Proof. By virtue of Corollary 22, we can assume that every relation appears once in ϕ, with associated variables in order. For simplicity, the variable x is therefore assumed the minimal element for the order. To define the reduction, we will just define how a given relation R is transposed between $\mathcal{S}_1$ and $\mathcal{S}_2$. If x does not appear in the corresponding atom, there is no difference i.e. $R^{\mathcal{S}_2} = R^{\mathcal{S}_1}$.

In the other case, R appears as $R(x, x_{a(1)}, ..., x_{a(k)})$ in ϕ, with $a : \{1,...,k\} \to \{1,...n\}$. R appears as $R(x_{a(1)}, ..., x_{a(k)})$ in ϕ'. From now, completing the reduction is easy: define $R^{\mathcal{S}_2} = \{(1, s_1, ..., s_k) \mid (s_1, ..., s_k) \in R^{\mathcal{S}_1}\}$ and $D_x^{\mathcal{S}_2} = \{1\}$. ◀

▶ **Corollary 23 (induced query)**
Let $\phi \in$ NCQ. For every ϕ' such that $\mathcal{H}(\phi') = \mathcal{H}(\phi)[S]$ for some S, we have $?\mathcal{Q}(\phi) \succ ?\mathcal{Q}(\phi')$.

Proof. For every $x \in \mathcal{V}(\mathcal{H}(\phi)) \setminus S$, apply Lemma 40, together with transitivity (Lemma 37). We have proved that $?\mathcal{Q}(\phi)$ expresses *some* $?\mathcal{Q}(\phi')$, such that $\mathcal{H}(\phi') = \mathcal{H}(\phi)[S]$; apply Corollary 22 to prove equivalence of this last query with *all* queries having the same hypergraph; transitivity concludes. ◀

▶ ***Lemma 41 (hardness result)*** Under hypothesis that the problem of deciding the presence of a triangle in a graph on n vertices cannot be decided in $\mathcal{O}(n^2 \log n)$:

$$\forall \phi \in \text{NCQ} \quad ?\mathcal{Q}(\phi) \in \text{qLin} \Rightarrow \mathcal{A}_\beta(\mathcal{H}(\phi))$$

Proof. For the sake of contradiction, assume that, for some ϕ, we have $?\mathcal{Q}(\phi) \in$ qLin and $\neg\mathcal{A}_\beta(\mathcal{H}(\phi))$. This implies we can find $S \subseteq \mathcal{V}(\mathcal{H})$ and $h \subseteq \mathcal{H}(\phi)$ such that $h[S]$ is a chordless cycle. Then, by Corollary 23 and Corollary 22, $?\mathcal{Q}(\phi)$ expresses the following query of signature σ:

$$P = ?\mathcal{Q}\left(\exists x_1 \in D_1 ... \exists x_k \in D_k \ \neg R_k(x_k, x_1) \wedge \bigwedge_{1 \leq i < k} \neg R_i(x_i, x_{i+1})\right)$$

that, by Lemma 37, is also in qLin.

Now, let us associate to any graph $\mathcal{G} = (V, E)$ with $\text{Card}(V) = n$ a σ-structure defined as follows. For each R_i with $i > 3$, set:

$$R_i^{\mathcal{S}} = \{(i, j) \in V^2 \mid i \neq j\}$$

For each R_i with $i \leq 3$, set:

$$R_i^{\mathcal{S}} = \{(i, j) \in V^2 \mid (i, j) \notin E\}$$

Set $D_i^{\mathcal{S}} = V$. We have $|\mathcal{S}| = \mathcal{O}(n^2)$. If this query is in qLin, we can decide the presence of a triangle in $\mathcal{G}$ in time $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|) = \mathcal{O}(n^2 \log n)$. ◀

Concluding Remark

β-acyclic existential first-order queries have many qualities, they only lack one thing: to include α-acyclic CQ. This is to be addressed in a future paper.

Acknowledgments The author expresses his gratitude to Étienne Grandjean for careful, multiple readings of the successive versions of this paper, despite his busy schedule, until any doubtful or unclear point had disappeared. Without his assistance, this paper would still be a confused set of sketchy notes.

The author would also like to thank anonymous referees for their quite detailed reviews that greatly contributed to improve the quality of the paper.

Chapitre 6

Dichotomies pour les requêtes conjonctives signées.

> En logique, rien n'est accidentel : quand la chose se présente dans un état de choses, c'est que la possibilité de l'état de choses doit déjà être préjugée dans la chose.
>
> Ludwig Wittgenstein – *Tractatus logico-philosophicus* §2.012

Résumé On sait qu'une requête conjonctive est décidable en temps linéaire si et seulement si son hypergraphe $\mathcal{H}$ est α–acyclique. On sait aussi qu'une requête conjonctive est énumérable à délai constant après un précalcul en temps linéaire si et seulement si son hypergraphe $\mathcal{H}$ et l'hypergraphe $\mathcal{H} \cup \{L\}$ où L est l'ensemble de ses variables libres sont tous deux α–acycliques.

Si on considère des requêtes conjonctives signées, il faut distinguer l'hypergraphe $\mathcal{H}_+$ de la partie positive de la conjonction et l'hypergraphe $\mathcal{H}_-$ de la partie négative de la conjonction. On a donc besoin d'une notion d'acyclicité pour les hypergraphes où cohabitent deux types différents d'arêtes. C'est ce que l'on propose dans une première section.

On donne donc pour ces hypergraphes dit bicolores une triple caractérisation d'une notion d'acyclicité qui généralise simultanément la double caractérisation des hypergraphes « simples » α–acycliques, ainsi que la triple caractérisation des hypergraphes β–acycliques.

Grâce à ce résultat, nous prouvons dans une seconde section des dichotomies pour les requêtes conjonctives signées qui généralisent celles des requêtes conjonctives et de leurs cousines négatives.

Sommaire

1 Acyclicité des hypergraphes 106
1.1 Rappels 106
1.2 Des hypergraphes bicolores 107
1.3 Un hypergraphe bicolore acyclique a des feuilles. . . 110
2 Des requêtes 116
2.1 Décision 116
2.2 Énumération 119

1 Acyclicité des hypergraphes

On définit une notion d'acyclicité pour des hypergraphes à deux couleurs, c'est–à–dire où on distingue deux sortes d'arêtes, que l'on appelle respectivement rouges et noires. Cette notion se réduit à l'α–acyclicité quand toutes les arêtes sont rouges, et à la β–acyclicité quand elles sont toutes noires. Entre les deux, on se retrouve dans une situation intermédiaire : si l'hypergraphe est β–acyclique lorsqu'on ignore les couleurs, alors l'hypergraphe bicolore est acyclique ; si l'hypergraphe bicolore est acyclique, alors l'hypergraphe est α–acyclique lorsqu'on ignore les couleurs.

On prouve une triple caractérisation de cette notion d'acyclicité, qui admet pour cas particuliers la double caractérisation de l'α–acyclicité et la triple caractérisation de la β–acyclicité. Le point techniquement difficile consiste à généraliser le résultat de Andries E. Brouwer et Antoon W.J. Kolen [BK80] qui établit qu'un hypergraphe β–acyclique a un point d'imbrication (« *nest point* » dans leur article). On généralise le résultat en prouvant qu'un hypergraphe bicolore acyclique contient un « point d'imbrication généralisé ». Pour ce, on adapte assez lourdement leur preuve.

1.1 Rappels

Définition 50 (hypergraphe) On appelle *hypergraphe* $\mathcal{H}$ un ensemble d'ensembles non vides, appelés *arêtes*. On définit la *taille* d'un hypergraphe $|\mathcal{H}|$ comme la somme des cardinalités de ses arêtes. On définit l'ensemble des *sommets* de $\mathcal{H}$ ainsi : $\mathcal{V}(\mathcal{H}) = \bigcup_{e\in\mathcal{H}} e$. On note $\mathcal{H}[S]$ l'*hypergraphe induit de $\mathcal{H}$ sur S*, $\mathcal{H}\langle x\rangle$ le *support du sommet x dans $\mathcal{H}$*, et $\mathrm{M}(\mathcal{H})$ *l'hypergraphe minimisé* de $\mathcal{H}$, définis ainsi :

$$\begin{aligned}\mathcal{H}[S] &= \{e\cap S \mid e\in\mathcal{H}\}\setminus\{\emptyset\}\\ \mathcal{H}\langle x\rangle &= \{\,e\in\mathcal{H}\mid x\in e\}\\ \mathrm{M}(\mathcal{H}) &= \{e\in\mathcal{H}\mid \nexists f\in\mathcal{H}\ e\subset f\}\end{aligned}$$

On utilisera la notation $\mathcal{H}[\setminus S]$ comme raccourci pour $\mathcal{H}[\mathcal{V}(\mathcal{H})\setminus S]$.

On dit d'un hypergraphe $\mathcal{H}$ qu'il est *acyclique*, noté $\mathcal{A}_\alpha(\mathcal{H})$, lorsqu'il est α–acyclique.

Un *sommet isolé* d'un hypergraphe $\mathcal{H}$ est un sommet $x\in\mathcal{V}(\mathcal{H})$ tel que $\mathrm{Card}(\mathcal{H}\langle x\rangle) = 1$. On définit l'ensemble des *feuilles d'un hypergraphe* (plus précisément : *α-feuilles*) $\mathcal{H}$, noté $\mathcal{L}(\mathcal{H})$, comme l'ensemble des sommets isolés de $\mathrm{M}(\mathcal{H})$.

Définition 51 (Tetra, Cycle) On définit :

$$\begin{aligned}\mathrm{Cycle}(k) &= \{\{i,i+1\}\mid 1\le i<k\}\cup\{\{1,k\}\}\\ \mathrm{Tetra}(k) &= \{\{1,...,k\}\setminus\{x\}\mid x\in\{1,...,k\}\}\end{aligned}$$

En particulier, $\mathrm{Tetra}(3) = \{\{1,2\},\{2,3\},\{3,1\}\} = \mathrm{Cycle}(3)$.

On rappelle la triple équivalence proposée au chapitre 4, théorème 11.

▶ **Théorème 24 (caractérisations de l'acyclicité α)**

Soit $\mathcal{H}$ un hypergraphe. Les propositions suivantes sont équivalentes :

1. $\mathcal{H}$ est acyclique, c'est-à-dire conforme est sans cycle.
2. On ne peut pas trouver $S\subseteq\mathcal{V}(\mathcal{H})$ tel que $\mathrm{M}(\mathcal{H}[S])$ soit (au choix) :
   - ou bien isomorphe à $\mathrm{Cycle}(\mathrm{Card}(S))$ avec $\mathrm{Card}(S)\ge 3$,

- ou bien isomorphe à Tetra(Card (S)) avec Card $(S) \geq 4$.

3 Ou bien $\mathcal{H}$ est vide, ou bien $\mathcal{H}$ possède une feuille x et pour toute feuille x, $\mathcal{H}[\setminus\{x\}]$ est acyclique.

La caractérisation 1 ramène à la définition, la 2 est une caractérisation par exclusion d'une certaine forme de cycles, la 3 est une définition inductive par retrait de feuilles. ◀

On peut tirer de ce théorème les faits suivants, présentés ici comme des corollaires.

▶ ***Lemme 42*** Soit $\mathcal{H}$ un hypergraphe et $S \subseteq \mathcal{V}(\mathcal{H})$. On a :

$$\begin{array}{lcl} \mathcal{A}_\alpha(\mathcal{H}) & \Leftrightarrow & \mathcal{A}_\alpha(\mathrm{M}(\mathcal{H})) \\ \mathcal{A}_\alpha(\mathcal{H}) & \Rightarrow & \mathcal{A}_\alpha(\mathcal{H}[S]) \end{array}$$ ◀

Définition 52 (ordre d'élimination) Soit $\mathcal{H}$ un hypergraphe tel que $\mathcal{V}(\mathcal{H}) = \{x_1,...,x_n\}$. On dit que $\mathcal{H}$ admet l'*ordre d'élimination alpha inverse (OEIα)* $(x_1,...,x_n)$ si et seulement si :

- le sommet x_n est une (α-)feuille de $\mathcal{H}$, c'est–à–dire $x_n \in \mathcal{L}(\mathcal{H})$ et
- $\mathcal{H}[\setminus\{x_n\}]$ admet l'ordre d'élimination (alpha) inverse $(x_1,...,x_{n-1})$.

1.2 Des hypergraphes bicolores

Définition 53 (hypergraphe bicolore) On appelle *hypergraphe à deux couleurs* (ou encore *bicolore*) le couple $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ où $\mathcal{H}_1$ et $\mathcal{H}_2$ sont des hypergraphes, avec $\mathcal{V}(\mathcal{H}_2) \subseteq \mathcal{V}(\mathcal{H}_1)$. $\mathcal{H}_1$ est l'ensemble des arêtes dites *rouges* de $\mathcal{H}$, $\mathcal{H}_2$ est l'ensemble des arêtes dites *noires* de $\mathcal{H}$, et $\mathcal{H}_1 \cup \mathcal{H}_2$ [1] est l'ensemble des arêtes de $\mathcal{H}$.

On définit la *taille* de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$, notée $\left|\binom{\mathcal{H}_2}{\mathcal{H}_1}\right|$ comme $|\mathcal{H}_1| + |\mathcal{H}_2|$. On étend les notations définies pour les hypergraphes ainsi :

$$\begin{array}{lcl} \mathcal{V}\binom{\mathcal{H}_2}{\mathcal{H}_1} & = & \mathcal{V}(\mathcal{H}_1) \cup \mathcal{V}(\mathcal{H}_2) = \mathcal{V}(\mathcal{H}_1) \\ \binom{\mathcal{H}_2}{\mathcal{H}_1}[S] & = & \binom{\mathcal{H}_2[S]}{\mathcal{H}_1[S]} \\ \binom{\mathcal{H}_2}{\mathcal{H}_1}\langle x\rangle & = & \binom{\mathcal{H}_2\langle x\rangle}{\mathcal{H}_1\langle x\rangle} \\ \mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1} & = & \binom{\{e \in \mathcal{H}_2 \mid \nexists f \in \mathcal{H}_1\ e \subseteq f\}}{\{e \in \mathcal{H}_1 \mid \nexists f \in \mathcal{H}_1\ e \subset f\}} \\ \mathcal{A}\binom{\mathcal{H}_2}{\mathcal{H}_1} & \Leftrightarrow & \forall h \subseteq \mathcal{H}_2 \quad \mathcal{A}_\alpha(\mathcal{H}_1 \cup h) \end{array}$$

On dit que le sommet x est un *point d'imbrication* d'un hypergraphe bicolore $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ quand on a :

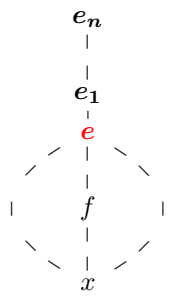
$$\forall e, f \in \mathcal{H}_1\langle x\rangle \cup \mathcal{H}_2\langle x\rangle \quad e \subseteq f \vee f \subseteq e$$

c'est-à-dire que l'ensemble des arêtes portant x dans $\mathcal{H}$ est totalement ordonné pour l'inclusion. On définit l'ensemble des *feuilles d'un hypergraphe bicolore* $\mathcal{H}$, noté $\mathcal{L}(\mathcal{H})$, comme l'ensemble des points d'imbrication de son hypergraphe minimisé $\mathrm{M}(\mathcal{H})$.

Remarque 34 (extension de l'acyclicité) Ces définitions étendent bien les notions d'alpha et de bêta–acyclicité.

L'opération de minimisation M (suppression des arêtes incluses) avait pour principale caractéristique de ne pas affecter l'acyclicité d'un hypergraphe $\mathcal{H}$ i.e. $\mathcal{A}_\alpha(\mathcal{H}) \Leftrightarrow \mathcal{A}_\alpha(\mathrm{M}(\mathcal{H}))$. Son extension aux hypergraphes bicolores préserve cette caractéristique, comme on le verra au lemme 43.

1 Cette union n'est pas nécessairement disjointe, ce qui explique l'utilisation de $\subseteq$ dans la définition de M ci–dessous.



e_i : n arêtes *noires*, $n \geq 0$.
e : une arête *rouge*.
f : des arêtes incluses dans e.

● **Figure 6.1 :** Le support d'une feuille généralisée x. Autant la partie supérieure $(e_1,...,e_n)$ que la partie inférieure (tout le reste) est optionnelle.

Par ailleurs, elle l'étend dans le sens où, si on note $\binom{\mathcal{H}'_2}{\mathcal{H}'_1} = \mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1}$, on a $\mathcal{H}'_1 = \mathrm{M}(\mathcal{H}_1)$.

La notion d'acyclicité définie est bien intermédiaire entre l'alpha et la bêta–acyclicité :

$$\mathcal{A}_\beta(\mathcal{H}_1 \cup \mathcal{H}_2) \Leftrightarrow \mathcal{A}\binom{\mathcal{H}_1 \cup \mathcal{H}_2}{\emptyset} \Rightarrow \mathcal{A}\binom{\mathcal{H}_2}{\mathcal{H}_1} \Rightarrow \mathcal{A}\binom{\emptyset}{\mathcal{H}_1 \cup \mathcal{H}_2} \Leftrightarrow \mathcal{A}_\alpha(\mathcal{H}_1 \cup \mathcal{H}_2)$$

La notion de feuille généralise à la fois les notions d'α–feuille et de β–feuille (ou : point d'imbrication [BK80]). En effet, une feuille x est un sommet dont le support $\mathcal{H}\langle x\rangle$ a la forme donnée figure 6.1. Il est facile de voir que si toutes les arêtes sont rouges, cela signifie que le support de x a une arête qui inclut toutes les autres, et que, par conséquent, notre sommet x est isolé dans $\mathrm{M}(\mathcal{H})$, ce qui est la définition même d'une α–feuille. Par ailleurs, si toutes les arêtes sont noires, cela signifie que le support du sommet x est une chaîne pour l'inclusion, ce qui est la définition d'un point d'imbrication, de fait la définition d'une β–feuille.

De plus, il est clair que cette configuration est maintenue quand on retire des arêtes noires, donc une feuille de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ est une feuille de $\binom{h}{\mathcal{H}_1}$ pour tout $h \subseteq \mathcal{H}_2$. On peut aussi noter qu'une feuille de $\binom{\mathcal{H}_2 \cup h}{\mathcal{H}_1}$ est aussi une feuille de $\binom{\mathcal{H}_2}{\mathcal{H}_1 \cup h}$. En conséquence de ces deux points, une feuille de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ est une α-feuille de $\mathcal{H}_1 \cup h$ pour tout $h \subseteq \mathcal{H}_2$.

On généralise le lemme 42.

▶ ***Lemme 43*** Soit $\mathcal{H}$ un hypergraphe bicolore. On a :

$$\begin{array}{lcl} & \mathcal{A}(\mathcal{H}) & \Leftrightarrow \quad \mathcal{A}(\mathrm{M}(\mathcal{H})) \\ \forall S \subseteq \mathcal{V}(\mathcal{H}) & \mathcal{A}(\mathcal{H}) & \Rightarrow \quad \mathcal{A}(\mathcal{H}[S]) \end{array}$$

Démonstration. Soit $\binom{\mathcal{H}'_2}{\mathcal{H}'_1} = \mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1}$. On a :

$$\begin{array}{lcll} \mathcal{A}\binom{\mathcal{H}_2}{\mathcal{H}_1} & \Leftrightarrow & \forall h \subseteq \mathcal{H}_2\ \mathcal{A}_\alpha(\mathcal{H}_1 \cup h) & \text{(définition)} \\ & \Leftrightarrow & \forall h \subseteq \mathcal{H}_2\ \mathcal{A}_\alpha(\mathrm{M}(\mathcal{H}_1 \cup h)) & \text{(lemme 42)} \\ & \Leftrightarrow & \forall h \subseteq \mathcal{H}'_2\ \mathcal{A}_\alpha(\mathrm{M}(\mathcal{H}'_1 \cup h)) & \text{(égalité 1)} \\ & \Leftrightarrow & \forall h \subseteq \mathcal{H}'_2\ \mathcal{A}_\alpha(\mathcal{H}'_1 \cup h) & \text{(lemme 42)} \\ & \Leftrightarrow & \mathcal{A}\big(\mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1}\big) & \text{(définition)} \end{array}$$

Soit $S \subseteq \mathcal{V}\binom{\mathcal{H}_2}{\mathcal{H}_1}$. On a :

$$\begin{array}{lcll} \mathcal{A}\binom{\mathcal{H}_2}{\mathcal{H}_1} & \Leftrightarrow & \forall h \subseteq \mathcal{H}_2\ \mathcal{A}_\alpha(\mathcal{H}_1 \cup h) & \text{(définition)} \\ & \Rightarrow & \forall h \subseteq \mathcal{H}_2\ \mathcal{A}_\alpha((\mathcal{H}_1 \cup h)[S]) & \text{(lemme 42)} \\ & \Leftrightarrow & \forall h \subseteq \mathcal{H}_2[S]\ \mathcal{A}_\alpha(\mathcal{H}_1[S] \cup h) & \text{(égalité 2)} \\ & \Leftrightarrow & \mathcal{A}\big(\binom{\mathcal{H}_2}{\mathcal{H}_1}[S]\big) & \text{(définition)} \end{array}$$

Les égalités référencées (1) et (2) sont les suivantes :

$$\{\mathrm{M}(\mathrm{M}(\mathcal{H}_1)\cup h)\mid h\subseteq\mathcal{H}_2'\}=\{\mathrm{M}(\mathcal{H}_1\cup h)\mid h\subseteq\mathcal{H}_2\}\qquad(1)$$

$$\{(\mathcal{H}_1[S]\cup h)\mid h\subseteq\mathcal{H}_2[S]\}=\{(\mathcal{H}_1\cup h)[S]\mid h\subseteq\mathcal{H}_2\}\qquad(2)$$

On laisse le lecteur se convaincre de leur validité. ◀

▶ **Théorème 25 (résultat principal)**

Soit $\mathcal{H}$ un hypergraphe bicolore. Les trois propriétés suivantes sont équivalentes :

1. $\mathcal{H}$ est acyclique.
2. On ne peut trouver un ensemble de sommets S, avec $\mathrm{Card}(S)\geq 3$ tel que, en posant $\binom{\mathcal{H}_2'}{\mathcal{H}_1'}=\mathrm{M}(\mathcal{H}[S])$:
   - ou bien $\mathcal{H}_1'$ est isomorphe à $\mathrm{Tetra}(\mathrm{Card}(S))$,
   - ou bien on peut trouver $h\in\mathcal{H}_2'$ tel que $h\cup\mathcal{H}_1'$ est isomorphe à $\mathrm{Cycle}(\mathrm{Card}(S))$.
3. Soit $\mathcal{H}$ est vide, soit $\mathcal{H}$ possède une feuille x et $\mathcal{H}[\smallsetminus\{x\}]$ est acyclique.

Démonstration (¬2 ⇒ ¬1). Si ¬(2), alors on peut trouver S tel que $\mathrm{M}(\mathcal{H}[S])$ n'est pas acyclique ; le lemme 43 conclut.

Démonstration (¬1 ⇒ ¬2). Considérons un hypergraphe bicolore non acyclique $\binom{\mathcal{H}_2}{\mathcal{H}_1}$. Prenons S tel que $\binom{\mathcal{H}_2}{\mathcal{H}_1}[S]$ n'est pas acyclique, on suppose S minimal. Il existe un certain $h\subseteq\mathcal{H}_2[S]$ tel que $\mathcal{H}'=\mathcal{H}_1[S]\cup h$ ne soit pas acyclique, on suppose h minimal. De par la caractérisation (2) du théorème 24, on sait qu'il existe S' tel que $\mathrm{M}(\mathcal{H}'[S'])$ est ou bien un cycle, ou bien une clique non conforme (détails juste après). Dans les deux cas, la minimalité de S nous impose $S=S'$. Si $\mathrm{M}(\mathcal{H}')$ est un cycle, c'est–à–dire un hypergraphe isomorphe à $\{\{x_i,x_{i+1}\}\mid 1\leq i\leq n\}\cup\{x_1,x_n\}$ avec $n\geq 3$, alors le résultat est prouvé.

Sinon, $\mathrm{M}(\mathcal{H}')$ est isomorphe à $\{S\smallsetminus\{x\}\mid x\in S\}$. Si h est vide, le résultat est prouvé. Sinon, prenons $e\in h$. Si l'arête e n'est pas de la forme $S\smallsetminus\{x\}$, alors $\mathrm{M}(\mathcal{H}_1[S]\cup h)=\mathrm{M}(\mathcal{H}_1[S]\cup(h\smallsetminus\{e\}))$, contredisant la minimalité de h. Si e est aussi présente dans $\mathcal{H}_1[S]$, on a encore la même contradiction. Par conséquent, $\mathrm{M}(\mathcal{H}_1[S]\cup(h\smallsetminus\{e\}))[\smallsetminus\{x\}]$ est de la forme $\{S'\smallsetminus\{y\}\mid y\in S'\}$ avec $S'=S\smallsetminus\{x\}$, ce qui contredit la minimalité de S.

Démonstration (3 ⇒ 1). Supposons que $\binom{\mathcal{H}_2}{\mathcal{H}_1}[\smallsetminus\{x\}]$ est acyclique. Alors chacun des hypergraphes $(\mathcal{H}_1\cup h)[\smallsetminus\{x\}]$ est acyclique. Supposons de plus que x est une feuille de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$. Alors cette feuille est une feuille de chacun des $\mathcal{H}_1\cup h$ pour $h\subseteq\mathcal{H}_2$, voir la remarque 34. De par ces deux faits, le théorème 24 nous dit que chacun des $\mathcal{H}_1\cup h$ pour $h\subseteq\mathcal{H}_2$ est acyclique, ce que l'on voulait.

Démonstration (1 ⇒ 3). Le lemme 43 nous dit que l'acyclicité est stable par induction, il ne « reste qu'à » montrer qu'un hypergraphe bicolore acyclique a des feuilles. C'est le résultat le plus dur, et l'objet du théorème 27. Se reporter à la sous–section suivante. ◀

Remarque 35 (test d'acyclicité) La troisième caractérisation du théorème 25 nous donne un algorithme polynomial pour tester l'acyclicité des hypergraphes bicolores. Pour espérer trouver un algorithme linéaire, il faudrait proposer un algorithme linéaire pour tester la β–acyclicité, puis généraliser simultanément celui-ci et l'algorithme linéaire de Tarjan et Yannakakis [TY84, TY85], qu'il s'agirait déjà de comprendre.

Définition 54 (ordre d'élimination) Soit $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ un hypergraphe bicolore avec $\mathcal{V}\binom{\mathcal{H}_2}{\mathcal{H}_1}=\{x_1,...,x_n\}$. On dit que $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ admet l'*ordre d'élimination inverse* (*OEI*) $(x_1,...,x_n)$ si et seulement si :

- le sommet x_n est une feuille de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$, c'est–à–dire $x_n \in \mathcal{L}(\mathcal{H})$,
- et $\binom{\mathcal{H}_2}{\mathcal{H}_1}[\searrow\{x_n\}]$ admet l'ordre d'élimination inverse $(x_1,...,x_{n-1})$.

Le lemme suivant généralise le lemme 13 sans s'appuyer sur lui : la preuve en est non constructive, cette fois, mais est nettement plus concise.

▶ ***Lemme 44 (ordre d'élimination)*** Un hypergraphe bicolore est acyclique si et seulement si il admet un ordre d'élimination inverse. Plus généralement, un OEI d'un hypergraphe bicolore acyclique $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ est un OEI (un ordre d'élimination inverse α, plus exactement) de chacun des hypergraphes simples $\mathcal{H}_1 \cup h$ pour $h \subseteq \mathcal{H}_2$.

Soit $\binom{\mathcal{H}_2 \cup \{\{x_1,...,x_n\}\}}{\mathcal{H}_1}$ un hypergraphe bicolore. S'il est acyclique, alors il admet un ordre d'élimination inverse de la forme $(x_1,...,x_n,y_1,...,y_m)$.

Démonstration. Le premier point est immédiat de par la caractérisation inductive du théorème 25.

On prouve par récurrence totale qu'aucune arête (sauf l'arête complète qui contient tous les sommets) ne saurait inclure toutes les feuilles d'un hypergraphe bicolore acyclique. On suppose que c'est le cas pour tout $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ de taille inférieure à n, et on considère $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ un hypergraphe bicolore acyclique de taille $n+1$. Si une arête contient tous les sommets, on peut la retirer sans changer ni l'acyclicité ni les feuilles et la récurrence conclut. On suppose donc le contraire. De même, si $\mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1} \neq \binom{\mathcal{H}_2}{\mathcal{H}_1}$, la récurrence conclut également ; désormais $\mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$.

Supposons qu'une arête $e \in \mathcal{H}_2$, que l'on suppose maximale pour l'inclusion, contienne toutes les feuilles. Soit x une feuille. Ce sommet n'a pas d'autres voisins que les sommets de e, sinon ou bien e serait incluse dans une autre arête, contredisant sa maximalité, ou alors x appartiendrait à deux arêtes incomparables et ne serait donc pas une feuille. Conséquemment, $\binom{\mathcal{H}_2}{\mathcal{H}_1}[\searrow\{x\}]$ a uniquement des feuilles dans $e \searrow \{x\}$.

Par récurrence, $e \searrow \{x\}$ contient donc tous les sommets de cet hypergraphe induit, e contient donc tous les sommets de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$, contradiction. ◀

▶ **Corollaire 26 (généralisation du théorème 13)**

Pour tout hypergraphe bicolore acyclique, on peut choisir une arête e et trouver un ordre d'élimination qui élimine tous les sommets hors de e avant d'éliminer les sommets de (dans) l'arête e.

Il s'ensuit qu'un hypergraphe bicolore $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ admet un ordre d'élimination qui élimine les sommets hors d'un ensemble S donné avant ceux dans S si et seulement si $\binom{\mathcal{H}_2 \cup \{S\}}{\mathcal{H}_1}$ est acyclique. ◀

1.3 Un hypergraphe bicolore acyclique a des feuilles.

Nous prouvons ici une généralisation du résultat fondamental de Andries Brouwer et Antoon J. Kolen [BK80], qui stipule qu'un hypergraphe β-acyclique, c'est–à–dire dont tous les sous–ensembles sont α–acycliques, a des points d'imbrication.

Nous suivons essentiellement la même preuve. Leur preuve consiste à prouver par récurrence non pas l'existence d'un point d'imbrication, mais de deux (si on suppose qu'il y a au moins deux sommets). Ce renforcement de l'hypothèse de récurrence est un élément clef.

▶ ***Lemme 45*** Soit $\mathcal{H}$ un hypergraphe bicolore, et x une de ses feuilles. $\mathcal{H}$ est acyclique si et seulement si $\mathcal{H}[\searrow\{x\}]$ l'est.

Démonstration. Le sens « si » est l'implication $3 \Rightarrow 1$ (qui a déjà été prouvée...) du résultat principal, le théorème 25. Le sens « seulement si » est un cas particulier du lemme 43. ◀

▶ **Théorème 27 (feuilles)**

Un hypergraphe bicolore acyclique non vide a des feuilles.

Démonstration. Il est clair qu'un hypergraphe bicolore à un seul sommet a une feuille : son unique sommet. Nous allons montrer, par récurrence, que tout hypergraphe bicolore acyclique à au moins deux sommets possède deux feuilles. Vérifier qu'un hypergraphe à deux sommets admet bien ses sommets comme feuilles est anecdotique.

Supposons que tout hypergraphe bicolore acyclique, à au moins deux sommets, et de taille au plus n admet au moins deux feuilles. Considérons $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ un hypergraphe bicolore acyclique à au moins trois sommets et de taille $n+1$.

On va maintenant exclure des cas pour lesquels il est facile d'appliquer l'hypothèse de récurrence ; ce faisant, on normalise l'hypergraphe bicolore sur trois plans. Premièrement, si $\mathrm{M}(\mathcal{H}) \neq \mathcal{H}$, l'hypothèse de récurrence appliquée à l'hypergraphe bicolore acyclique $\mathrm{M}(\mathcal{H})$ de taille strictement plus petite que celle de $\mathcal{H}$ établit le résultat pour $\mathcal{H}$. On suppose donc désormais que $\mathrm{M}(\mathcal{H}) = \mathcal{H}$, c'est–à–dire que les arêtes de $\mathcal{H}_1$ sont minimales pour l'inclusion parmi les arêtes de $\mathcal{H}_1 \cup \mathcal{H}_2$; les feuilles de $\mathcal{H}$ seront alors exactement ses points d'imbrication. Deuxièmement, si deux sommets x et y sont indistinguables dans $\mathcal{H}$, c'est–à–dire $\mathcal{H}\langle x\rangle = \mathcal{H}\langle y\rangle$, l'hypothèse de récurrence appliquée à l'hypergraphe bicolore acyclique $\mathcal{H}[\setminus\{x\}]$, de taille strictement plus petite que celle de $\mathcal{H}$, établit le résultat pour $\mathcal{H}$. On suppose donc désormais que tous les sommets sont distinguables dans $\mathcal{H}$. Troisièmement, si on peut trouver deux arêtes incomparables (pour l'inclusion) e et f, et une arête $g \in \mathcal{H}_2$ telle que $g \subseteq e \cap f$, $\mathcal{H}' = \binom{\mathcal{H}_2 \setminus \{g\}}{\mathcal{H}_1}$ a les mêmes feuilles que $\mathcal{H}$:

- Puisque f et g sont présentes, aucune arête rouge de $\mathcal{H}$ (donc de $\mathcal{H}_1$) ne les inclut. Il s'ensuit que, pour tout $u \in g$, u n'est une feuille ni de $\mathcal{H}$ ni de $\mathcal{H}'$.
- Pour $u \notin g$, $\mathcal{H}\langle u\rangle = \mathcal{H}'\langle u\rangle$, donc u est une feuille de $\mathcal{H}$ si et seulement si u est une feuille de $\mathcal{H}'$.

De plus, $\mathcal{H}'$ est trivialement acyclique ; la récurrence s'applique alors : $\mathcal{H}'$ a deux feuilles qui sont des feuilles de $\mathcal{H}$. On suppose donc qu'on ne trouve aucune arête noire (dans $\mathcal{H}_2$) incluse dans une intersection stricte, c'est–à–dire une intersection d'arêtes incomparables pour l'inclusion.

On va maintenant établir que l'hypergraphe bicolore $\mathcal{H}$ a deux feuilles — ou, de manière équivalente, deux points d'imbrication (puisque $\mathrm{M}(\mathcal{H}) = \mathcal{H}$, les feuilles de $\mathcal{H}$ sont exactement ses points d'imbrication). C'est une preuve par l'absurde décomposée en deux parties :

1. On va d'abord d'abord envisager que $\mathcal{H}$ n'ait aucun point d'imbrication, une contradiction s'ensuivra.
2. On va ensuite supposer que le point d'imbrication est unique, ce qui amènera encore une contradiction.

En conséquence de ces deux faits, $\mathcal{H}$ a au moins deux points d'imbrication.

Démonstration (1). Supposons que $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ n'ait aucun point d'imbrication. Puisque $\mathcal{H}_1 \cup \mathcal{H}_2$ est acyclique, on peut trouver un sommet x tel que $\mathcal{H}\langle x\rangle$ a une arête maximale pour l'inclusion (qui inclut toutes les autres) qu'on notera e_x. Si $e_x \in \mathcal{H}_1$, alors x est une feuille de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$,

contradiction ; e_x est donc une arête noire (i.e. appartient à $\mathcal{H}_2$). A l'instar de la preuve de Brouwer et Kolen [BK80], on introduit l'ensemble de sommets I défini ainsi :

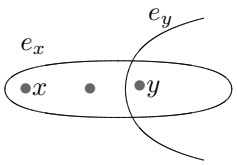
$$I = \{u \mid \forall e \in \mathcal{H}\langle u\rangle \; e \subseteq e_x\}$$

Autrement dit, I est l'ensemble des sommets de e_x qui sont des feuilles de $\mathcal{H}_1 \cup \mathcal{H}_2$ et pour lesquels e_x est l'arête maximale qui les contient.

Supposons $I = e_x$. Dans ce cas, toute arête qui intersecte I, par définition de I, est contenue dans e_x. On a montré que toute arête de $\mathcal{H}$ est soit incluse dans e_x, soit disjointe de e_x. Par hypothèse de récurrence, l'hypergraphe bicolore acyclique $\binom{\mathcal{H}_2 \smallsetminus \{e_x\}}{\mathcal{H}_1}$ possède une feuille y (et même deux). Comme $\mathcal{H}$, il est minimisé, donc ses feuilles sont des points d'imbrication. Comme e_x n'intersecte strictement aucune arête, les deux hypergraphes ont exactement les mêmes points d'imbrication, en particulier y est un point d'imbrication de $\mathcal{H}$, contradiction.

On a donc prouvé que $I \subsetneq e_x$. On considère maintenant l'hypergraphe bicolore acyclique $\mathcal{H}' = \binom{\mathcal{H}'_2}{\mathcal{H}'_1} = \binom{\mathcal{H}_2}{\mathcal{H}_1}[\smallsetminus I]$. Toujours par hypothèse de récurrence, il a une feuille y. Puisque, pour tout sommet $u \notin e_x$, $\mathcal{H}\langle u\rangle = \mathcal{H}'\langle u\rangle$ (vu qu'aucune arête de $\mathcal{H}$ portant u n'intersecte I), on en déduit $y \in e_x \smallsetminus I$, sans quoi y serait une feuille de $\mathcal{H}$.

On nomme e_y l'arête maximale (qui existe puisque y est une feuille) portant y dans $\mathcal{H}[\smallsetminus I]$. L'ensemble $e_x \smallsetminus I$ est une arête de cet hypergraphe, qui contient y, donc, de par la définition de e_y, $e_x \smallsetminus I \subseteq e_y$. Comme y n'appartient pas à I, y a un voisin t hors de e_x, qui est donc contenu dans e_y, donc $e_x \smallsetminus I \subset e_y$. On appelle e'_y une arête de $\mathcal{H}$ telle que $e'_y \smallsetminus I = e_y$. Si un sommet de I était contenu dans e'_y, alors il serait voisin de t, ce qui contredirait son appartenance à I. Par conséquent, $e'_y = e'_y \smallsetminus I = e_y$. En conséquence de quoi e_y est en fait une arête de $\mathcal{H}$, qui inclut strictement $e_x \smallsetminus I$. Finalement, e_x et e_y sont des arêtes incomparables de $\mathcal{H}$, avec $I = e_x \smallsetminus e_y$. On est bien dans cette situation :



On distingue deux cas : soit I est un singleton, soit pas.

- Dans le cas où $I = \{x\}$, et puisque $\mathcal{H}$ ne contient pas de point d'imbrication, x n'en est pas un, et donc x est contenu dans deux arêtes a et b incomparables pour l'inclusion. De par la définition de x, les arêtes a et b sont contenues dans e_x. Conséquemment, $a \cup b \smallsetminus \{x\} \subseteq e_x \smallsetminus I \subset e_y$. On en déduit que deux sommets de $a \cup b$ sont toujours voisins dans $\mathcal{H}$ par une arête parmi e_y, a, b. Par conséquent, $a \cup b$ est une clique de $\mathcal{H}_1 \cup \{a, b, e_y\}$. Ce dernier hypergraphe est acyclique par construction : il s'agit de toutes les arêtes de $\mathcal{H}_1$ et de certaines arêtes de $\mathcal{H}_2$. Comme il est acyclique donc conforme, une arête doit contenir la clique $a \cup b$. Cette arête ne peut être ni e_y, ni a ni b, c'est donc une arête de $\mathcal{H}_1$. Cette arête inclut a, ce qui contredit $\mathrm{M}(\mathcal{H}) = \mathcal{H}$.
- Dans le cas où $I \neq \{x\}$, on pose $\mathcal{H}' = \binom{\mathcal{H}'_2}{\mathcal{H}'_1} = \binom{\mathcal{H}_2}{\mathcal{H}_1}[\smallsetminus (I \smallsetminus \{x\})]$, qui est donc plus petit que $\mathcal{H}$. Cet hypergraphe bicolore a deux feuilles par l'hypothèse de récurrence, dont au moins une n'est pas x, on l'appelle z. $e_x \smallsetminus I \cup \{x\}$ et e_y sont deux arêtes incomparables de $\mathcal{H}'$. Prenons un sommet $u \in e_y \smallsetminus e_x$. Par définition de x, x n'est pas voisin de u. Il s'ensuit qu'aucune arête n'inclut à la fois les arêtes

incomparables $e_x \smallsetminus I \cup \{x\}$ et e_y. On rappelle que l'intersection de ces deux arêtes est $e_x \smallsetminus I$, et comme on vient de le montrer, il n'y a pas de feuille dans cet ensemble. En particulier, $z \notin e_x \smallsetminus I$, et comme $z \neq x$, $z \notin e_x$.

Par définition de I, z n'a, dans $\mathcal{H}$, aucun voisin dans I. Il s'ensuit que $\mathcal{H}\langle z\rangle = \mathcal{H}'\langle z\rangle$. Par conséquent, z est une feuille de $\mathcal{H}$, contradiction.

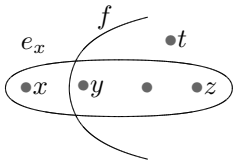
Ceci conclut la preuve par l'absurde. On a établi que $\mathcal{H}$ possède au moins un point d'imbrication.

Démonstration (2). Il reste à montrer, toujours par l'absurde, que $\mathcal{H}$ a au moins deux points d'imbrication. Supposons donc que $\mathcal{H}$ possède un unique point d'imbrication x. L'idée de la preuve est que la suppression d'une feuille ne peut en créer qu'une seule nouvelle. Toujours par l'hypothèse de récurrence, $\mathcal{H}[\smallsetminus x]$ a deux feuilles, que l'on appelle y et z. On va montrer que ces deux feuilles sont en fait « la même », c'est–à–dire deux sommets indistinguables.

Comme précédemment, on va tout d'abord clarifier la situation en montrant essentiellement que la prochaine figure la décrit honnêtement. Supposons que y soit une feuille de $\mathcal{H}[\smallsetminus\{x\}]$ mais pas un point d'imbrication. Dans ce cas, deux arêtes incomparables a et b portent y et sont contenues dans une arête rouge c. Si a était une arête de $\mathcal{H}$, elle aurait été supprimée, donc $a \cup \{x\}$ et $b \cup \{x\}$ sont deux arêtes incomparables de $\mathcal{H}$, donc x n'est pas un point d'imbrication, contradiction. Il s'ensuit que y, et donc z, par symétrie, sont des points d'imbrication de $\mathcal{H}[\smallsetminus\{x\}]$.

Supposons que x et y ne soient pas voisins dans $\mathcal{H}$. Alors $\mathcal{H}\langle y\rangle = \mathcal{H}[\smallsetminus\{x\}]\langle y\rangle$; par conséquent, y est aussi un point d'imbrication de $\mathcal{H}$, contradiction. Donc x et y sont voisins dans $\mathcal{H}$, ainsi que x et z. Il existe donc nécessairement deux arêtes minimales e_y et e_z telles que $\{x, y\} \subseteq e_y$ et $\{x, z\} \subseteq e_z$. Comme toutes deux portent x, elles sont comparables, l'une des deux inclut donc $\{x, y, z\}$, mettons[2] e_z. On la nomme aussi e_x, et c'est la plus petite arête qui contienne à la fois x, y, et z.

Comme z n'est pas un point d'imbrication de $\mathcal{H}$, on peut trouver deux arêtes a et b dans $\mathcal{H}$ qui portent z et sont incomparables. Si a et b portent x, x n'est pas un point d'imbrication, contradiction. Au moins l'une ne porte pas x, par exemple a. Supposons que b ne porte pas non plus x. Alors a et b sont des arêtes incomparables de $\mathcal{H}[\smallsetminus\{x\}]$ qui portent z, donc z n'est pas un point d'imbrication de $\mathcal{H}[\smallsetminus\{x\}]$, contradiction. Donc a porte x et b ne porte pas x. Puisque z est un point d'imbrication de $\mathcal{H}[\smallsetminus\{x\}]$, les trois arêtes $e_x \smallsetminus \{x\}$, $a \smallsetminus \{x\}$ et b sont comparables. Si $b \subseteq a \smallsetminus \{x\}$, alors a et b sont comparables, contradiction. Donc $a \smallsetminus \{x\} \subset b$. Puisque e_x est défini comme la plus petite arête de $\mathcal{H}$ portant x et z, on a $e_x \subseteq a$ et donc finalement $e_x \smallsetminus \{x\} \subseteq a \smallsetminus \{x\} \subset b$. On nomme f cette arête b qui ne porte pas x et est incomparable avec e_x, i.e. elle contient un certain t non contenu dans e_x. Nous sommes donc bien dans la situation suivante :



2 On peut le supposer par symétrie. Ce faisant, on rompt justement ladite symétrie. Ce n'est pas grave, on prouvera plus loin qu'aucune arête portant x n'est strictement incluse dans e_z, il s'ensuivra que $e_y = e_z$, ce qui restaurera la situation de symétrie.

Bien noter que ni e_x ni f ne sont incluses dans une arête rouge, ce qui contredirait $\mathrm{M}(\mathcal{H}) = \mathcal{H}$. Nous allons maintenant chercher à « supprimer » autant d'arêtes portant y et z que possible, jusqu'à montrer que seules les arêtes incluant e_x les contiennent simultanément, ce qui contredira la distinguabilité supposée de y et z.

Dans un premier temps, on va montrer qu'il n'y a dans $\mathcal{H}$ aucune arête incluse dans e_x qui porte x. Supposons donc le contraire, c'est–à–dire qu'une arête $g \subset e_x$ inclue x. On définit un second hypergraphe bicolore $\mathcal{H}'$ obtenu à partir de $\mathcal{H}$ en remplaçant l'arête g par $g \smallsetminus \{x\}$ de même couleur que g. Plus formellement, et plus longuement :

$$\mathcal{H}' = \begin{pmatrix} \mathcal{H}'_2 \\ \mathcal{H}'_1 \end{pmatrix} = \begin{cases} \begin{pmatrix} \mathcal{H}_2 \smallsetminus \{g\} \cup \{g \smallsetminus \{x\}\} \\ \mathcal{H}_1 \end{pmatrix} & \text{si } g \text{ est noire.} \\ \begin{pmatrix} \mathcal{H}_2 \\ \mathcal{H}_1 \smallsetminus \{g\} \cup \{g \smallsetminus \{x\}\} \end{pmatrix} & \text{si } g \text{ est rouge.} \end{cases}$$

Dans un premier temps, on va montrer que $\mathcal{H}'$ aussi est acyclique, l'hypothèse de récurrence nous indiquera donc qu'il a deux feuilles, puis on prouvera que $\mathcal{H}$ et $\mathcal{H}'$ ont les mêmes feuilles, ce qui induira que $\mathcal{H}$ a deux feuilles, d'où la contradiction.

On va montrer par la contraposée que si $\mathcal{H}$ est acyclique, alors l'hypergraphe $\mathcal{H}'$ lui aussi est acyclique. Supposons donc que $\mathcal{H}'$ est non acyclique. Alors il existe un ensemble de sommets $S \subseteq \mathcal{V}(\mathcal{H}') = \mathcal{V}(\mathcal{H})$ tel que $\mathcal{H}'[S]$ ne soit pas acyclique, et que l'on suppose minimal pour cette propriété. Si $x \notin S$, alors $\mathcal{H}[S] = \mathcal{H}'[S]$ qui n'est pas acyclique, donc $\mathcal{H}$ n'est pas acyclique, contradiction ; on a donc $x \in S$. Comme x est un point d'imbrication de $\mathcal{H}'$, c'est aussi un point d'imbrication de $\mathcal{H}'[S]$, donc une feuille. Comme $\mathcal{H}'[S]$ n'est pas acyclique, le lemme 45 nous indique que $\mathcal{H}'[S][\smallsetminus\{x\}] = \mathcal{H}'[S \smallsetminus \{x\}]$ est non acyclique, ce qui est contradictoire avec la minimalité supposée de S. On a montré que $\mathcal{H}'$ est acyclique et, comme $|\mathcal{H}'| < |\mathcal{H}|$, par l'hypothèse de récurrence, $\mathcal{H}'$ a deux feuilles, dont l'une n'est pas x, et que l'on nomme x'. On montre alors que x' est une feuille de $\mathcal{H}$:

- Si $x' \in g \subseteq e_x \smallsetminus \{x\}$, alors x' est contenu dans e_x et f, qui sont incomparables, donc incluses dans une arête rouge e de $\mathcal{H}'$. Comme $g \smallsetminus \{x\}$ n'inclut pas e_x, $e \in \mathcal{H}'_1 \smallsetminus \{e_x\} \subset \mathcal{H}_1$. Par conséquent, une arête de $\mathcal{H}_1$ inclut e_x et f, ce qui contredit $\mathrm{M}(\mathcal{H}) = \mathcal{H}$.
- Si $x' \notin g$, $\mathcal{H}\langle x \rangle = \mathcal{H}'\langle x \rangle$ donc x' est aussi une feuille de $\mathcal{H}$.

Par conséquent $\mathcal{H}$ a deux feuilles, ce qui contredit l'unicité de x. Il n'y a donc pas d'arête plus petite que e_x qui contienne x.[3] On sait donc qu'une arête contenant x inclut nécessairement e_x, et donc contient simultanément y et z : une arête contenant x ne permet donc pas de distinguer les sommets y et z.

On peut maintenant réellement entreprendre de se débarrasser d'éventuelles arêtes permettant de distinguer y et z, les deux points d'imbrication de $\mathcal{H}[\smallsetminus\{x\}]$, dont on rappelle qu'ils jouent (à nouveau) des rôles symétriques. Supposons donc qu'on puisse trouver une arête g qui contienne y mais pas z.

Supposons que g n'est pas incluse dans $e_x \smallsetminus \{x\}$. Alors elle contient un élément hors de cette arête. Par le point précédent, si g contient x, alors elle inclut e_x, et par conséquent elle contient z, contradiction. L'élément extérieur n'est donc pas x, il s'agit d'un élément $t' \notin e_x$. Alors g et $e_x \smallsetminus \{x\}$ sont deux arêtes incomparables de $\mathcal{H}[\smallsetminus\{x\}]$, donc y n'est pas

3 Il s'ensuit que $e_y = e_z$, on a donc restauré la situation de symétrie entre y et z.

un point d'imbrication de cet hypergraphe, contradiction. L'arête g est donc incluse dans $e_x \smallsetminus \{x\}$.

On rappelle que $e_x \smallsetminus \{x\} = e_x \cap f$. Puisque l'arête g est incluse dans $e_x \cap f$, elle est donc rouge.[4] Comme précédemment, on pose $\mathcal{H}' = \binom{\mathcal{H}_2}{\mathcal{H}'_1}$ avec :

$$\mathcal{H}'_1 = \mathcal{H}_1 \smallsetminus \{g\} \cup \{g \smallsetminus \{y\}\}$$

Autrement dit, $\mathcal{H}'$ est obtenu à partir de $\mathcal{H}$ en remplaçant l'arête rouge g par une nouvelle arête rouge $g \smallsetminus \{y\}$.

La contradiction s'obtient exactement comme précédemment : on va prouver l'acyclicité de $\mathcal{H}'$ (de la même manière), et le fait que ses feuilles soient des feuilles de $\mathcal{H}$ (même preuve), il s'ensuivra une contradiction de l'unicité de x.

Comme $\mathcal{H}\langle x\rangle = \mathcal{H}'\langle x\rangle$, x est également un point d'imbrication de $\mathcal{H}'$, donc une feuille, donc de par le lemme 45, $\mathcal{H}'[\smallsetminus\{x\}]$ est acyclique si et seulement $\mathcal{H}'$ l'est. Il nous suffit donc de montrer que $\mathcal{H}'[\smallsetminus\{x\}]$ est acyclique, sachant que $\mathcal{H}[\smallsetminus\{x\}]$ l'est ; on considère donc essentiellement ces deux hypergraphes. De plus, comme d'une part y est un point d'imbrication de $\mathcal{H}[\smallsetminus\{x\}]$ et qu'en conséquence les arêtes de $\mathcal{H}_1 \cup \mathcal{H}_2[\smallsetminus\{x\}]\langle y\rangle$ forment une chaîne pour l'inclusion, et que, d'autre part :

$$(\mathcal{H}'_1 \cup \mathcal{H}_2)[\smallsetminus\{x\}]\langle y\rangle \subset (\mathcal{H}_1 \cup \mathcal{H}_2)[\smallsetminus\{x\}]\langle y\rangle$$

y est donc un point d'imbrication de $\mathcal{H}'[\smallsetminus\{x\}]$.

On va à nouveau prouver l'acyclicité de cet hypergraphe par l'absurde : supposons donc que $\mathcal{H}'[\smallsetminus\{x\}]$ n'est pas acyclique. Il s'ensuit que l'on peut trouver S tel que $\mathcal{H}'[\smallsetminus\{x\}][S]$ est non acyclique, avec S minimal pour cette propriété. Supposons que y n'est pas dans S. Alors $g \cap S = (g \smallsetminus \{y\}) \cap S$ donc $\mathcal{H}'[\smallsetminus\{x\}][S] = \mathcal{H}[\smallsetminus\{x\}][S]$ qui est acyclique, contradiction ; y est donc dans S. Comme y est un point d'imbrication de $\mathcal{H}'[\smallsetminus\{x\}]$, c'est à plus forte raison un point d'imbrication de $\mathcal{H}'[\smallsetminus\{x\}][S]$. De par le lemme 45, $\mathcal{H}'[\smallsetminus\{x\}][S \smallsetminus \{y\}]$ est donc non acyclique, ce qui contredit la minimalité supposée de S.

On a ainsi montré que $\mathcal{H}'[\smallsetminus\{x\}]$ est acyclique, et que, par conséquent, $\mathcal{H}'$ l'est aussi. Comme $|\mathcal{H}'| < |\mathcal{H}|$, l'hypothèse de récurrence nous indique que $\mathcal{H}'$ a deux feuilles, dont une n'est pas x, que l'on appelle x'. Il reste à montrer que x' est nécessairement une feuille de $\mathcal{H}$.

- Supposons $x' \in g$. Alors $x' \in g \smallsetminus \{x\} \subset e_x \smallsetminus \{x\}$. Donc x' est contenu dans les deux arêtes incomparables e_x et f. Donc il existe une arête rouge $e \in \mathcal{H}'_1$ incluant ces deux arêtes. Comme $g \smallsetminus \{x\}$ n'inclut ni f ni e_x, $e \in \mathcal{H}'_1 \smallsetminus \{g \smallsetminus \{x\}\} = \mathcal{H}_1$, il s'ensuit qu'une arête de $\mathcal{H}_1$ inclut f, donc $\mathrm{M}(\mathcal{H}) \neq \mathcal{H}$, contradiction.
- Sinon, $x' \notin g$. Par conséquent, $\mathcal{H}'\langle x'\rangle = \mathcal{H}\langle x'\rangle$, donc x' est aussi une feuille de $\mathcal{H}$, contradiction.

On ne peut donc trouver d'arête g qui contienne y sans contenir z, ni le contraire. Il s'ensuit que y et z sont portés par les mêmes arêtes, c'est-à-dire sont indistinguables, contradiction.[5] Le point d'imbrication ne saurait donc être unique, ce qui conclut la preuve par récurrence du théorème 27, et donc du résultat principal, le théorème 25. ◀

4 Voir le troisième point de la normalisation, au tout début de la preuve.

5 Voir le deuxième point de la normalisation, au tout début de la preuve.

2 Des requêtes

On généralise très simplement le constat qu'on avait fait pour les requêtes conjonctives et les requêtes conjonctives négatives.

▶ ***Lemme 46*** Soit $\phi = (x_1,...,x_n)\exists x_{n+1}...\exists x_k\ \psi^+\wedge\psi^-$ une requête conjonctive signée non triviale, c'est–à–dire qui ne contient pas un atome et son opposé, où ψ^+ est une conjonction d'atomes positifs et ψ^- est une conjonction d'atomes négatifs. Alors la complexité de $\mathcal{Q}(\phi)$ ne dépend que de $\{x_1,...,x_n\}$, $\mathcal{H}(\psi^+)$ et $\mathcal{H}(\psi^-)$.

Démonstration. Il suffit de reprendre exactement les mêmes arguments que ceux des résultats similaires précédents. Le seul cas qu'il reste à envisager après cela est le cas trivial où ψ^+ et ψ^- contiennent un atome exactement identique, c'est–à–dire composé du même symbole de relation, qui porte sur le même n-uplet de variables. Ce cas a été exclu d'emblée par l'énoncé du lemme. ◀

2.1 Décision

2.1.1 Facilité

Pour montrer la facilité de la décision, le meilleur algorithme à ϕ fixée que l'on ait trouvé repose sur le compte, et est linéaire.

On va voir que l'on peut compter les solutions d'une requête conjonctive signée en temps linéaire, i.e. $\mathcal{O}_\phi(|\mathcal{S}|)$, cependant la complexité combinée n'est pas jolie, jolie : on a une dépendance exponentielle en ϕ.

▶ **Théorème 28 (complexité du compte)**
Une requête $\#\mathcal{Q}(\phi)$, où ϕ est sans quantificateurs, et dont l'hypergraphe bicolore $\binom{\mathcal{H}_-(\phi)}{\mathcal{H}_+(\phi)}$ est acyclique se calcule en temps :

$$\mathcal{O}\left(m2^{m_-}|\mathcal{S}|\right)$$

où m (resp. m_-) est le nombre de relations (resp. de relations niées) dans ϕ, i.e. $m = \mathrm{Card}\left(\mathcal{H}_+(\phi)\right) + \mathrm{Card}\left(\mathcal{H}_-(\phi)\right)$ (resp $.m_- = \mathrm{Card}\left(\mathcal{H}_-(\phi)\right)$).

Démonstration. On a :

$$\#\mathcal{Q}\left((...)\bigwedge_i R_i \wedge \bigwedge_{1\leq j\leq m_-} \neg S_j\right) = \sum_{p\subseteq\{1,..,m_-\}} (-1)^{\mathrm{Card}(p)} \#\mathcal{Q}\left((...)\bigwedge_i R_i \wedge \bigwedge_{j\in p} S_j\right)$$

Cela se déduit de la formule ensembliste suivante :

$$\mathrm{Card}\left(E \cap \bigcap_{1\leq j\leq n} \overline{F_j}\right) = \sum_{p\subseteq\{1,..,n\}} (-1)^{\mathrm{Card}(p)} \mathrm{Card}\left(E \cap \bigcap_{j\in p} F_j\right)$$

établie par une récurrence facile sur n.

On s'est ramené au compte de 2^{m_-} requêtes conjonctives α-acycliques (par définition de l'acyclicité bicolore), qui nécessitent chacune un temps $\mathcal{O}(|\mathcal{S}|)$, donc on calcule les termes de la somme en temps $\mathcal{O}\big(2^m|\mathcal{S}|\big)$.

Le calcul de la somme consiste à additionner ou soustraire 2^{m_-} éléments de taille bornée par $\mathcal{O}(m\log|\mathcal{S}|)$. On compte séparément les retenues : d'un côté on additionne sans retenue les 2^m éléments, ce qui coûte un temps $\mathcal{O}(m2^{m_-}\log|\mathcal{S}|)$, de l'autre on additionne un maximum de 2^{m_--1} bits de retenue. On les additionne à travers un arbre binaire équilibré, qui effectue donc $2^{m_-}-1$ sommes unitaires, pour un coût total de 2^{m_-}, négligeable donc.

On a bien une complexité en $\mathcal{O}\big(m2^{m_-}|\mathcal{S}|\big)$. ◀

On peut faire le constat que cet algorithme de décision est un algorithme de décision dans un sens assez strict : même s'il permet de compter les solutions, en revanche il ne permet pas même de trouver *une* solution ; ce n'est pas un algorithme de recherche ! Nous verrons que pour obtenir une solution, nous ne proposons rien d'autre que de les énumérer toutes.

2.1.2 Difficulté

On va définir les hypothèses de complexité irréductibles sur lesquelles on s'appuie pour prouver le résultat de difficulté : ces hypothèses seront suffisantes, mais aussi nécessaires à la dichotomie pour la décision.

Définition 55 (hypothèse $\mathrm{H}_d(k)$) On définit les problèmes de décision paramétrés $\mathrm{P}(k)$ portant sur un hypergraphe $\mathcal{H}$, où $\overline{\mathcal{H}}$ désigne le complémentaire de $\mathcal{H}$ i.e. $\overline{\mathcal{H}} = \{e \subseteq \mathcal{V}(\mathcal{H}) \mid e \notin \mathcal{H}\}$, ainsi : $\mathrm{P}(k) =$

$$\begin{array}{ll} \exists \mathrm{Cycle}\,(-k+1) \subseteq \overline{\mathcal{H}} & \text{si } k < -1 \quad (-k+1 \geq 3) \\ \exists a,b,c\, \{a,b\} \in \mathcal{H} \wedge \{b,c\} \notin \mathcal{H} \wedge \{a,c\} \notin \mathcal{H} & \text{si } k = -1 \\ \exists a,b,c\, \{a,b\} \in \mathcal{H} \wedge \{b,c\} \in \mathcal{H} \wedge \{a,c\} \notin \mathcal{H} & \text{si } k = 1 \\ \exists \mathrm{Tetra}\,(k+1) \subseteq \mathcal{H} & \text{si } k > 1 \qquad (k+1 \geq 3) \end{array}$$

On définit $\mathrm{H}_d(0)$ comme « vrai », et $\mathrm{H}_d(k)$ comme « le problème $\mathrm{P}(k)$ n'est pas décidable en temps linéaire. »

On distingue quelques familles de ces hypothèses :

$$\begin{array}{ll} \mathrm{H}_d^{\mathrm{ns}} & = \forall k \geq 3 \quad \mathrm{H}_d(k) \\ \mathrm{H}_d^{+} & = \forall k > 1 \quad \mathrm{H}_d(k) \\ \mathrm{H}_d^{-} & = \forall k < -1 \quad \mathrm{H}_d(k) \\ \mathrm{H}_d^{0} & = \mathrm{H}_d(-1) \wedge \mathrm{H}_d(1) \\ \mathrm{H}_d^{\star} & = \forall k \quad \mathrm{H}_d(k) \end{array}$$

Remarque 36 (canonicité des hypothèses) Les problèmes de décision portant sur les hypergraphes $\mathrm{P}(k)$, pour $k \leq 1$, sont en fait des problèmes de décision portant sur les graphes. En effet, une arête $\{x, y\}$ est dans un hypergraphe $\mathcal{H}$ si et seulement si elle est dans $\mathcal{G}_{\mathcal{H}}$ le sous-ensemble de $\mathcal{H}$ composé de ses arêtes de taille 2, i.e. $\mathcal{G}_{\mathcal{H}} = \{e \in \mathcal{H} \mid \mathrm{Card}\,(e) = 2\}$.

De même, une arête $\{x, y\}$ est présente dans $\overline{\mathcal{H}}$ si et seulement si elle est présente dans $\overline{\mathcal{G}_{\mathcal{H}}} = \big\{\{x,y\} \subseteq \mathcal{V}(\mathcal{G}) \;\big|\; \{x,y\} \notin \mathcal{G}\big\} = \mathcal{G}_{\overline{\mathcal{H}}}$.

Il s'ensuit que, pour décider $\mathrm{P}(k)(\mathcal{H})$ avec $k \leq 1$, il suffit de (et il faut, trivialement) décider $\mathrm{P}(k)\big(\mathcal{G}_{\mathcal{H}}\big)$. La réduction est linéaire dans les deux sens ; et même, les problèmes d'énumération associés se réduisent parcimonieusement l'un à l'autre.

On verra que $\mathrm{H}_d^{\mathrm{ns}}$ est la seule hypothèse non standard. On prouve maintenant que les autres hypothèses sont raisonnables, en montrant qu'une hypothèse de complexité raisonnable les implique.

Définition 56 (hypothèse H_{T}) On définit l'hypothèse H_{T} comme la proposition : « on ne peut pas tester la présence d'un triangle dans un graphe à n sommets en temps $\mathcal{O}(n^2)$ ».

► ***Lemme 47*** L'hypothèse H_{T} implique les assertions H_d^{-}, H_d^{0}, et $\mathrm{H}_d(2)$.

Démonstration. Supposons que H_d^{-} ne tienne pas, i.e. on peut, pour un certain $k \geq 3$, tester la présence d'un cycle de longueur k dans le graphe

complémentaire en temps $\mathcal{O}(n^2)$, ce qui se ramène [6] à tester la présence d'un cycle de longueur k dans un graphe dans le même temps. Soit $\mathcal{G}$ un graphe dans lequel on souhaite tester la présence d'un triangle.

Soit $\mathcal{G}'$, ainsi défini à partir de $\mathcal{G}$:

$$\mathcal{G}' = \begin{cases} \big\{\{(a,1),(b,2)\} \,\big|\, \{a,b\} \in \mathcal{G}\big\} \\ \uplus \big\{\{(a,2),(b,3)\} \,\big|\, \{a,b\} \in \mathcal{G}\big\} \\ \uplus \big\{\{(a,i),(a,i{+}1)\} \,\big|\, a \in \mathcal{V}(\mathcal{G}), i \in \{3,...,k-1\}\big\} \\ \uplus \big\{\{(a,k),(b,1)\} \,\big|\, \{a,b\} \in \mathcal{G}\big\} \end{cases}$$

On se convainc facilement qu'il existe un triangle dans $\mathcal{G}$ si et seulement si il existe un cycle de longueur k dans $\mathcal{G}'$.

On a $\mathrm{Card}(\mathcal{V}(\mathcal{G}')) = k \times \mathrm{Card}(\mathcal{V}(\mathcal{G}))$, on peut donc tester la présence d'un triangle dans $\mathcal{G}$ en temps $\mathcal{O}\big((kn)^2\big)$, ce qui contredit $\mathrm{H_T}$.

On peut prouver similairement que l'hypothèse $\mathrm{H_T}$ implique les assertions H_d^0 et $\mathrm{H}_d(2)$. ◀

On rappelle que l'outil permettant de prouver des résultats de difficulté est la réduction parcimonieuse $\preccurlyeq$, qui permet de ramener un problème réputé difficile aux hypothèses de complexité.

▶ ***Lemme 48*** Soit $\phi \in \mathrm{SCQ}$. On a :

$$\mathcal{Q}(\phi) \succcurlyeq \mathcal{Q}(\phi[S])$$
$$\mathcal{Q}(\phi) \eqsim \mathcal{Q}(\mathrm{M}(\phi))$$

Démonstration. Le premier point a été prouvé au chapitre 4 ; le second y a aussi été prouvé dans le cas où $\phi \in \mathrm{CQ}$, l'adaptation de la preuve est triviale. ◀

On dispose désormais de tous les éléments pour montrer le résultat de difficulté pour la décision.

▶ ***Lemme 49*** On a les équivalences suivantes :

$$\begin{array}{lcl} \mathrm{H}_d^\star & \Leftrightarrow & \forall \phi \in \mathrm{SCQ} \;\; \big(?\mathcal{Q}(\phi) \in \textsc{Lin} \Rightarrow \mathcal{A}\binom{\mathcal{H}_-(\phi)}{\mathcal{H}_+(\phi)}\big) \\ \mathrm{H}_d^+ & \Leftrightarrow & \forall \phi \in \mathrm{CQ} \;\; \big(?\mathcal{Q}(\phi) \in \textsc{Lin} \Rightarrow \mathcal{A}_\alpha(\mathcal{H}(\phi))\big) \\ \mathrm{H}_d^- & \Leftrightarrow & \forall \phi \in \mathrm{NCQ} \;\; \big(?\mathcal{Q}(\phi) \in \textsc{Lin} \Rightarrow \mathcal{A}_\beta(\mathcal{H}(\phi))\big) \end{array}$$

Démonstration. En appliquant le lemme 48 à la caractérisation négative de l'acyclicité du théorème 25, on prouve qu'une requête signée non acyclique exprime une requête parmi les suivantes :

- une requête conjonctive négative dont l'hypergraphe est un graphe-cycle, ou bien
- une requête conjonctive positive dont l'hypergraphe est isomorphe à $\mathrm{Tetra}(k)$ avec $k \geq 3$, ou bien
- une requête conjonctive signée dont l'hypergraphe est un triangle (avec au moins une arête négative et au moins une positive).

Ces requêtes sont équivalentes [7] aux problèmes $\mathrm{P}(k)$. ◀

2.1.3 Dichotomie

On n'a plus qu'à appliquer le résultat de facilité (théorème 28) pour obtenir les résultats de dichotomie.

6 Noter que, en temps $\mathcal{O}(n^2)$, on peut construire le complémentaire d'un graphe.

7 On a prouvé l'équivalence uniquement au sens de la décision, i.e. on a une réduction linéaire qui préserve le nombre de variables dans les deux sens.

ÉlimineQuantificateurs(ϕ)($\mathcal{S}$) :
  Soit $(x_1,...,x_n)$ un OEI de ϕ, avec $x_k,...,x_n$ les variables quantifiées
  pour $i \in (n,...,k)$:
    tant qu'on peut trouver $R(\bar{y},\bar{z},x_i)$ et $S(\bar{z},x_i)$ dans ϕ :
      $R^{\mathcal{S}} \leftarrow \{(\bar{a},\bar{b},c) \in R^{\mathcal{S}} \mid (\bar{b},c) \in S^{\mathcal{S}}\}$
      Retirer $S(\bar{z},x_i)$ de ϕ
    tant qu'on peut trouver $R(\bar{y},\bar{z},x_i)$ et $\neg S(\bar{z},x_i)$ dans ϕ :
      $R^{\mathcal{S}} \leftarrow \{(\bar{a},\bar{b},c) \in R^{\mathcal{S}} \mid (\bar{b},c) \notin S^{\mathcal{S}}\}$
      Retirer $\neg S(\bar{z},x_i)$ de ϕ
    tant que $D^{\mathcal{S}}_{x_i} \neq \{0\}$:
      si x_i est contenu dans un atome positif $R^+_a(\bar{y},x_i)$:
        $(R^+_{a-1})^{\mathcal{S}} \leftarrow \{(e_1,...,e_{a-1}) \mid \exists e_a\,(e_1,...,e_a) \in (R^+_a)^{\mathcal{S}}\}$
        $(R^-_a)^{\mathcal{S}} \leftarrow ((R^+_{a-1})^{\mathcal{S}} \times \{0,1\}) \setminus (R^+_a)^{\mathcal{S}}$
        Remplacer $R^+_a(\bar{y},x_i)$ par $R^+_{a-1}(\bar{y}) \wedge \neg R^-_a(\bar{y},x_i)$ dans ϕ
        $(\phi,\mathcal{S}) \leftarrow$ RetirePointImbrication(ϕ,x_i,$\mathcal{S}$)
        $(R^+_{a-1})^{\mathcal{S}} \leftarrow (R^+_{a-1})^{\mathcal{S}} \setminus (R^-_a)^{\mathcal{S}}$
      sinon
        $(\phi,\mathcal{S}) \leftarrow$ RetirePointImbrication(ϕ,x_i,$\mathcal{S}$)
      tant que ϕ contient un atome négatif (resp. positif) $\neg R()$ (resp. $R()$) :
        si $R^{\mathcal{S}} \neq \emptyset$ (resp. $R^{\mathcal{S}} = \emptyset$) :
          renvoyer Faux
        Retirer $\neg R()$ (resp. $R()$) de ϕ
  renvoyer ϕ,$\mathcal{S}$

● **Algorithme 6.1 :** Élimination de quantificateurs pour les SCQ sur domaines booléens.

▶ **Théorème 29 (dichotomies pour la décision)**

On a :

$$\begin{array}{lcl} \mathrm{H}^{\star}_d & \Leftrightarrow & \forall \phi \in \mathrm{SCQ}\ \left(?\mathcal{Q}(\phi) \in \textsc{Lin} \Leftrightarrow \mathcal{A}\binom{\mathcal{H}_-(\phi)}{\mathcal{H}_+(\phi)}\right) \\ \mathrm{H}^{+}_d & \Leftrightarrow & \forall \phi \in \mathrm{CQ}\ \left(?\mathcal{Q}(\phi) \in \textsc{Lin} \Leftrightarrow \mathcal{A}_\alpha(\mathcal{H}(\phi))\right) \\ \mathrm{H}^{-}_d & \Leftrightarrow & \forall \phi \in \mathrm{NCQ}\ \left(?\mathcal{Q}(\phi) \in \textsc{Lin} \Leftrightarrow \mathcal{A}_\beta(\mathcal{H}(\phi))\right) \end{array}$$

En particulier, si l'on suppose $\mathrm{H_T}$ vraie, ces équivalences deviennent :

$$\begin{array}{lcl} \mathrm{H}^{\mathrm{ns}}_d & \Leftrightarrow & \forall \phi \in \mathrm{SCQ}\ \left(?\mathcal{Q}(\phi) \in \textsc{Lin} \Leftrightarrow \mathcal{A}\binom{\mathcal{H}_-(\phi)}{\mathcal{H}_+(\phi)}\right) \\ \mathrm{H}^{\mathrm{ns}}_d & \Leftrightarrow & \forall \phi \in \mathrm{CQ}\ \left(?\mathcal{Q}(\phi) \in \textsc{Lin} \Leftrightarrow \mathcal{A}_\alpha(\mathcal{H}(\phi))\right) \\ & & \forall \phi \in \mathrm{NCQ}\ \left(?\mathcal{Q}(\phi) \in \textsc{Lin} \Leftrightarrow \mathcal{A}_\beta(\mathcal{H}(\phi))\right) \end{array}$$

Démonstration. Le premier point est dû au résultat de difficulté (lemme 49) et au résultat de facilité (lemme 28). Le second point est un corollaire du premier en appliquant le lemme 47. ◀

2.2 Énumération

2.2.1 Facilité

Élimination des quantificateurs On propose un algorithme d'élimination des quantificateurs, qui constitue un nouvel algorithme de décision. Il s'agit d'une variante évidente de l'algorithme du chapitre 5 de ce mémoire (notre article [BB12a]), à ceci près qu'en plus, il procède à un filtrage à chaque fois que c'est possible.

Dans un premier temps, on traite l'aspect combinatoire de l'algorithme à l'aide d'un petit lemme qui établit formellement la correspon-

dance (en fait l'équivalence) entre la notion d'ordre d'élimination inverse et le procédé d'élimination, extrêmement proche, qui sera utilisé par l'algorithme.

▶ ***Lemme 50*** Soit $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ un hypergraphe bicolore acyclique et supposons que $(x_1,...,x_n)$ est un OEI de cet hypergraphe (ce que l'on peut toujours forcer en renommant les variables). Alors on peut réduire $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ à $\binom{\emptyset}{\emptyset}$ en appliquant la procédure suivante :

pour $i \in [n,...,1]$:
$\binom{\mathcal{H}_2}{\mathcal{H}_1} \leftarrow \binom{\mathcal{H}_2 \setminus \{e \in \mathcal{H}_2\langle x_i\rangle \mid \exists f \in \mathcal{H}_1\ e \subseteq f\}}{\mathcal{H}_1 \setminus \{e \in \mathcal{H}_1\langle x_i\rangle \mid \exists f \in \mathcal{H}_1\ e \subset f\}}$
$\binom{\mathcal{H}_2}{\mathcal{H}_1} \leftarrow \binom{\mathcal{H}_2}{\mathcal{H}_1}[\setminus\{x_i\}]$

De plus, chaque x_i est un point d'imbrication au moment de son retrait.

Démonstration. On considère l'hypergraphe bicolore :

$$\binom{\mathcal{H}_2 \setminus \{e \in \mathcal{H}_2\langle x_i\rangle \mid \exists f \in \mathcal{H}_1\ e \subseteq f\}}{\mathcal{H}_1 \setminus \{e \in \mathcal{H}_1\langle x_i\rangle \mid \exists f \in \mathcal{H}_1\ e \subset f\}} = \binom{\mathcal{H}_2}{\mathcal{H}_1} \setminus \binom{\mathcal{H}_2}{\mathcal{H}_1}\langle x_i\rangle \cup \mathrm{M}\left(\binom{\mathcal{H}_2}{\mathcal{H}_1}\langle x_i\rangle\right)$$

Cet hypergraphe est une minimisation « partielle », qui consiste à retirer des arêtes incluses dans une arête de $\mathcal{H}_1\langle x_i\rangle$. Aussi, cet hypergraphe, que l'on nomme $\binom{\mathcal{H}'_2}{\mathcal{H}'_1}$, est tel que $\mathrm{M}\binom{\mathcal{H}'_2}{\mathcal{H}'_1} = \mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1}$. Puisque $\mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1}$ et $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ ont les mêmes OEI, et que, de la même manière, les hypergraphes bicolores $\binom{\mathcal{H}'_2}{\mathcal{H}'_1}$ et $\mathrm{M}\binom{\mathcal{H}'_2}{\mathcal{H}'_1} = \mathrm{M}\binom{\mathcal{H}_2}{\mathcal{H}_1}$ ont les mêmes OEI, on en déduit que $\binom{\mathcal{H}'_2}{\mathcal{H}'_1}$ et $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ ont les mêmes OEI.

Pour conclure, il suffit de noter qu'une feuille de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ est évidemment un point d'imbrication de $\binom{\mathcal{H}'_2}{\mathcal{H}'_1}$. ◀

▶ **Théorème 30 (élimination des quantificateurs)**

Une requête conjonctive signée $\mathcal{Q}(\phi)$ telle que l'hypergraphe bicolore $\binom{\mathcal{H}^-(\phi) \cup \{\mathrm{F}(\phi)\}}{\mathcal{H}^+(\phi)}$, où $\mathrm{F}(\phi)$ est l'ensemble des variables libres de ϕ, est acyclique, peut être transformée en une requête conjonctive signée *sans quantificateurs* qui admet exactement les mêmes solutions ; ceci *via* une réduction calculable en temps $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$, qui préserve [8] les tailles de la structure et de la formule.

Démonstration. On transforme la requête en requête conjonctive signée à domaines booléens avec $n \log |\mathcal{S}|$ variables, qui aura également trivialement la propriété d'acyclicité. On applique l'algorithme 6.1, dont la correction et la complexité des routines sont explicitées au chapitre 5 (article [BB12a]). Ce papier indique un temps $\mathcal{O}(n|\mathcal{S}| \log |\mathcal{S}|)$, qui résultait d'une erreur de calcul : chaque « clause » est consultée $\log |\mathcal{S}|$ fois, ce qui donne un temps $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$.

La seule différence avec l'algorithme du chapitre 5 est le fait qu'on effectue un filtrage à chaque fois que possible : la correction de cette étape est justifiée par le lemme 50. ◀

Énumération des requêtes sans quantificateurs On va maintenant présenter deux algorithmes d'énumération pour les requêtes conjonctives signées acycliques.

Le premier fait usage d'un précalcul coûteux (exponentiel en ϕ, mais tout de même quasi-linéaire en $|\mathcal{S}|$ à ϕ fixée), mais permet ainsi d'énumérer à délai constant (ou à délai $\mathcal{O}(\log \log |\mathcal{S}|)$ si l'on souhaite un ordre lexicographique). On invite le lecteur à vérifier par lui-même que, suite à ce précalcul, on pourrait même traiter le problème du i-ième

8 Elle les fait même diminuer, i.e. $|\phi'| \leq |\phi|$ et $|\mathcal{S}'| \leq |\mathcal{S}|$.

élément à délai logarithmique. Ceci montre qu'il s'agit d'un précalcul aussi « complet » que possible.

Par ailleurs, on va présenter un second algorithme d'énumération dont le précalcul est *très* raisonnable, précisément $\mathcal{O}(|\mathcal{S}|\log|\mathcal{S}|)$ et qui permet d'énumérer à délai logarithmique.

Le fait suivant a été prouvé au chapitre 4.

▶ ***Lemme 51*** Soit $\phi = (x_1,...,x_n)\,|\,\psi$ une requête conjonctive où ψ est sans quantificateurs. Si $\mathcal{H}(\phi)$ admet $(x_1,...,x_n)$ comme ordre d'élimination inverse, alors $\mathcal{Q}(\phi)$ est énumérable à délai $\mathcal{O}(1)$, dans un ordre lexicographique correspondant à $(x_1,...,x_n)$, après un précalcul en temps $\mathcal{O}(|\mathcal{S}|)$. ◀

▶ ***Lemme 52*** Si $(x_1,...,x_n)$ est un ordre d'élimination inverse de l'hypergraphe bicolore $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ alors c'est aussi un ordre d'élimination inverse (un OEI α) de chacun des hypergraphes $h \cup \mathcal{H}_1$ pour $h \subseteq \mathcal{H}_2'$ où :

$$\mathcal{H}_2' \;=\; \bigcup_{1\leq k\leq n} \mathcal{H}_2\big[\{x_1,...,x_k\}\big]$$

Démonstration. On prouve par récurrence sur n qu'un OEI de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ est un OEI de $\binom{\mathcal{H}_2'}{\mathcal{H}_1}$. On suppose donc cela vrai pour tout hypergraphe bicolore à n sommets, et on considère un hypergraphe bicolore à $n+1$ sommets. On suppose que $(x_1,...,x_{n+1})$ est un OEI de cet hypergraphe. On pose $\mathcal{H}_2' = \bigcup_{k=1}^{n+1} \mathcal{H}_2\big[\{x_1,...,x_k\}\big]$.

1 Remarquons que $\binom{\mathcal{H}_2}{\mathcal{H}_1}\langle x_{n+1}\rangle = \binom{\mathcal{H}_2'}{\mathcal{H}_1}\langle x_{n+1}\rangle$. Il s'ensuit que $\binom{\mathcal{H}_2'}{\mathcal{H}_1}$ admet bien x_{n+1} comme feuille.

2 L'ordre $(x_1,...,x_n)$ est un OEI de $\binom{\mathcal{H}_2}{\mathcal{H}_1}\big[\smallsetminus\{x_n\}\big]$, donc par hypothèse de récurrence c'en est aussi un de $\binom{\mathcal{H}_2''}{\mathcal{H}_1[\smallsetminus\{x_{n+1}\}]}$ où :

$$\mathcal{H}_2'' = \bigcup_{1\leq k\leq n} \mathcal{H}_2\big[\smallsetminus\{x_{n+1}\}\big]\big[\{x_1,...,x_k\}\big] = \mathcal{H}_2'\big[\smallsetminus\{x_{n+1}\}\big]$$

donc $x_1,...,x_n$ est un OEI de $\binom{\mathcal{H}_2'}{\mathcal{H}_1}\big[\smallsetminus\{x_{n+1}\}\big]$.

Ces deux faits prouvent conjointement que $(x_1,...,x_{n+1})$ est un OEI de $\binom{\mathcal{H}_2'}{\mathcal{H}_1}$, ce qui clôt la récurrence. Le premier point du lemme 44 (page 110) nous donne alors le résultat attendu. ◀

▶ ***Lemme 53 (énumération naïve)*** Soit la formule :

$$\phi = (x_1,...,x_n)\,|\,\exists x_{n+1}...x_k\;\psi \text{ où}$$
$$\psi = \neg R_1(\vec{v}_1)\wedge...\wedge\neg R_l(\vec{v}_l)\wedge R_{l+1}(\vec{v}_{l+1})\wedge...\wedge R_m(\vec{v}_m)$$

Si l'hypergraphe bicolore $\binom{\mathcal{H}^-(\phi)\cup\{\{x_1,...,x_n\}\}}{\mathcal{H}^+(\phi)}$ est acyclique, alors $\mathcal{Q}(\phi)$ est énumérable :

- à délai $\mathcal{O}(1)$,
- ou à délai $\mathcal{O}(m\log\log|\mathcal{S}|)$ dans l'ordre lexicographique $(x_1,...,x_n)$,

après un précalcul en temps $\mathcal{O}\big((\log|\mathcal{S}|)^{|\phi|}|\mathcal{S}|\big)$.

Démonstration. Comme précédemment, on se ramène au domaine booléen, ce qui maintient l'ordre d'élimination pour les mêmes raisons. Par le théorème 30, on peut se débarrasser des quantificateurs en temps $\mathcal{O}(|\mathcal{S}|\log|\mathcal{S}|)$.

Grâce à l'astuce booléenne de Zanuttini [ZH02], mentionnée dans le chapitre 5 [BB12a], on peut construire, en temps linéaire pour chaque R_i où $i \leq l$, un ensemble de relations R_i^j telle que $\neg R_i(\vec{v}_i) = \biguplus_j R_i^j(\vec{v}_i^{\,j})$, avec $j \leq \lceil\log|\mathcal{S}|\rceil$. On a $l \leq m$ disjonctions de $\lceil\log|\mathcal{S}|\rceil$ éléments.

Par distributivité, on réécrit la formule ψ sous la forme d'une « disjonction disjointe » de conjonctions, c'est-à-dire sous la forme

$\phi = (x_1,...,x_n) \mid \psi$ où :

$$\psi = \biguplus_{i_1,...,i_l} R_1^{i_1}(\vec{v}_1^{i_1}) \wedge ... \wedge R_l^{i_l}(\vec{v}_l^{i_l}) \wedge R_{l+1}(\vec{v}_{l+1}) \wedge ... \wedge R_m(\vec{v}_m)$$

où les i_j sont majorés par $\lceil \log |\mathcal{S}| \rceil$. On obtient donc ici une disjonction de $(\log |\mathcal{S}|)^m$ conjonctions positives. De par le lemme 52, elles sont toutes α–acycliques, et même admettent un ordre *commun* d'α–élimination. Par conséquent, ces conjonctions sont toutes énumérables à délai $\mathcal{O}(1)$ *dans le même ordre lexicographique*, grâce au lemme 51.

Finalement, après un précalcul en temps $\mathcal{O}\big(|\mathcal{S}| \log |\mathcal{S}| + |\mathcal{S}|(\log |\mathcal{S}|)^m\big)$ c'est-à-dire, en majorant très grossièrement, $\mathcal{O}\big(|\mathcal{S}|(\log |\mathcal{S}|)^{|\phi|}\big)$,

- soit on énumère chaque requête conjonctive l'une après l'autre, ce qui nous donne un délai constant ;
- soit, afin d'énumérer dans un ordre lexicographique, on effectue une fusion de $(\log |\mathcal{S}|)^m$ listes triées (et disjointes) et on obtient un délai de $\mathcal{O}\big(\log\big((\log |\mathcal{S}|)^m\big)\big) = \mathcal{O}(m \log\log |\mathcal{S}|)$, délai dû à l'usage d'un tas contenant à tout moment $\lceil \log |\mathcal{S}| \rceil^m$ éléments. ◀

▶ **Théorème 31 (facilité de l'énumération)**

Une requête $\mathcal{Q}(\phi)$ telle que l'hypergraphe bicolore $\binom{\mathcal{H}_{-(\phi)} \cup \{\mathrm{F}(\phi)\}}{\mathcal{H}_{+(\phi)}}$ est acyclique, où $\mathrm{F}(\phi)$ est l'ensemble des variables libres de ϕ, s'énumère à délai $\mathcal{O}(|o| \log |\mathcal{S}|)$. après précalcul $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$, où n est le nombre de variables de ϕ et $|o|$ la taille d'une solution produite.

Démonstration. On peut là encore se débarrasser des quantificateurs. On va modifier l'algorithme de décision quasi–linéaire pour obtenir l'algorithme 6.2. Déjà, on réécrit la boucle sous forme récursive pour rendre explicite la réduction. De plus, on redéfinit la routine RetirePointImbrication pour que, en plus de retirer le point d'imbrication, elle construise une formule F_k de taille $|\mathcal{S}|$ portant sur les variables $x_1,...,x_k$ telle que :

$$((a_1,...,a_k) \vDash F_k) \quad \Leftrightarrow \quad \exists (a_{k+1},...,a_n)\ (a_1,...,a_n) \in \mathcal{Q}(\phi)(\mathcal{S})$$

Pour ce, il suffit d'ajouter, dans sa définition (chapitre 5, page 99), à la suite de la ligne $F \leftarrow \mathrm{Res}(F,x)$, la ligne $F_k \leftarrow \{C \in F \mid \nexists C' \in \mathrm{Res}(F,x)\ C' \to C\}$, et de retourner aussi cette formule. Cette ligne signifie que la formule F_k est l'ensemble des clauses C de F qui ne sont pas impliquées par celles du résolvant. Qu'une clause C de F ne soit pas impliquée par une clause du résolvant est ici équivalent à ce que cette clause privée du littéral portant sur x_k ne soit pas présente dans le résolvant.

Il s'ensuit que les modèles de $\bigwedge_i F_i$ sont les modèles de F, c'est–à–dire les solutions de la requête. On explique maintenant l'utilisation de ces clauses pour énumérer. Comme les modèles partiels $x_1,...,x_k$ qui ne violent aucune formule F_k pour $k \leq n$ admettent toujours un prolongement, on sait déjà qu'au bout d'au plus $2n$ échecs, on produit une solution. Reste à déterminer la complexité du test $m \cup \{\neg x_k\} \vDash F_k$.

Notons déjà que F_k est une conjonction de clauses imbriquées ; plus formellement, cela signifie qu'il existe une suite $j_1, j_2,...$ *croissante* d'indices telle que toute clause C de F_k porte exactement sur les variables $x_{j_1},...,x_{j_i}$ pour un certain i (qui, lui, dépend de la clause). Il est clair que $j_1 = k$. Comme on le fait pour le calcul du résolvant, on peut donc représenter F_k sous forme d'une liste L de mots sur $+,-$, que l'on peut supposer lexicographiquement ordonnée. De plus, on peut faire en sorte qu'aucun mot de L n'est préfixe d'un autre [9] mot de L.

9 Si un mot w_1 est préfixe d'un autre mot w_2, alors la clause C_1 correspondant à

RetirePointImbrication(ϕ,x,$\mathcal{S}$) :
  // ψ est sous la forme : $\bigwedge_i \neg R_i\left(x_{f(i,1)},...,x_{f(i,n(i))}\right)$
  // On pose $\sigma(1)=\neg$, $\sigma(0)=\epsilon$
  $F \leftarrow \bigwedge_i \bigwedge_{(a_1,...,a_{n(i)})\in R_i^{\mathcal{S}}} \sigma(a_1)x_{f(i,1)} \vee ... \vee \sigma(a_{n(i)})x_{f(i,n(i))}$
  Remplacer $\neg R(x_1,...,x_n,x)$ par $\neg R(x_1,...,x_n)$ dans ϕ
  $F_r \leftarrow \mathrm{Res}(F,x)$
  $F' \leftarrow \{C \in F \mid x \in C \wedge C \setminus \{x\} \notin F_r\}$
  Reconstruire $\mathcal{S}$ à partir de F_r
  renvoyer $(\phi,\mathcal{S},F')$

SCQ − BoolD($\phi(x_1,...,x_k)$)($\mathcal{S}$) :
  si $n=0$:
    si ϕ contient un atome négatif $\neg R()$ tel que $R^{\mathcal{S}} \neq \emptyset$:
      arrêter
    si ϕ contient un atome positif $R()$ tel que $R^{\mathcal{S}} = \emptyset$:
      arrêter
    produire $\emptyset$
  sinon
    tant qu'on peut trouver $R(\bar{y},\bar{z},x_k)$ et $S(\bar{z},x_k)$ dans ϕ :
      $R^{\mathcal{S}} \leftarrow \left\{(\bar{a},\bar{b},c) \in R^{\mathcal{S}} \mid (\bar{b},c) \in S^{\mathcal{S}}\right\}$
      Retirer $S(\bar{z},x_k)$ de ϕ
    tant qu'on peut trouver $R(\bar{y},\bar{z},x_k)$ et $\neg S(\bar{z},x_k)$ dans ϕ :
      $R^{\mathcal{S}} \leftarrow \left\{(\bar{a},\bar{b},c) \in R^{\mathcal{S}} \mid (\bar{b},c) \notin S^{\mathcal{S}}\right\}$
      Retirer $\neg S(\bar{z},x_k)$ de ϕ
    si x_k est contenu dans un atome positif $R_a^+(\bar{y},x_k)$:
      $(R_{a-1}^+)^{\mathcal{S}} \leftarrow \{(e_1,...,e_{a-1}) \mid \exists e_a\,(e_1,...,e_a) \in (R_a^+)^{\mathcal{S}}\}$
      $(R_a^-)^{\mathcal{S}} \leftarrow ((R_{a-1}^+)^{\mathcal{S}} \times \{0,1\}) \setminus (R_a^+)^{\mathcal{S}}$
      Remplacer $R_a^+(\bar{y},x_k)$ par $R_{a-1}^+(\bar{y}) \wedge \neg R_a^-(\bar{y},x_k)$ dans ϕ
      $(\phi,\mathcal{S},F_k) \leftarrow$ RetirePointImbrication(ϕ,x_k,$\mathcal{S}$)
      $(R_{a-1}^+)^{\mathcal{S}} \leftarrow (R_{a-1}^+)^{\mathcal{S}} \setminus (R_a^-)^{\mathcal{S}}$
    sinon
      $(\phi,\mathcal{S},F_k) \leftarrow$ RetirePointImbrication(ϕ,x_k,$\mathcal{S}$)
    pour $m \in$ SCQ − BoolD($\phi(x_1,...,x_{k-1})$)($\mathcal{S}$) :
      si $m \cup \{\neg x_k\} \vDash F_k$:
        produire $m \cup \{\neg x_k\}$
      si $m \cup \{\ x_k\} \vDash F_k$:
        produire $m \cup \{\ x_k\}$

● **Algorithme 6.2 :** Énumération des SCQ sur les domaines booléens.

Satisfaire la formule F_k revient à ne violer aucune des clauses qui la composent. On peut le faire par recherche dichotomique, ce que l'on prouve maintenant. Soit m le modèle (c'est-à-dire une conjonction de littéraux où toutes les variables sont présentes) ; on peut le supposer restreint à l'ensemble des x_{j_i}, c'est-à-dire ignorer les littéraux portant sur des variables n'apparaissant pas dans F_k, car ils n'interviennent pas.

Soit C une clause contenant x_k représentée par un mot w, qui porte donc sur les littéraux d'indices croissants $x_{j_1},...,x_{j_i}$ pour un certain i. Le modèle m viole C si et seulement si $C \to \overline{m}$, où $\overline{m}$ est la disjonction des littéraux opposés à ceux de m. Cet énoncé est vrai si et seulement si $\overline{m}$ porte tous les littéraux présents dans C, autrement dit si et seulement si le mot w_m associé à la « clause » $\overline{m}$ admet w (le mot représentant

w_1 implique la clause C_2 correspondant à w_2, qui peut être oubliée. Supprimer ainsi les clauses impliquées trivialement par d'autres se fait en temps linéaire (il suffit de trier puis de parcourir la liste une seule fois).

C) comme préfixe. Comme aucun mot de L n'est préfixe d'un autre, soit L contient exactement un mot préfixe de w_m, soit aucun. C'est ce préfixe que l'on va rechercher par dichotomie. Pour ce, on recherche dichotomiquement dans L la présence du mot w_m, et on conserve l'indice du dernier mot auquel on a comparé w_m, qui constitue soit le mot précédant w_m, soit le mot succédant à w_m dans l'ordre lexicographique.

On peut ainsi connaître w_p le mot précédant w_m dans l'ordre lexicographique, qui est le mot de L ayant le plus long préfixe commun avec w_m. Si w_p est effectivement préfixe de ou égal à w_m, on a la réponse. Dans le cas contraire, supposons qu'il existe un mot p de L qui soit préfixe de w_m. Puisque w_p possède le plus long préfixe commun avec w_m, p est aussi préfixe de w_p, contradiction. Par conséquent, si w_p n'est pas préfixe de w_m, alors aucun mot de L n'est préfixe de w_m. On n'effectue ainsi qu'une seule recherche dichotomique *standard* sur un objet de taille $|\mathcal{S}|$, ce qui coûte donc $\mathcal{O}(\log|\mathcal{S}|)$.

On a au total un délai de $\mathcal{O}\left(n'\log|\mathcal{S}|\right)$ où n' est le nombre de variables de la requête booléanisée, on a donc un délai de $\mathcal{O}\big(n(\log|\mathcal{S}|)^2\big)$ où n est le nombre de variables de la formule originelle. La solution o produite étant de taille $\Theta(n\log|\mathcal{S}|)$, on a un délai $\mathcal{O}(|o|\log|\mathcal{S}|)$. ◀

A posteriori, on peut maintenant dire que l'algorithme de décision 6.1 n'est qu'une version abrégée de l'algorithme d'énumération.

2.2.2 Difficulté

On redéfinit des variantes un peu renforcées des hypothèses précédentes dans lesquelles le temps *quasi-linéaire* se substitue au temps *linéaire*.

Définition 57 Comme au chapitre 4, qLin (resp. qLinD) se réfère au temps quasi-linéaire $n(\log n)^{\mathcal{O}(1)}$ (resp. délai quasi-linéaire). La complexité LogD∘qLin se réfère au délai logarithmique ($\mathcal{O}(|o|\log n)$ où o est l'objet produit) après un précalcul *quasi-linéaire*.

Définition 58 (hypothèse $\mathrm{qH_T}$) On définit l'hypothèse $\mathrm{qH_T}$ comme « on ne peut tester la présence d'un triangle dans un graphe à n sommets en temps $\mathcal{O}(n^2\log n)$ ». Similairement, on redéfinit toutes les hypothèses précédentes (définition 55) préfixées par « q » pour indiquer qu'elles se réfèrent désormais au temps quasi-linéaire, i.e. $|\mathcal{S}|(\log|\mathcal{S}|)^{\mathcal{O}(1)}$.

On définit maintenant une hypothèse de complexité pour l'énumération, dont on verra au lemme suivant qu'elle est raisonnable, car elle est, elle aussi, impliquée par l'hypothèse concernant le problème du triangle.

Définition 59 (hypothèse H_e) On définit $\mathrm{H}_e(k) = \mathrm{P}_e(k) \notin$ LogD∘qLin, où :

$$\mathrm{P}_e(k) = \begin{cases} \mathcal{Q}((x_1,x_3)\,|\,\exists x_2\ R(x_1,x_2)\wedge\ R(x_2,x_3)) & \text{si } k=1 \\ \mathcal{Q}((x_1,x_3)\,|\,\exists x_2\ R(x_1,x_2)\wedge\neg R(x_2,x_3)) & \text{si } k=2 \\ \mathcal{Q}\left((x_1,x_k)\;\middle|\;\exists x_2 ... \exists x_k \bigwedge_{1\le i<k} \neg R(x_i,x_{i+1})\right) & \text{si } k\ge 3 \end{cases}$$

On définit $\mathrm{H}_e^\star = \forall k \ge 1\ \mathrm{H}_e(k)$.

► ***Lemme 54*** L'hypothèse $\mathrm{qH_T}$ implique $\mathrm{H}_e^\star$.

Démonstration. Supposons $\mathrm{qH_T}$ vraie et $\mathrm{H}_e^\star$ fausse. Alors on peut trouver k tel que P_e soit dans LogD∘qLin. Alors, par une preuve similaire à celle du lemme 47, $\mathrm{P}_e(1)$ est énumérable à délai $\mathcal{O}\big(\log n\big)$ après précalcul

$\mathcal{O}(n^2 \log n)$. Dans ce cas on peut calculer le produit de deux matrices booléennes carrées de dimension n en temps $\mathcal{O}(n^2 \log n)$, et par conséquent l'hypothèse $\mathrm{qH_T}$ est invalidée. ◀

▶ ***Lemme 55*** On a :

$$\mathrm{qH}_d^\star \wedge \mathrm{H}_e^\star \quad \Leftrightarrow \quad \forall \phi \in \mathrm{SCQ} \ \left(\mathcal{Q}(\phi) \in \textsc{LogD}\circ\textsc{qLin} \Rightarrow \mathcal{A}\left(\begin{smallmatrix} \mathcal{H}_-(\phi) \cup \{\mathrm{F}(\phi)\} \\ \mathcal{H}_+(\phi) \end{smallmatrix}\right) \right)$$

où $\mathrm{F}(\phi)$ est l'ensemble des variables libres de ϕ.

Démonstration. Il est aisé de vérifier que, si l'une des hypothèses de complexité est fausse, il existe une requête qui est facile malgré son hypergraphe non acyclique. On se place dans le cas contraire, i.e. les hypothèses de complexité tiennent. Soit ϕ une requête conjonctive signée. Si $\mathcal{A}\left(\begin{smallmatrix} \mathcal{H}_-(\phi) \cup \{\mathrm{F}(\phi)\} \\ \mathcal{H}_+(\phi) \end{smallmatrix}\right)$, le résultat est trivial (il n'y a rien à montrer).

Sinon, on a deux cas. Si $\neg\mathcal{A}\left(\begin{smallmatrix} \mathcal{H}_-(\phi) \\ \mathcal{H}_+(\phi) \end{smallmatrix}\right)$, alors on peut réutiliser la preuve de résultat de difficulté de la décision (lemme 49) en remplaçant « linéaire » par « quasi-linéaire », pour montrer que la requête n'est pas décidable en temps quasi-linéaire (sinon $\mathrm{qH}_d^\star$ serait fausse).

Dans le cas contraire, si $\neg\mathcal{A}\left(\begin{smallmatrix} \mathcal{H}_-(\phi) \cup \{\mathrm{F}(\phi)\} \\ \mathcal{H}_+(\phi) \end{smallmatrix}\right)$, alors, de par le théorème 25, notre hypergraphe contient (en retirant des arêtes noires et des sommets) un graphe-cycle, dont $\mathrm{F}(\phi)$ (ou son intersection avec les sommets restants) est une arête. Dans ce cas, notre requête exprime bien, de par le lemme 48, l'un des problèmes $\mathrm{P_e}(k)$. Il s'ensuit que la requête n'est pas énumérable à délai logarithmique après précalcul quasi-linéaire (sans quoi l'hypothèse de complexité $\mathrm{H}_\mathrm{e}^\star(k)$ serait fausse). ◀

2.2.3 Dichotomie

▶ **Théorème 32 (dichotomie pour l'énumération)**

On a :

$$\mathrm{qH}_d^\star \wedge \mathrm{H}_e^\star \quad \Leftrightarrow \quad \forall \phi \in \mathrm{SCQ} \ \left(\mathcal{Q}(\phi) \in \textsc{LogD}\circ\textsc{qLin} \Leftrightarrow \mathcal{A}\left(\begin{smallmatrix} \mathcal{H}_-(\phi) \cup \{\mathrm{F}(\phi)\} \\ \mathcal{H}_+(\phi) \end{smallmatrix}\right) \right)$$

où $\mathrm{F}(\phi)$ est l'ensemble des variables libres de ϕ.

En particulier, si l'on suppose $\mathrm{qH_T}$ vraie, cette équivalence devient :

$$\mathrm{qH}_d^{\mathrm{ns}} \quad \Leftrightarrow \quad \forall \phi \in \mathrm{SCQ} \ \left(\mathcal{Q}(\phi) \in \textsc{LogD}\circ\textsc{qLin} \Leftrightarrow \mathcal{A}\left(\begin{smallmatrix} \mathcal{H}_-(\phi) \cup \{\mathrm{F}(\phi)\} \\ \mathcal{H}_+(\phi) \end{smallmatrix}\right) \right)$$

Quand l'énumération à délai quasi–constant après précalcul quasi–linéaire est possible, l'énumération s'effectue de fait à délai $\mathcal{O}(|o| \log |\mathcal{S}|)$, après précalcul $\mathcal{O}(|\mathcal{S}| \log |\mathcal{S}|)$ où $|o|$ est la taille d'un objet produit.

Démonstration. Le premier point est une conséquence du théorème 31 et du lemme 55 ; le second point en est un corollaire direct de par le lemme 54. ◀

Remarque 37 (inutilité de la notion d'étoile) Alors que la caractérisation des requêtes conjonctives faciles à énumérer nécessitait l'introduction d'une notion supplémentaire de requête étoilée, avec la définition que nous donnons à l'acyclicité, une requête conjonctive signée est facile à énumérer si et seulement elle est acyclique.

Il faut bien noter qu'en fait la dichotomie pour la décision n'est qu'un cas particulier de celle pour l'énumération, et repose sur les mêmes hypothèses, à la différence près qu'il y a entre quasi-linéaire et linéaire.

▶ **Corollaire 33 (facilité des requêtes ∃FO β–acycliques)**

Soit ϕ une requête existentielle du premier ordre. Si $\mathcal{H}(\phi)$, l'hypergraphe correspondant à l'ensemble des atomes présents dans ϕ, est β–acyclique, alors on peut procéder à l'énumération (dans n'importe quel ordre lexicographique) à délai $\mathcal{O}_\phi(|\mathcal{S}|\log|\mathcal{S}|)$, et aucune propriété plus faible de $\mathcal{H}(\phi)$ ne peut même garantir une décision en temps quasi–linéaire.

Si $\mathcal{H}(\phi) \cup \{\mathrm{F}(\phi)\}$ est β–acyclique, où $\mathrm{F}(\phi)$ est l'ensemble des variables libres de ϕ, alors on peut procéder à l'énumération (dans un certain ordre lexicographique) à délai $\mathcal{O}\big(n(\log|\mathcal{S}|)^2\big)$, après un précalcul en temps $\mathcal{O}\big((\log|\mathcal{S}|)^2|\mathcal{S}|\big)$; et aucune propriété plus faible de $\mathcal{H}(\phi), \mathrm{F}(\phi)$ ne peut garantir cette complexité.

Démonstration. Dans les deux cas, on peut toujours se débarrasser des quantificateurs.

Le premier point vient de la facilité de la décision : on réduit l'énumération à la décision à l'aide d'une généralisation de l'algorithme de Creignou et Hébrard [CH97], voir algorithme 2. On découpe les domaines de manière dichotomique. La décision se fait par réduction au cas SCQ : on met ϕ en forme normale disjonctive où chaque modèle apparaît comme une conjonction d'atomes positifs ou négatifs. On a 2^m problèmes de décision qui sont tous des SCQ β–acycliques.

Le second point consiste aussi à mettre ϕ en DNF. On a alors à énumérer une réunion de $2^{|\phi|}$ requêtes SCQ admettant le même ordre lexicographique d'énumération. ◀

Concernant la pertinence du problème d'énumération, le résultat notable de cette partie est le suivant :

▶ **Corollaire 34 (lien entre décision et énumération)**

Soit $\phi = (x_1,...,x_n) \mid \exists x_{n+1}...\exists x_k\ \psi$ avec ψ une conjonction d'atomes positifs ou négatifs sans quantificateur. On savait déjà (par l'alg. de Creignou et Hébrard) :

$$\mathcal{Q}(\phi) \in \text{qLinD} \quad \Leftrightarrow \quad ?\mathcal{Q}(\phi) \in \text{qLin}$$

Sous les hypothèses de complexité $\mathrm{qH_T}$ et $\mathrm{qH}_d^{\mathrm{ns}}$, on a un lien plus fort :

$$\mathcal{Q}(\phi) \in \text{LogD}\circ\text{qLin} \quad \Leftrightarrow \quad ?\mathcal{Q}(\neg R(x_1,...,x_n)\wedge\psi) \in \text{Lin}$$

Conclusion

On a une jolie classe qui inclut et élargit nettement la classe on ne peut plus classique des requêtes conjonctives acycliques, en conservant la même complexité pour la décision. Si l'algorithme de compte semble souffrir dans sa complexité combinée d'une difficulté intrinsèque à compter les modèles de formules CNF quand bien même elles sont bêta–acycliques, celui d'énumération est quasi–optimal.

Cet algorithme d'énumération constitue un apport dans la mesure où il apporte de nouvelles briques algorithmiques à l'évaluation de requêtes : là où on disposait uniquement de projections et de filtrages, on dispose désormais de la résolution de Davis et Putnam pour un point d'imbrication et de filtrages généralisés, qui les expriment. De ces nouvelles briques, la première se distingue en ceci des autres qu'elle est algorithmiquement non-triviale (mais tout de même simple).

Chapitre 7

Encore un peu de combinatoire ?

> La démonstration en logique n'est qu'un auxiliaire mécanique pour reconnaître plus aisément une tautologie, quand elle est compliquée.
>
> Ludwig Wittgenstein – *Tractatus logico-philosophicus* §6.1262

Les faits de combinatoire présentés jusqu'ici ont deux travers majeurs :

- Ils se réfèrent à des résultats non prouvés dans la thèse, ni même faciles à trouver sous une forme simple dans la littérature ;
- ils admettent des preuves trop longues, trop compliquées, et trop peu intelligibles.

On se propose ici d'y porter remède.

Pour ce faire, on a finalement assez peu à faire : la preuve trop longue est la généralisation de la preuve de Brouwer et Kolen du chapitre précédent, et la dépendance externe tient essentiellement au fait qu'un hypergraphe α-acyclique, i.e. conforme et sans cycle, a des feuilles (c'est-à-dire des sommets isolés dans son hypergraphe minimisé).

On va donc re–prouver le théorème 27 de manière simple et ce, *sans s'appuyer sur aucun résultat*. Un autre intérêt de notre preuve est de constituer une preuve alternative à la triple caractérisation de l'alpha-acyclicité, qui de toute façon n'est pas simple à trouver dans la littérature (voir par exemple [AV82] dans [Duc95]).

Notre généralisation précédente de la preuve de Brouwer et Kolen lui donnait du sens, là où il y avait une preuve effective mais sans stratégie lisible. Ce sens est le suivant : la suppression d'une feuille n'apporte essentiellement qu'une nouvelle feuille (ou alors plusieurs feuilles équivalentes). Cette idée peut être exploitée plus directement ; c'est ce que nous faisons dans cette nouvelle preuve plus courte qui non seulement démontre leur théorème, mais aussi le généralise et donne une résultat plus fort. Ce renforcement n'est pas anecdotique, il donne immédiatement un lien fort entre l'ordre d'élimination des sommets et l'acyclicité, qui généralise à la fois l'α-acyclicité et la β-acyclicité. En suivant ces idées, on reprouve maintenant notre résultat principal. Cette preuve est plus simple, mieux architecturée, et porte un sens plus clair.

Définitions

Définition 60 (hypergraphe) On appelle *hypergraphe* (ou *hypergraphe simple*) $\mathcal{H}$ en ensemble d'ensembles non vides, appelés *arêtes*. On définit la *taille* d'un hypergraphe $|\mathcal{H}|$ comme la somme des cardinalités de ses

arêtes. On définit l'ensemble des *sommets* de $\mathcal{H}$ comme la réunion de ses arêtes : $\mathcal{V}(\mathcal{H}) = \bigcup_{e\in\mathcal{H}} e$. On note $\mathcal{H}[S]$ l'*hypergraphe induit de $\mathcal{H}$ sur S*, $\mathcal{H}\langle x\rangle$ le *support du sommet x dans $\mathcal{H}$*, et $\mathrm{M}(\mathcal{H})$ *l'hypergraphe minimisé* de $\mathcal{H}$, définis comme suit :

$$\begin{aligned}\mathcal{H}[S] &= \{e\cap S \mid e\in\mathcal{H}\}\setminus\{\emptyset\}\\ \mathcal{H}\langle x\rangle &= \{\,e\in\mathcal{H}\mid x\in e\}\\ \mathrm{M}(\mathcal{H}) &= \{e\in\mathcal{H}\mid \nexists f\in\mathcal{H}\ e\subset f\}\end{aligned}$$

On dit d'un hypergraphe $\mathcal{H}$ qu'il est *acyclique*, ce que l'on note $\mathcal{A}(\mathcal{H})$ lorsqu'il est α–acyclique, c'est-à-dire conforme et sans cycle.

« Conforme » signifie que toute clique (ensemble de sommets deux à deux voisins, i.e. contenus dans une arête commune) est incluse dans une hyperarête ; « sans cycle » signifie qu'il n'existe aucun ensemble de sommets S tel que l'hypergraphe induit sur S est un graphe classique cyclique, avec, éventuellement, des arêtes singletons en plus.

Un *sommet isolé* d'un hypergraphe $\mathcal{H}$ est un sommet $x\in\mathcal{V}(\mathcal{H})$ tel que $\mathrm{Card}(\mathcal{H}\langle x\rangle) = 1$. On définit l'ensemble des *feuilles d'un hypergraphe* $\mathcal{H}$, noté $\mathcal{L}(\mathcal{H})$, comme l'ensemble des sommets isolés de $\mathrm{M}(\mathcal{H})$.

Définition 61 (hypergraphe bicolore) On appelle *hypergraphe à deux couleurs* (ou encore *bicolore*) le couple $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ où $\mathcal{H}_1$ et $\mathcal{H}_2$ sont des hypergraphes, avec $\mathcal{V}(\mathcal{H}_2)\subseteq\mathcal{V}(\mathcal{H}_1)$. $\mathcal{H}_1$ est l'ensemble des arêtes dites *rouges* de $\mathcal{H}$, $\mathcal{H}_2$ est l'ensemble des arêtes dites *noires* de $\mathcal{H}$, et $\mathcal{H}_1\cup\mathcal{H}_2$ [1] est l'ensemble des arêtes de $\mathcal{H}$.

On définit la *taille* de $\binom{\mathcal{H}_2}{\mathcal{H}_1}$, notée $\left|\binom{\mathcal{H}_2}{\mathcal{H}_1}\right|$ comme $|\mathcal{H}_1|+|\mathcal{H}_2|$. On étend les notations définies pour les hypergraphes ainsi :

$$\begin{aligned}\mathcal{V}\tbinom{\mathcal{H}_2}{\mathcal{H}_1} &= \mathcal{V}(\mathcal{H}_1)\cup\mathcal{V}(\mathcal{H}_2) = \mathcal{V}(\mathcal{H}_1)\\ \tbinom{\mathcal{H}_2}{\mathcal{H}_1}[S] &= \tbinom{\mathcal{H}_2[S]}{\mathcal{H}_1[S]}\\ \tbinom{\mathcal{H}_2}{\mathcal{H}_1}\langle x\rangle &= \tbinom{\mathcal{H}_2\langle x\rangle}{\mathcal{H}_1\langle x\rangle}\\ \mathrm{M}\tbinom{\mathcal{H}_2}{\mathcal{H}_1} &= \binom{\{e\in\mathcal{H}_2\mid\nexists f\in\mathcal{H}_1\ e\subseteq f\}}{\{e\in\mathcal{H}_1\mid\nexists f\in\mathcal{H}_1\ e\subset f\}}\\ \mathcal{A}\tbinom{\mathcal{H}_2}{\mathcal{H}_1} &\Leftrightarrow \forall h\subseteq\mathcal{H}_2\quad \mathcal{A}(\mathcal{H}_1\cup h)\end{aligned}$$

On dit que le sommet x est un *point d'imbrication* d'un hypergraphe bicolore $\binom{\mathcal{H}_2}{\mathcal{H}_1}$ quand on a :

$$\forall e,f\in\mathcal{H}_1\langle x\rangle\cup\mathcal{H}_2\langle x\rangle\quad e\subseteq f\vee f\subseteq e$$

c'est-à-dire que l'ensemble des arêtes portant x dans $\mathcal{H}$ est totalement ordonné pour l'inclusion. On définit l'ensemble des *feuilles d'un hypergraphe bicolore* $\mathcal{H}$, noté $\mathcal{L}(\mathcal{H})$, comme l'ensemble des points d'imbrication de son hypergraphe minimisé $\mathrm{M}(\mathcal{H})$.

Définition 62 (voisinage strict) Soit $\mathcal{H}$ un hypergraphe (simple). On dit d'une arête $e\in\mathcal{H}$ qu'elle est *complète* dans $\mathcal{H}$ si $e = \mathcal{V}(\mathcal{H})$; de la même manière, on dit d'une arête $e\in\mathcal{H}$ qu'elle est *incomplète* dans $\mathcal{H}$ si $e\neq\mathcal{V}(\mathcal{H})$. On dit que deux sommets de $\mathcal{H}$, x et y, sont *voisins* si on peut trouver une arête $e\in\mathcal{H}$ telle que $\{x,y\}\subseteq e$. On dit que deux sommets x et y sont *strictement voisins* si on peut trouver une arête incomplète $\mathcal{V}(\mathcal{H})\neq e\in\mathcal{H}$ telle que $\{x,y\}\subseteq e$.

Soit $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ un hypergraphe bicolore. On dit que deux sommets x et y sont *voisins* (resp. *strictement voisins*) dans $\mathcal{H}$ s'ils sont *voisins* (resp. *strictement voisins*) dans $\mathcal{H}_1\cup\mathcal{H}_2$.

1 Cette union n'est pas nécessairement disjointe, ce qui explique l'utilisation de $\subseteq$ dans la définition de M.

Théorème

Nous redémontrons beaucoup plus concisément le théorème 27, cette fois–ci sans aucune dépendance extérieure ; on prouve aussi accidentellement le lemme 44, qui généralisait un théorème principal du chapitre 4, le théorème 13.

▶ **Théorème 35**

Un hypergraphe bicolore acyclique non vide $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ est tel que :

- $\mathcal{V}(\mathcal{H}) \in \mathcal{H}_1$, i.e. il contient une arête rouge complète,
- ou bien il possède deux feuilles qui ne sont pas strictement voisines.

Démonstration. On le montre par récurrence totale sur la taille de l'hypergraphe. On redéfinit la taille en pondérant d'un poids supérieur à un, deux par exemple, la taille des arêtes noires i.e. $\left|\binom{\mathcal{H}_2}{\mathcal{H}_1}\right| = |\mathcal{H}_1| + 2|\mathcal{H}_2|$. L'unique hypergraphe de taille 1, $\binom{\emptyset}{\{x\}}$ valide bien cette propriété. On suppose qu'elle est vraie pour tout hypergraphe de taille $< n$. Soit $\mathcal{H} = \binom{\mathcal{H}_2}{\mathcal{H}_1}$ un hypergraphe bicolore acyclique de taille n. Si $\mathcal{H}$ contient une arête rouge complète, la propriété est validée. On suppose désormais le contraire, i.e. $\mathcal{V}(\mathcal{H}) \notin \mathcal{H}_1$.

Supposons que $\mathcal{H}$ contienne une arête *noire* complète i.e. $\mathcal{V}(\mathcal{H}) \in \mathcal{H}_2$. Alors on pose $\mathcal{H}' = \binom{\mathcal{H}_2 \setminus \{\mathcal{V}(\mathcal{H})\}}{\mathcal{H}_1}$. Notons que $\mathcal{H}'$ est acyclique par définition avec $\mathcal{V}(\mathcal{H}) = \mathcal{V}(\mathcal{H}_1) = \mathcal{V}(\mathcal{H}')$, et que $\mathcal{H}'$ n'a pas non plus d'arête complète rouge. L'hypothèse de récurrence nous indique donc qu'il a deux feuilles x et y qui ne sont pas strictement voisines. Comme $\mathcal{H}$ et $\mathcal{H}'$ ont les mêmes feuilles, ce sont aussi des feuilles de $\mathcal{H}$. De plus, $\mathcal{H}$ et $\mathcal{H}'$ ont la même relation de voisinage strict, et donc x et y ne sont pas des voisins stricts. On a montré la propriété dans ce cas, on se place désormais dans le cas contraire, où $\mathcal{H}$ n'a aucune arête complète. On peut donc faire l'amalgame dans $\mathcal{H}$ entre voisinage et voisinage strict.

On va montrer le fait suivant : si un sommet x est une feuille de l'hypergraphe simple $\mathcal{H}_1 \cup \mathcal{H}_2$, alors on peut trouver un sommet y qui est une feuille de l'hypergraphe bicolore $\mathcal{H}$ et qui n'est pas un voisin strict de x. On suppose donc que l'on a une feuille x de $\mathcal{H}_1 \cup \mathcal{H}_2$. Alors il existe une arête e_x qui inclut toutes les arêtes de $\mathcal{H}\langle x\rangle$. Considérons $\mathcal{H}' = \mathcal{H}[\setminus\{x\}]$. Comme e_x est incomplète dans $\mathcal{H}$, et que $x \in e_x$, alors $e_x \setminus \{x\}$ est incomplète dans $\mathcal{H}'$. L'hypergraphe $\mathcal{H}'$ est acyclique et donc, par récurrence, *ou bien* il contient une arête complète rouge *ou bien* deux feuilles qui ne sont pas des voisins stricts :

- Supposons que $\mathcal{H}'$ contient une arête rouge complète. Choisissons $y \in \mathcal{V}(\mathcal{H}') \setminus e_x$. Par définition de e_x, les sommets x et y ne sont pas voisins dans $\mathcal{H}$. Il s'ensuit que $\mathcal{H}\langle y\rangle = \mathcal{H}'\langle y\rangle$. Comme y est trivialement une feuille de $\mathcal{H}'$, c'en est une de $\mathcal{H}$, qui n'est pas voisine de x.
- Supposons que $\mathcal{H}'$ ne contient pas d'arête rouge complète et possède deux feuilles y et z non strictement voisines. Comme y et z ne sont pas strictement voisines, et que $e_x \setminus \{x\}$ est incomplète dans $\mathcal{H}'$, au moins l'un des deux n'est pas inclus dans $e_x \setminus \{x\}$, par exemple y. Ce sommet y n'est donc pas voisin de x dans $\mathcal{H}$, il s'ensuit que $\mathcal{H}\langle y\rangle = \mathcal{H}'\langle y\rangle$, et donc y est aussi une feuille de $\mathcal{H}$, qui n'est pas voisine de x.

Dans les deux cas, $\mathcal{H}$ a donc bien une feuille y non (strictement) voisine de x. Il nous reste à faire usage de ce fait. Pour ce, on a besoin d'une feuille de l'hypergraphe acyclique $\mathcal{H}_1 \cup \mathcal{H}_2$, dont on prouve au point

suivant [2] l'existence. Par le fait précédent, on peut trouver y une feuille de $\mathcal{H}$. y est en particulier une feuille de $\mathcal{H}_1 \cup \mathcal{H}_2$, donc on peut en déduire à nouveau par le fait précédent que l'on peut trouver z une feuille non (strictement) voisine de y, ce qui conclut la récurrence.

Démonstration ($\mathcal{H}_1 \cup \mathcal{H}_2$ a une feuille). On prouve que $\mathcal{H}_1 \cup \mathcal{H}_2$ a une feuille. Supposons $\mathcal{H}_2 \neq \emptyset$. Alors $\binom{\emptyset}{\mathcal{H}_1 \cup \mathcal{H}_2}$ est un hypergraphe bicolore acyclique plus petit que $\mathcal{H}$ (de par la pondération des arêtes noires), qui a une feuille x, qui est une feuille de l'hypergraphe $\mathcal{H}_1 \cup \mathcal{H}_2$, ce que l'on voulait. Sinon, $\mathcal{H}_2 = \emptyset$, en particulier, $\mathrm{M}(\mathcal{H}) = \binom{\emptyset}{\mathrm{M}(\mathcal{H}_1)}$. Si $\mathcal{H} \neq \mathrm{M}(\mathcal{H})$, alors $\mathrm{M}(\mathcal{H})$ est un hypergraphe bicolore acyclique plus petit que $\mathcal{H}$, qui a donc deux feuilles qui sont aussi des feuilles de $\mathcal{H}$, en particulier $\mathcal{H}$ a une feuille, ce que l'on voulait. Dans le cas contraire, $\mathrm{M}(\mathcal{H}_1) = \mathcal{H}_1$. De plus, si deux sommets x et y sont indistinguables dans $\mathcal{H}_1$, i.e. $\mathcal{H}_1\langle x\rangle = \mathcal{H}_1\langle y\rangle$, alors par récurrence $\mathcal{H}[\searrow\{y\}]$ est acyclique et a une feuille, qui est une feuille de $\mathcal{H}$. On suppose désormais que $\mathcal{H}$ n'a pas de feuille, et on introduit le graphe orienté :

$$\mathcal{G} = \{s \to t \mid t \in \mathcal{L}(\mathcal{H}[\searrow\{s\}])\}$$

Supposons qu'un sommet x ait n prédécesseurs $x_1,...,x_n$. Alors x est voisin [3] de chacun des x_i. Dans $\mathcal{H}[\searrow\{x_i\}]$, x est une feuille donc il existe une arête qu'on nomme e_i qui inclut tous les voisins de x dans $\mathcal{H}[\searrow\{x_i\}]$, c'est–à–dire tous les voisins de x dans $\mathcal{H}$ sauf x_i. On nomme e l'ensemble des voisins de x. Pour tout i, $e_i = e \searrow \{x_i\}$. Supposons qu'une e_i donnée soit présente sous la forme $e_i \cup \{x_i\}$ dans $\mathcal{H}$. Alors $e_i \cup \{x_i\}$ inclut aussi tous les voisins de x dans $\mathcal{H}$, donc x est une feuille de $\mathcal{H}$, contradiction. Donc chacun des e_i est présent en l'état dans $\mathcal{H}$. De plus $e \notin \mathcal{H}$. Si $n \geq 3$, e constitue une clique : e_1 est une clique et il faut montrer que tous les autres sommets sont voisins de x_1. L'arête e_2 nous apprend que tous les sommets de $e \searrow \{x_2\}$ sont voisins de x_1. L'arête e_3 prouve que x_1 et x_2 sont voisins. e est une clique non incluse dans une arête, donc $\mathcal{H}$ n'est pas acyclique, contradiction.

On a prouvé qu'un sommet de $\mathcal{G}$ n'a que deux prédécesseurs au plus. Et comme par hypothèse de récurrence, chaque sommet a deux successeurs au moins, il s'ensuit que chaque sommet a exactement deux prédécesseurs et deux successeurs.

Soit x un sommet. Il y a deux sommets y et z prédécesseurs de x. Le sommet x est porté au moins par les deux arêtes e_y et e_z, les deux arêtes portant x de taille maximale incluant resp. y et z. On a $e_z \searrow \{x\} \subseteq e_y$ et réciproquement. On a donc $e_y = \{y\} \cup e$ et $e_z = \{z\} \cup e$, où $e = e_y \cap e_z$. S'il existe une autre arête f portant x, incomparable nécessairement avec e_y et e_z, alors $e = e_y \searrow \{y\} \subset f$ et de même pour z. Alors, $e_y = e \cup \{y\}$, $e_z = e \cup \{z\}$, et $f = e \cup \{t,...\}$ qui sont bien deux à deux incomparables, et même e_z et f sont présentes dans $\mathcal{H}[\searrow\{y\}]$ et toujours incomparables, donc x n'est pas une feuille de $\mathcal{H}[\searrow\{y\}]$, contradiction. Il s'ensuit que tout sommet x est porté exactement par deux arêtes, qui portent chacune un prédécesseur de x. Supposons qu'un autre sommet t soit contenu dans l'intersection des deux arêtes qui portent x. Il n'est porté par aucune autre arête, donc x et t sont indistinguables, contradiction. Il s'ensuit que tout sommet x est porté exactement par deux arêtes $\{x, y\}$ et $\{x, z\}$, qui portent chacune un des deux prédécesseurs y et z de x. Il s'ensuit

2 Ainsi, on sépare clairement la preuve de l'existence de feuilles généralisées multiples et celle (se ramènant aux graphes) de l'existence d'une alpha-feuille.

3 Le terme « voisin » signifiera *toujours* « voisin dans $\mathcal{H}$ » (et non dans $\mathcal{G}$).

que les successeurs et les prédécesseurs de x sont les mêmes sommets, donc $i \to j \;\Rightarrow\; j \to i$.

Finalement, $\mathcal{H}$ est un graphe non orienté classique où chaque sommet a deux voisins exactement. Ce graphe est donc cyclique, a fortiori non acyclique, contradiction. ◀

Remarque

À titre de comparaison, nous donnons la simplification de la preuve précédente dans le cas particulier du résultat de Brouwer et Kolen. La preuve suivante donne un résultat plus fort que le résultat de leur article [BK80], et ce, en peu de lignes. Pour que la comparaison soit pertinente, on admet, comme ils le font, qu'un hypergraphe α-acyclique a une α-feuille (un sommet isolé dans l'hypergraphe minimisé).

▶ **Théorème 36**

Tout hypergraphe β-acyclique $\mathcal{H}$ à deux sommets ou plus possède deux points d'imbrication non voisins dans $\mathcal{H} \setminus \{\mathcal{V}(\mathcal{H})\}$.

Démonstration. Supposons le théorème vrai pour tous les hypergraphes de taille n ou moins, et considérons un hypergraphe β–acyclique $\mathcal{H}$ de taille $n+1$. Si $\mathcal{H} = \{\mathcal{V}(\mathcal{H})\}$, le résultat est trivial, on suppose le contraire.

Supposons $\mathcal{V}(\mathcal{H}) \in \mathcal{H}$. Soit $\mathcal{H}' = \mathcal{H} \setminus \{\mathcal{V}(\mathcal{H})\}$, qui est donc un hypergraphe β–acyclique à moins un sommet. Si $\mathcal{V}(\mathcal{H}) = \mathcal{V}(\mathcal{H}')$, alors par récurrence il contient deux points d'imbrication non voisins dans $\mathcal{H}' \setminus \{\mathcal{V}(\mathcal{H}')\} = \mathcal{H} \setminus \{\mathcal{V}(\mathcal{H})\}$, ce que l'on voulait. Sinon (si $\mathcal{V}(\mathcal{H}') \neq \mathcal{V}(\mathcal{H})$), un sommet $x \in \mathcal{V}(\mathcal{H}) \setminus \mathcal{V}(\mathcal{H}')$ est uniquement porté par l'arête $\mathcal{V}(\mathcal{H})$ dans $\mathcal{H}$, c'est donc un point d'imbrication. Par ailleurs, $\mathcal{H}'$ a au moins un sommet ; s'il n'en a qu'un, c'est trivialement un point d'imbrication ; sinon, par récurrence, $\mathcal{H}'$ a des points d'imbrications. Dans les deux cas, $\mathcal{H}'$ a un point d'imbrication y, qui est aussi un point d'imbrication de $\mathcal{H}$.

On a montré que $\mathcal{H}$ possède deux points d'imbrications x et y trivialement non voisins dans $\mathcal{H} \setminus \mathcal{V}(\mathcal{H})$, ce qui clôt ce cas, on peut désormais supposer que $\mathcal{V}(\mathcal{H}) \notin \mathcal{H}$. De plus, si $\mathcal{H}$ n'a que deux sommets, alors on a $\mathcal{H} = \{\{x\}, \{y\}\}$ qui vérifie la propriété. On suppose désormais aussi que $\mathcal{H}$ a trois sommets ou plus.

On prouve le fait suivant : « s'il existe un sommet x tel que $\mathcal{H}\langle x\rangle$ admette un élément maximum pour l'inclusion e_x, alors il existe un point d'imbrication y de $\mathcal{H}$ qui n'est pas voisin de x dans $\mathcal{H} \setminus \{\mathcal{V}(\mathcal{H})\}$ ». Soit $\mathcal{H}' = \mathcal{H}[\setminus\{x\}]$. Comme $e_x \neq \mathcal{V}(\mathcal{H})$ (point précédent), $e_x \setminus \{x\} \neq \mathcal{V}(\mathcal{H}[\setminus\{x\}])$. Cet hypergraphe $\mathcal{H}'$ est β-acyclique, et a donc, par l'hypothèse de récurrence, deux points d'imbrication y et z non voisins dans $\mathcal{H}' \setminus \mathcal{V}(\mathcal{H}')$, en particulier dans l'arête $e_x \setminus \{x\}$ de $\mathcal{H}'$. L'un d'eux, mettons y, n'est pas contenu dans cette dernière, ni donc dans l'arête e_x de $\mathcal{H}$. Par définition de e_x, le sommet y n'est donc pas voisin de x dans $\mathcal{H}$. Par conséquent $\mathcal{H}\langle y\rangle = \mathcal{H}'\langle y\rangle$. Le sommet y est donc est un point d'imbrication de $\mathcal{H}$ qui n'est ni voisin de x dans $\mathcal{H}$, ni a fortiori dans $\mathcal{H} \setminus \{\mathcal{V}(\mathcal{H})\}$.

Exploitons ce fait : $\mathcal{H}$ est β–acyclique et donc α–acyclique. On peut donc trouver un sommet a tel que l'ensemble d'arêtes $\mathcal{H}\langle a\rangle$ admet un élément maximum. Le fait nous prouve l'existence d'un point d'imbrication b de $\mathcal{H}$, qui est trivialement tel que $\mathcal{H}\langle b\rangle$ admet un élément maximum pour l'inclusion. Encore par le fait, on peut trouver un point d'imbrication c non voisin de b dans $\mathcal{H} \setminus \{\mathcal{V}(\mathcal{H})\}$, ce qui conclut la récurrence. ◀

Corollaire

On a montré qu'un hypergraphe bicolore acyclique admet un ordre d'élimination. En remarquant que, pour un hypergraphe bicolore acyclique, l'existence d'un ordre d'élimination de cet hypergraphe implique celle d'un ordre d'élimination pour tout sous-hypergraphe induit, il est très facile de montrer l'équivalence entre l'acyclicité et l'existence d'un ordre d'élimination.

On annonçait plus haut que notre théorème prouve aussi le lemme 44, qui généralise le théorème 13. En effet, on a prouvé qu'un hypergraphe bicolore acyclique possède deux feuilles qui ne sont contenues simultanément dans aucune arête incomplète. Il s'ensuit donc (par une récurrence triviale) que l'on peut toujours trouver un ordre d'élimination qui préserve une arête donnée, c'est-à-dire qui élimine tous les sommets hors de cette arête avant d'éliminer ceux contenus dans cette arête, le résultat annoncé.

Conclusion de la seconde partie

Obsolescence

Le chapitre six rend quasiment obsolètes les chapitres quatre et cinq ; le chapitre sept élimine la dernière dépendance extérieure, simplifie considérablement la plus longue preuve du chapitre six, et rend inutile la généralisation de la notion de connexe-acyclicité définie dans [Bag09, BDG07]. On est désormais capable d'énoncer la totalité des faits de la seconde partie et de les prouver dans un article très court, qu'il reste bien évidemment à rédiger. Ce fait indique que notre recherche du chemin le plus court arrive à son terme : maintenant que les choses sont simples, rendons-les évidentes.

Le mérite de notre travail tient à notre avis bien plus aux faits que l'on peut désormais effacer qu'à la petite quantité de nouveaux faits que l'on peut écrire. Ceci prouve *a posteriori* la pertinence des notions manipulées, et l'inadéquation de certaines autres. Par exemple, les notations très efficaces introduites au chapitre quatre ne suffisent pas complètement à rendre simples les éléments de combinatoire, certains énoncés restent un peu hésitants. Le problème, c'est l'usage, banni dans les chapitres suivants, de la notion d'arbre de jointure. Des notions propres de feuille et d'ordre d'élimination permettront de montrer les mêmes résultats mais plus simplement et dans un cadre plus général.

L'accumulation de faits est inutile : leur totalité est déjà présente dans les axiomes. Il est donc plus constructif de remplacer des faits acceptés comme pertinents par des faits ayant un meilleur rapport pertinence/complexité de l'énoncé, et ce, sans introduire de notion nouvelle. Bref, il ne s'agit pas tant de construire que de choisir.

Vers une dichotomie pour le premier ordre existentiel ?

On a montré qu'une requête conjonctive signée est facile si et seulement si elle est acyclique (en donnant un sens naturel à cette acyclicité). Un corollaire immédiat est le suivant :

- Une requête du premier ordre existentiel est facile *si* on peut la mettre sous une forme normale disjonctive où chaque clause est une conjonction signée acyclique.

Les résultats portant sur le compte et sur l'énumération se propagent trivialement.

► **Conjecture**

Cette condition est également nécessaire : on peut remplacer le *si* ci-dessus par *si et seulement si*. ◄

Et l'énumération ?

On savait déjà que l'énumération d'une requête du premier ordre se fait à délai quasi-linéaire si et seulement si le problème de décision associé se fait en temps quasi-linéaire.

On propose un lien plus intime entre décision et énumération en prouvant que toute requête conjonctive signée de la forme

$$\mathcal{Q}((x_1,...,x_n) \mid \exists y_1...y_k \psi)$$

est énumérable à délai logarithmique après précalcul quasi-linéaire si et seulement si la requête conjonctive signée

$$?\mathcal{Q}(\psi \wedge \neg R(x_1,...,x_n))$$

est décidable en temps linéaire.

À la question de la pertinence du problème d'énumération, on peut finalement apporter la réponse suivante : en ce qui concerne le premier ordre existentiel, comme en ce qui concernait la logique propositionnelle, le problème d'énumération n'est pas moins pertinent que le problème de décision, ce qui n'est pas nécessairement la réponse mitigée à laquelle on aurait pu s'attendre.

Quelques éclaircissements

Autonomie des résultats de combinatoire

J'annonce dans le manuscrit atteindre l'autonomie pour ce qui est des résultats de combinatoire ; j'étaye ici mon propos. Voici donc la preuve que, effectivement, je n'ai pas besoin de la triple caractérisation standard de l'alpha-acyclicité.

On énonce dans ce manuscrit plusieurs caractérisations de l'acyclicité alpha :

- $P_1(\mathcal{H})$ $\mathcal{H}$ est conforme et sans cycle.
- $P_2(\mathcal{H})$ $\mathcal{H}$ est réductible à vide par la réduction de Graham/Yu et Ozoyoglu (réduction GYO)
- $P_3(\mathcal{H})$ $\mathcal{H}$ a un arbre de jointure
- $P_4(\mathcal{H})$ Il n'existe pas S tel que $\mathrm{M}(\mathcal{H}[S])$ est isomorphe à un graphe-cycle ou à un $\mathrm{Tetra}(k)$.
- $P_5(\mathcal{H})$ $\mathcal{H}$ est réductible à vide par retraits successifs d'alpha-feuilles (i.e. de sommets isolés dans $\mathrm{M}(\mathcal{H})$)

Les trois premières sont les caractérisations standards.

Noter que P_1 et P_4 sont proches : P_1 indique l'absence de « problème », P_4 indique l'absence de « problème canonique ».

Les propriétés P_2, P_3 et P_5 sont également proches entre elles : l'arbre de jointure peut être vu comme une représentation statique et condensée de l'application de la procédure GYO. La procédure GYO et la procédure d'effeuillage définie en P_5 sont des procédures non-déterministes qui sont cependant confluentes. La différence entre la procédure GYO et la procédure d'effeuillage tient à ce que la seconde peut trivialement être rendue déterministe, et qu'elle n'utilise qu'une seule opération.

Le lemme suivant se prouve *sans dépendance extérieure* (même si on utilise un résultat externe pour le montrer au chapitre 4).

▶ ***Lemme (lemme unique)*** Pour tout $i \in \{1,2,3,4,5\}$:

$P_i(\mathcal{H}) \Leftrightarrow P_i(\mathrm{M}(\mathcal{H}))$

$P_i(\mathcal{H}) \Rightarrow P_i(\mathcal{H}[S])$ pour tout S

Démonstration. Considérer les 5 cas un par un. Le lemme 17 traite un cas moins trivial que les autres ($i = 3$). L'autre cas pas complètement trivial est $i = 2$, que je dois pouvoir retrouver sur un vieux brouillon, si nécessaire. ◀

Ce qui est surtout intéressant, c'est de prouver l'équivalence entre P_1, P_4 et P_5 : ce sont les caractérisations les plus simples à manipuler et les plus utiles.

▶ **Théorème**

Les propriétés P_1, P_4 et P_5 sont équivalentes.

Démonstration. Le théorème 10 contient la preuve de l'équivalence de P_1 et P_4, qui doit utiliser le lemme unique et non son énoncé du chapitre 4 (qui utilise le résultat standard). L'implication $P_1 \Rightarrow P_4$ est alors évidente, et pour le sens retour ($\neg P_1 \Rightarrow \neg P_4$) la seule « astuce » consiste à choisir une clique non conforme *minimale*.

Le chapitre 7 annonce triomphalement donner le seul résultat de combinatoire qui manquait, et le fait en prouvant, en seconde partie de la preuve du théorème 35, qu'un hypergraphe alpha acyclique a une feuille. Il s'ensuit par le lemme unique que $P_1 \Rightarrow P_5$.

On peut conclure facilement en montrant, par exemple $P_5 \Rightarrow P_4$, encore à l'aide du lemme unique. ◀

Pertinence des notions introduites

J'annonçais aussi développer des formalismes plus adaptés aux problèmes traités.

La preuve ci-dessus doit vous suggérer fortement que l'induction $\mathcal{H}[S]$ et la minimisation $\mathrm{M}(\mathcal{H})$ sont décidément des outils efficaces.

Par ailleurs, la simplicité de la partie 2 de la preuve du théorème 35, qui établit qu'un hypergraphe alpha-acyclique a des feuilles ($P_1 \Rightarrow P_5$), appelle une comparaison avec la preuve de la caractérisation GYO (sa partie $P_1 \Rightarrow P_3$), que je n'ai pris le temps de chercher, et que je doute de trouver.

Je ne peux donc que vous demander de me croire si je vous dit que la caractérisation P_5 que je propose est plus pertinente que ne le sont les caractérisations standards P_2 et P_3.

Par ailleurs, ayant moi-même prouvé (j'ai gaspillé des mois et des pages) l'équivalence entre P_2 et P_3, je suis convaincu que le meilleur moyen de prouver leur équivalence est de prouver séparément $P_2 \Leftrightarrow P_5$ et $P_3 \Leftrightarrow P_5$. (en faisant, c'est original, un usage intensif du lemme unique ci–dessus)

De l'acyclicité bicolore

J'introduis cette notion qui vise à généraliser simultanément l'alpha et la bêta acyclicité, afin de caractériser les requêtes conjonctives signées faciles.

Pour bien la comprendre, une idée peut être de comprendre son usage : les arêtes dites noires correspondent au atomes négatifs, les arêtes dites rouges correspondent au atomes positifs. C'est une notion *asymétrique*, car les atomes positifs et les atomes négatifs jouent des rôles asymétriques dans une conjonction. Pour insister sur cet aspect asymétrique, j'ai appelé les arêtes noires et rouges (et non pas négatives et positives), et ai écrit l'hypergraphe bicolore sous forme verticale. En effet :

$$\mathcal{A}\binom{\mathcal{H}_2 \cup \mathcal{H}}{\mathcal{H}_1} \quad \Rightarrow \quad \mathcal{A}\binom{\mathcal{H}_2}{\mathcal{H}_1 \cup \mathcal{H}}$$

c'est-à-dire que les arêtes peuvent "tomber" sans effort, mais le contraire est faux : les arêtes ne peuvent pas "remonter" sans effort.

Pour travailler avec cette notion, qui est définie à partir de l'alpha-acyclicité, on généralise les notions très pratiques d'induction et de minimisation de telle sorte que le lemme unique ci-dessus soit maintenu pour cette propriété :

$\mathcal{A}(\mathcal{H}) \Leftrightarrow \mathcal{A}(\mathrm{M}(\mathcal{H}))$
$\mathcal{A}(\mathcal{H}) \Rightarrow \mathcal{A}(\mathcal{H}[S])$ pour tout S

De la sorte, on peut généraliser facilement les différentes caractérisations de l'acyclicité alpha à cette notion.

Résultats, résumés en 5 lignes

Première partie : La dichotomie de Creignou et Hébrard (variante de celle de Schaefer, plus pertinente que l'original) était trop timide : au lieu de délai polynomial, il fallait parler de délai linéaire après temps polynomial de décision.

Seconde partie :

- Résultat théorique : Une SCQ est décidable en temps linéaire (à formule fixée) ssi elle est acyclique (dans un sens précis) (+variante pour l'énumération)
- Résultat pratique : Un algorithme de décision et d'énumération des SCQ acycliques d'une complexité *combinée* quasi-optimale.
- Résultat supplémentaire : La combinatoire sous-jacente a été entièrement résolue.

Table des matières

I Logique propositionnelle 1

1 Logique propositionnelle et énumération 5

1 Logique propositionnelle 6
- 1.1 De la logique propositionnelle aux formules CNF . . . 7
- 1.2 Satisfaisabilité et énumération 8
- 1.3 Des cas particuliers faciles 10

2 Satisfaisabilité généralisée 12
- 2.1 Décision : dichotomie de Schaefer 12
- 2.2 Énumération : dichotomie de Creignou & Hébrard . . 13

3 Des formules affines 14
- 3.1 Définitions 15
- 3.2 Décision et précalcul 15
- 3.3 Énumération 17

2 Formules bijonctives et Horn-renommages 19

1 Des formules bijonctives 20
- 1.1 Décision et précalcul 20
  - 1.1.1 Décision et première solution 20
  - 1.1.2 Suite du précalcul : normalisation 22
- 1.2 Énumération 22
  - 1.2.1 Approche type contraintes (naïve) 23
  - 1.2.2 Parcours de l'arbre de recherche 25
  - 1.2.3 Finalement 27

2 Des formules Horn-renommables 28
- 2.1 Décision 28
  - 2.1.1 Définitions 29
  - 2.1.2 Réduction de HornRen à Auplus1Sat 29
  - 2.1.3 Réduction linéaire de Auplus1Sat à 2Sat 30
- 2.2 Forme canonique 31
  - 2.2.1 Simplifions 31
  - 2.2.2 Réduction et composantes fortement connexes 32
- 2.3 Énumération 33
  - 2.3.1 Énumération « naïve » 34
  - 2.3.2 Énumération par extension 34
  - 2.3.3 Énumération par réduction 35
  - 2.3.4 Combinaison des différents algorithmes 35

3 Models of Horn Formulas are Enumerable at Linear Delay 39

1 A First Approach 40
- 1.1 Algorithm Overview 40
- 1.2 First Complexity Considerations 42

2 Incremental Circuit Building 42

3 The Final Algorithm 43

3.1 Using an Union-Find Algorithm 43
3.2 The Final Algorithm 44

II Logique du premier ordre 49

4 About Conjunctive Queries 53

1 Optimal Enumeration Class and Reduction 55
1.1 Optimal Enumeration Class 55
1.2 A Reduction 58
2 Properties of Queries Classes 61
2.1 Queries Definition and Basic Properties 61
2.1.1 Definitions and Examples 61
2.1.2 Class Invariance Property 64
2.1.3 Closure Property 66
2.2 Conjunctive Queries 66
2.2.1 Definition 66
2.2.2 Class Invariant: Formula Hypergraph 67
2.2.3 Class Properties Written with Hypergraphs . . 68
2.3 Additional Properties and Summary 70
2.3.1 Existential Conjunctive Queries Minor Closure 70
2.3.2 Summary 71
2.3.3 Queries Complexity Classes 71
3 Conjunctive Queries and Acyclicity 72
3.1 Quantifier-Free Conjunctive Queries and Acyclicity . . 72
3.1.1 Acyclicity: Definition and Properties 72
3.1.2 Consequences on Quantifier-Free Conjunctive Queries 76
3.2 Existential Conjunctive Queries and Acyclicity-based Properties 79
3.2.1 Starring Property 80
3.2.2 Ordering Property 82
3.3 The Whole Picture 84
3.3.1 The Trivial Class 84
3.3.2 The Whole Picture 86
3.3.3 Conjunctive Queries 88

5 A Negative Conjunctive Query is Easy iff it is Beta-Acyclic 89

1 Preliminaries and Results 91
1.1 Preliminaries 91
1.2 Acyclicity Notions 93
1.3 Statement of the Results 94
2 Davis-Putnam Resolution w.r.t a Nest Point 95
2.1 Definition and Properties 95
2.2 Algorithm and Complexity 96
3 Easiness Result 97
3.1 NCQ on the Boolean Domain 98
3.2 Easiness Result for SCQ on the Boolean Domain . . . 99
3.3 Final Easiness Result 101
4 Hardness Result 102
4.1 A Technical Point 102
4.2 Hardness Result 103

6 Dichotomies pour les requêtes conjonctives signées. 105

1 Acyclicité des hypergraphes 106
1.1 Rappels 106

1.2 Des hypergraphes bicolores 107
1.3 Un hypergraphe bicolore acyclique a des feuilles. . . . 110
2 Des requêtes 116
2.1 Décision 116
2.1.1 Facilité 116
2.1.2 Difficulté 117
2.1.3 Dichotomie 118
2.2 Énumération 119
2.2.1 Facilité 119
2.2.2 Difficulté 124
2.2.3 Dichotomie 125

7 Encore un peu de combinatoire ? 127

Table des matières 139

Définitions, lemmes et théorèmes 143

Liste des algorithmes et figures 147

Bibliographie 149

Définitions et théorèmes

Définitions

Définition 1 formule propositionnelle . . . 6
Définition 2 CNF, clause, litteral, modèle . . . 8
Définition 3 énumération . . . 9
Définition 4 clause unitaire, littéral pur . . . 9
Définition 5 classes de CNF . . . 10
Définition 6 classes de structures . . . 12
Définition 7 formule affine . . . 15
Définition 8 code de Gray . . . 17
Définition 9 Horn-renommage . . . 29
Définition 10 Horn-renommage . . . 29
Définition 11 Auplus1 . . . 29
Definition 12 ConstD . . . 55
Definition 13 ConstD∘Lin . . . 56
Definition 14 optimal, easy, tractable . . . 57
Definition 15 nice reduction $\prec$ . . . 58
Definition 16 nice reduction $\prec$ (cont.) . . . 59
Definition 18 signature, structure . . . 61
Definition 19 query . . . 61
Definition 20 answering a query . . . 62
Definition 21 nice, simple, sorted . . . 64
Definition 22 Conjunctive Queries . . . 66
Definition 23 hypergraph notations . . . 67
Definition 24 hypergraph induction . . . 68
Definition 25 minimization $r_e^\star$ . . . 69
Definition 26 edge contraction . . . 70
Definition 27 pseudo-minor . . . 70
Definition 28 r_e, r_v, $r^\star$ . . . 72
Definition 29 elimination order . . . 72
Definition 30 standard definitions . . . 73
Definition 31 Tetra– and Cycle–graphs . . . 75
Definition 32 Tetra problem . . . 77
Definition 33 starred hypergraph . . . 80
Definition 34 decision hardness hypotheses . . . 82
Definition 35 starred query . . . 82
Definition 36 ordered hypergraph . . . 82
Definition 37 order hypothesis . . . 83
Definition 38 trivial hypergraph . . . 84
Definition 39 trivial query . . . 84
Definition 40 sentence, structure, query, CQ, NCQ, SCQ . . . 91
Definition 41 complexity classes . . . 92
Definition 42 hypergraph of a query . . . 92

Definition 43 induced hypergraph, nest point . . . 93
Definition 44 β-acyclicity . . . 93
Definition 45 β-cycle . . . 94
Definition 46 CNF formula–related . . . 95
Definition 47 NCQ-BoolD, SCQ-BoolD . . . 98
Definition 48 linear reduction $\prec$ . . . 102
Definition 49 simple, sorted, normalized NCQ . . . 102
Définition 50 hypergraphe . . . 106
Définition 51 Tetra, Cycle . . . 106
Définition 52 ordre d'élimination . . . 107
Définition 53 hypergraphe bicolore . . . 107
Définition 54 ordre d'élimination . . . 109
Définition 55 hypothèse $\mathrm{H}_d(k)$ . . . 117
Définition 56 hypothèse H_T . . . 117
Définition 58 hypothèse qH_T . . . 124
Définition 59 hypothèse H_e . . . 124
Définition 60 hypergraphe . . . 127
Définition 61 hypergraphe bicolore . . . 128
Définition 62 voisinage strict . . . 128

Théorèmes et lemmes

Théorème 1 facilité de certaines classes de CNF . . . 10
Théorème 2 Schaefer [Sch78] . . . 12
Théorème 3 Creignou et Hébrard [CH97] . . . 13
Lemme 1 HornRen = Auplus1Sat . . . 29
Lemme 2 réduction linéaire . . . 30
Théorème 4 Horn-renommage [Héb94] . . . 31
Théorème 5 énumération des Horn-renommages (adaptation) 34
Théorème 6 énumération des Horn-renommages (synthèse) . 35
Theorem 7 [Gra96] . . . 57
Lemma 14 minimization equivalence . . . 69
Lemma 16 pseudo-minor . . . 70
Theorem 8 equivalence classes . . . 71
Theorem 9 invariance and closure . . . 71
Theorem 10 standard result . . . 73
Lemma 17 cleaning . . . 73
Theorem 11 acyclicity . . . 75
Theorem 12 QFCQ conditional dichotomies . . . 78
Lemma 21 glue . . . 80
Theorem 13 S-starred . . . 80
Theorem 14 ECQ dichotomy . . . 82
Theorem 15 ordering . . . 82
Theorem 16 ordered enumeration dichotomies . . . 83
Theorem 17 trivial . . . 84
Theorem 18 trivial conjunction . . . 85
Theorem 19 trivial query . . . 86
Lemma 27 β-acyclicity inductive characterisation . . . 93
Lemma 28 β-acyclicity as absence of β-cycles . . . 94
Theorem 20 dichotomy . . . 94
Theorem 21 easiness . . . 94
Lemma 29 resolvent properties . . . 95
Lemma 30 resolvent computation . . . 96

Lemma 31 NCQ-BoolD nest point . . . 98
Lemma 32 NCQ-BoolD easiness . . . 99
Lemma 34 SCQ-BoolD easiness . . . 100
Lemma 36 SCQ easiness . . . 101
Lemma 37 linear reduction properties . . . 102
Corollary 22 hypergraph as only relevant aspect . . . 103
Corollary 23 induced query . . . 103
Lemma 41 hardness result . . . 103
Théorème 24 caractérisations de l'acyclicité α . . . 106
Théorème 25 résultat principal . . . 109
Lemme 44 ordre d'élimination . . . 110
Corollaire 26 généralisation du théorème 13 . . . 110
Théorème 27 feuilles . . . 111
Théorème 28 complexité du compte . . . 116
Théorème 29 dichotomies pour la décision . . . 119
Théorème 30 élimination des quantificateurs . . . 120
Lemme 53 énumération naïve . . . 121
Théorème 31 facilité de l'énumération . . . 122
Théorème 32 dichotomie pour l'énumération . . . 125
Corollaire 33 facilité des requêtes ∃FO β–acycliques . . . 125
Corollaire 34 lien entre décision et énumération . . . 126

Remarques

Remarque 2 importance de l'ordre . . . 11
Remarque 3 de l'algo de Creignou et Hébrard . . . 13
Remark 15 about symbol diversity . . . 65
Remark 18 on domains . . . 67
Remarque 34 extension de l'acyclicité . . . 107
Remarque 35 test d'acyclicité . . . 109
Remarque 36 canonicité des hypothèses . . . 117
Remarque 37 inutilité de la notion d'étoile . . . 125

Liste des algorithmes et figures

Liste des algorithmes

1.1 Algorithme de Davis et Putnam (énumération) 10
1.2 Algorithme de Creignou et Hébrard 14
1.3 Décision et énumération des formules affines 16

2.1 Construction des Composantes Fortement Connexes . . . 20
2.2 Remplacement du domaine par l'activité 24
2.3 Parcours avec contraction des branches unaires 24
2.4 Parcours n-aire avec production à chaque noeud 26
2.5 Algorithme final 26
2.6 Énumération dans l'ordre lexicographique à délai $\mathcal{O}(n)$ . . 28
2.7 Énumération des Horn-renommages 36

3.1 Basic algorithm 41
3.2 Incremental circuit building 42
3.3 Final algorithm 44

4.1 Decision of acyclic existential conjunctive queries 77
4.2 Count and enumeration for acyclic QFCQ 78
4.3 Jeth for acyclic quantifier-free conjunctive queries 79

5.1 Resolvent computation 96
5.2 SCQ-BoolD decision algorithm 100

6.1 Élimination de quantificateurs pour les SCQ – BoolD . . . 119
6.2 Énumération des SCQ – BoolD 123

Table des figures

1.1 Transformation d'une formule propositionnelle en CNF . . 7
1.2 Dichotomie de Schaefer 13
1.3 Dichotomie de Creignou et Hébrard 14

2.1 Exemple de calcul de composantes fortement connexes . . 21
2.2 Exemple d'exécution de l'algorithme naïf 23
2.3 Exécution des algorithmes 2.2, 2.3 et 2.4 25
2.4 Exemple de normalisation 33
2.5 Étude de cas exhaustive 38

4.1 Illustration of Definition 30 73
4.2 Acyclicity Vs non-acyclicity 74
4.3 Illustration of Theorem 11 76
4.4 Illustration of Theorem 12 76

4.5 The whole picture of ECQ problems complexity 87

6.1 Support d'une feuille généralisée 108

Bibliographie

[AGK12] Isolde Adler, Tomás Gavenciak, and Tereza Klimosova. Hypertree-depth and minors in hypergraphs. *Theor. Comput. Sci.*, 463:84–95, 2012.

[AHU74] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas and its evaluation. *Information Processing Letters*, 8:121–123, 1979.

[Asp80] Bengt Aspvall. Recognizing disguised nr(1) instances of the satisfiability problem. *J. Algorithms*, 1(1):97–103, 1980.

[AV82] B. Devadas Acharya and Michel Las Vergnas. Hypergraphs with cyclomatic number zero, triangulated graphs, and an inequality. *Journal of Combinatorial Theory, Series B*, 33(1):52 – 56, 1982.

[Bag06] Guillaume Bagan. Mso queries on tree decomposable structures are computable with linear delay. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[Bag09] Guillaume Bagan. *Algorithmes et complexité des problèmes d'énumération pour l'évaluation de requêtes logiques*. Thèse de doctorat, Université de Caen Basse-Normandie, 2009.

[BB12a] Johann Brault-Baron. A Negative Conjunctive Query is Easy if and only if it is Beta-Acyclic. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 137–151, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[BB12b] Johann Brault-Baron. Models of horn formulas are enumerable at linear delay. Submitted to Information Processing Letters on May, under review, 2012.

[BCF12] Manuel Bodirsky, Hubie Chen, and Tomás Feder. On the complexity of mmsnp. *SIAM J. Discrete Math.*, 26(1):404–414, 2012.

[BCRV03] Elmar Böhler, Nadia Creignou, Steffen Reith, and Heribert Vollmer. Playing with boolean blocks, part 1 : Post's lattice with applications to the complexity theory. 2003.

[BCRV04] Elmar Böhler, Nadia Creignou, Steffen Reith, and Heribert Vollmer. Playing with boolean blocks, part 2: Constraint satisfaction problems. *ACM SIGACT-Newsletter*, 35, 2004.

[BDG88] José Luis Balcázar, Josep Días, and Joaquim Gabarró. *Structural Complexity*, volume I and II. Springer-Verlag, monograph on theoretical computer science edition, 1988.

[BDG07] Guillaume Bagan, Arnaud Durand, and Étienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. *Computer Science Logic*, 4646:208–222, 2007.

[BDGO08] Guillaume Bagan, Arnaud Durand, Etienne Grandjean, and Frédéric Olive. Computing the jth solution of a first-order query. *ITA*, 42(1):147–164, 2008.

[Ber69] Claude Berge. *Graphes et Hypergraphes.* Dunod, 1969.

[BFMY83] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *Journal of the ACM*, 30(3):479–513, 1983.

[BFS95] Kenneth A. Berman, John V. Franco, and John S. Schlipf. Unique satisfiability of horn sets can be solved in nearly linear time. *Discrete Applied Mathematics*, 60(1-3):77–91, 1995.

[BG82] Andreas Blass and Yuri Gurevich. On the unique satisfiability problem. *Information and Control*, 55(1-3):80–88, 1982.

[BG02] Régis Barbanchon and Etienne Grandjean. Local problems, planar local problems and linear time. In Julian C. Bradfield, editor, *Computer Science Logic, 16th International Workshop, CSL 2002, 11th Annual Conference of the EACSL, Edinburgh, Scotland, UK, September 22-25, 2002, Proceedings*, volume 2471 of *Lecture Notes in Computer Science*, pages 397–411. Springer, 2002.

[BG04] Régis Barbanchon and Étienne Grandjean. The minimal logically-defined np-complete problem. In *Proc. Symposium on Theoretical Aspect of Computer Science (STACS'04)*, 2004.

[BK80] Andries E. Brouwer and Antoon W. J. Kolen. A superbalanced hypergraph has a nest point. *Technical report, Math. centr. report ZW146, Amsterdam*, 1980.

[BL99] Hans Kleine Büning and Theodor Lettman. *Propositional Logic: Deduction and Algorithm.* Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1999.

[BM08] John Adrian Bondy and U.S.R. Murty. *Graph Theory, Graduate Texts in Mathematics*. Springer, 2008.

[BP11] Manuel Bodirsky and Michael Pinsker. Schaefer's theorem for graphs. In Lance Fortnow and Salil P. Vadhan, editors, *STOC*, pages 655–664. ACM, 2011.

[CCH+90] Vijaya Chandru, Collette R. Coulard, Peter L. Hammer, Miguel Montanez, and Xiaorong Sun. On renamable horn and generalized horn functions. *Annals of Mathematics and Artificial Intelligence*, 1, 1990.

[CCHS10] Victor Chepoi, Nadia Creignou, Miki Hermann, and Gernot Salzer. The helly property and satisfiability of boolean formulas defined on set families. *Eur. J. Comb.*, 31(2):502–516, 2010.

[CE12] Bruno Courcelle and Joost Engelfriet. *Graph structure and monadic second-order logic, a language theoretic approach*. Cambridge University Press, 2012.

[CH96] Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996.

[CH97] Nadia Creignou and Jean-Jacques Hébrard. On generating all solutions of generalized satisfiability problems. *ITA*, 31(6):499–511, 1997.

[Cha05] Philippe Chapdelaine. On the structure of linear-time reducibility. Technical Report, 2005.

[CK73] C.C. Chang and H.J. Keisler. *Model Theory*. North-Holland, Amsterdam, 1973.

[CKS01] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean Constraint Satisfaction Problems*. Peter Hammer, SIAM Monographs on Discrete Mathematics and Applications, 2001.

[CL86] Gary Chartrand and Linda Lesniak. *Graphs and digraphs*. Wadsworth and Brooks, California, 1986.

[CLR91] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. Mc Graw Hill, 1991.

[CLZ10] Gary Chartrand, Linda Lesniak, and Ping Zhang. *Graphs & Digraphs (5th ed.)*. CRC Press, 2010.

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. *ACM Symp. on Theory of Computing*, pages 77–90, 1977.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *stoc71*, pages 151–158, 1971.

[Cou09] Bruno Courcelle. Linear delay enumeration and monadic second-order logic. *Discrete Applied Mathematics*, 157(12):2675–2700, 2009.

[Cre93] Nadia Creignou. *Temps linéaire et problèmes NP-complets*. PhD thesis, Université de Caen, France, 1993.

[DF99] R.G. Downey and Michael R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.

[DG07] Arnaud Durand and Étienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Trans. Comput. Log.*, 8(4), 2007.

[DHK05] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. *Theor. Comput. Sci.*, 340(3):496–513, 2005.

[Die10] Reinhard Diestel. *Graph theory*, volume 176. Springer-Verlag, Heidelberg, 2010.

[DLL62] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[DO06] Arnaud Durand and Frédéric Olive. First-order queries over one unary function. In LNCS, editor, *Proc. 15th Annual Conference of the EACSL (CSL'06)*, volume 4207, pages 334–348, 2006.

[DP60] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[DP89] Rina Dechter and Judea Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3):353–366, 1989.

[DS11] Arnaud Durand and Yann Strozecki. Enumeration complexity of logical query problems with second-order variables. In Marc Bezem, editor, *Computer Science Logic, 25th International Workshop / 20th Annual Conference of the EACSL, CSL 2011, September 12-15, 2011, Bergen, Norway, Proceedings*, volume 12 of *LIPIcs*, pages 189–202. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[Duc95] Pierre Duchet. Hypergraphs. In Ronald l. Graham, Martin Grötschel, and Làszlò Lovàsz, editors, *Handbook of combinatorics*. Elsevier, 1995.

[Dur08] David Duris. Some characterizations of gamma and beta-acyclicity of hypergraphs. November 2008.

[Dur09] David Duris. *Acyclicité des hypergraphes et liens avec la logique sur les structures relationnelles finies*. Thèse de doctorat, Université Paris Diderot - Paris 7, 2009.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer-Verlag, 1995.

[EIS76] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[Fag74] Ronald Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In Richard M. Karp, editor, *Complexity of Computation, SIAM-AMS Proceedings*, pages 43–73, 1974.

[Fag83] Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of the ACM*, 30:514–550, 1983.

[Fag93] Ronald Fagin. Finite-model theory–a personal perspective. *Theoretical Computer Science*, 116(1):3–31, 1993.

[Fed94] Tomàs Feder. Network flow and 2-satisfiability. *Algorithmica*, 11:291–319, 1994. 10.1007/BF01240738.

[FFG02] Jörg Flum, M. Frick, and Martin Grohe. Query evaluation via tree decompositions. *Journal of the ACM*, 49(6):716–752, 2002.

[FG04] Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM J. Comput.*, 33(4):892–922, 2004.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[FM09] John Franco and John Martin. A history of satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 3–74. IOS Press, Amsterdam, The Netherlands, 2009.

[FMU82] Ronald Fagin, Alberto O. Mendelzon, and Jeffrey D. Ullman. A simplified universal relation assumption and its properties. *ACM Trans. Database Syst.*, 7(3):343–360, 1982.

[FV98] Tomás Feder and Moshe Y. Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[Gai82] Haim Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North Holland, 1982.

[GI91] Zvi Galil and Giuseppe F. Italiano. Data structures and algorithms for disjoint set union problems. *ACM Computing Surveys*, 23(3):319–344, 1991.

[Gib85] Alan Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[GJ79] Michael R. Garey and Davis S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1979.

[GKL+07] Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Martin Marx, Joel Spencer, Moshe Y. Vardi, Yde Venema, and Scott Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. Springer, 2007.

[GKS04] Georg Gottlob, Phokion G. Kolaitis, and Thomas Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. *J. ACM*, 51(2):312–362, 2004.

[GLS01] Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

[GLS02] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.

[GO04] Etienne Grandjean and Frédéric Olive. Graph properties checkable in linear time in the number of vertices. *J. Comput. Syst. Sci.*, 68(3):546–597, 2004.

[GPFW99] Jun Gu, Paul W. Purdom, John Franco, and Benjamin Wah. *Algorithms for the Satisfiability (SAT) Problem*. 1999.

[Gra79] Marc H. Graham. On the universal relation. Technical report, University of Toronto, september, 1979.

[Grä90] Erich Grädel. On the notion of linear time computability. *International Journal of Fundations of Computer Sciences*, 1:295–307, 1990.

[Grä92] Erich Grädel. Capturing complexity classes by fragments of second-order logic. *Theor. Comput. Sci.*, 101(1):35–57, 1992.

[Gra94a] Étienne Grandjean. Invariance properties of RAM's and linear time. *Computational Complexity*, 4:62–106, 1994.

[Gra94b] Étienne Grandjean. Linear time algorithms and NP-complete problems. *SIAM Journal on Computing*, 23:573–597, 1994.

[Gra96] Étienne Grandjean. Sorting, Linear Time and the Satisfiability Problem. *Ann. Math. Artif. Intell.*, 16:183–236, 1996.

[GS89] Yuri Gurevich and Saharon Shelah. Nearly linear time. In Albert R. Meyer and Michael A. Taitslin, editors, *Logic at Botik '89, Symposium on Logical Foundations of Computer Science, Pereslav-Zalessky, USSR, July 3-8, 1989, Proceedings*, volume 363 of *Lecture Notes in Computer Science*, pages 108–118. Springer Berlin / Heidelberg, 1989.

[GS02] Étienne Grandjean and Thomas Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM Journal on Computing*, 32(1):196–230, 2002.

[GSS01] Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 657–666, 2001.

[Héb94] Jean-Jacques Hébrard. A linear algorithm for renaming a set of clauses as a horn set. *Theoretical Computer Science*, 124(2):343 – 350, 1994.

[Héb95] Jean-Jacques Hébrard. Unique horn renaming and unique 2-satisfiability. *Inf. Process. Lett.*, 54(4):235–239, 1995.

[HJ85] Pierre Hansen and Brigitte Jaumard. Uniquely solvable quadratic boolean equations. *Discrete Applied Mathematics*, 12:147–154, 1985.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[IM82] Alon Itai and Johann A. Makowsky. On the complexity of herbrand's theorem. Technical Report 243, Dpt of Computer Science, Israel Institute of Technology, 1982.

[Imm89] Neil Immerman. Descriptive and computational complexity. In J. Hartmanis, editor, *Computational Complexity Theory, Proc. Symp. Applied Math., Vol. 38*, pages 75–91. American Mathematical Society, 1989.

[Imm99] Neil Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer, 1999.

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. On generating all maximal independent sets. *Inf. Process. Lett.*, 27(3):119–123, 1988.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computers Computations*, IBM Symp. 1972. Plenum Press, New York, 1972.

[Kol03] Phokion G. Kolaitis. Constraint satisfaction, databases, and logic. In *IJCAI*, pages 1587–1595, 2003.

[Koz94] Dexter Kozen. *Automata and Computability.* Springer-Verlag, New York, NY, 1994.

[KS] Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Transactions on Computational Logic.* To appear.

[KS11] Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7(2), 2011.

[Lev73] Leonid Levin. Universal search problems (russian). *Problems of Information Transmission (Russian:Problemy Peredachi Informatsii)*, 9(3):265–266, 1973. Translated into English in [Tra84].

[Lew78] Harry R. Lewis. Renaming a set of clauses as a horn set. *J. ACM*, 25(1):134–135, 1978.

[Lib04] Leonid Libkin. *Elements of Finite Model Theory.* Springer, 2004.

[Lin08] Steven Lindell. A normal form for first-order logic over doubly-linked data structures. *Int. J. Found. Comput. Sci.*, 19(1):205–217, 2008.

[Lov06] Làszló Lovàsz. Graph minor theory. In *Bulletin of the American Mathematical Society*, volume 43, pages 75–86, 2006.

[LS89] Greg Lindhorst and Farhad Shahrokhi. On renaming a set of clauses as a horn set. *Inf. Process. Lett.*, 30(6):289–293, 1989.

[Min92] Michel Minoux. The unique horn-satisfiability problem and quadratic boolean equations. *Annals of Mathematics and Artificial Intelligence*, 6:253–266, 1992. 10.1007/BF01531031.

[MM85] Heikki Mannila and Kurt Mehlhorn. A fast algorithm for renaming a set of clauses as a horn set. *Inf. Process. Lett.*, 21(5):269–272, 1985.

[OPS10] Sebastian Ordyniak, Daniel Paulusma, and Stefan Szeider. Satisfiability of Acyclic and Almost Acyclic CNF Formulas. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 84–95, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pre93] Daniele Pretolani. A linear time algorithm for unique horn satisfiability. *Information Processing Letters*, 48(2):61–66, 1993.

[PY99] Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.

[Reg93] Kenneth W. Regan. Machine models and linear time complexity. In *SIGACT News*, volume 24, Fall 1993.

[RM12] Kristoffer H. Rose and Ross Moore. *Xy-Pic Reference Manual*. 2012.

[Ros99] Kristoffer H. Rose. *Xy-pic User's Guide*. 1999.

[Sch78] Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 216–226, New York, NY, USA, 1978. ACM.

[Sch97] Thomas Schwentick. Algebraic and logical characterizations of deterministic linear time classes. In *Proc. 12th Symposium on Theoretical Aspect of Computer Science (STACS'97)*, pages 463–474, 1997.

[See96] Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, December 1996.

[Sip97] Michael Sipser. *Introduction to the theory of computation*. PWS Publishing Company, 1997.

[Str10] Yann Strozecki. *Enumeration complexity and matroid decomposition*. Thèse de doctorat, Université Paris Diderot - Paris 7, 2010.

[Tar71] Robert Endre Tarjan. Depth-first search and linear graph algorithms. In *SWAT (FOCS)*, pages 114–121, 1971.

[Tar72] Robert Endre Tarjan. Depth-first search and linear time algorithms. *SIAM Journal of Computing*, 1:146–160, 1972.

[Tar75] Robert Endre Tarjan. Efficiency of a good but not linear set union algorithm. *J. ACM*, 22(2):215–225, 1975.

[Tra84] Boris A. Trakhtenbrot. A survey of russian approaches to perebor (brute-force searches) algorithms. *Annals of the History of Computing*, 6(4):384–400, 1984.

[TY84] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

[TY85] Robert Endre Tarjan and Mihalis Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 14(1):254–255, 1985.

[Uno98] Takeaki Uno. A new approach for speeding up enumeration algorithms. In Kyung-Yong Chwa and Oscar H. Ibarra, editors, *Algorithms and Computation, 9th International Symposium, ISAAC '98, Taejon, Korea, December 14-16, 1998, Proceedings*, volume 1533 of *Lecture Notes in Computer Science*, pages 287–296. Springer, 1998.

[Uno01] Takeaki Uno. A fast algorithm for enumerating bipartite perfect matchings. In Peter Eades and Tadao Takaoka, editors, *Algorithms and Computation, 12th International Symposium, ISAAC 2001, Christchurch, New Zealand, December 19-21, 2001, Proceedings*, volume 2223 of *Lecture Notes in Computer Science*, pages 367–379. Springer, 2001.

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[Weg02] Ingo Wegener. A simplified correctness proof for a well-known algorithm computing strongly connected components. *Inf. Process. Lett.*, 83(1):17–19, 2002.

[Yan81] Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings Conf. on Very Large Databases*, pages 82–94, 1981.

[YO79] Clement T. Yu and Meral Z. Özsoyoglu. An algorithm for tree-query membership of a distributed query. pages 306–312, 1979.

[ZH02] Bruno Zanuttini and Jean-Jacques Hébrard. A unified framework for structure identification. *Information Processing Letters*, 81(6):335 – 339, 2002.

Résumé

Au-delà de la décision de problèmes de satisfaisabilité, on s'intéresse à la génération exhaustive de leurs solutions, l'énumération.

Nous interrogeons d'abord la pertinence du problème d'énumération dans le cadre très classique de la logique propositionnelle. La dichotomie de Creignou et Hébrard prouve déjà l'équivalence entre les classes polynomiales pour la décision non triviale et celles pour l'énumération. On donne des algorithmes d'énumération optimaux pour chacune de ces classes, qui généralisent tout algorithme de décision non triviale, suggérant que l'énumération est *le* problème pertinent dans ce cadre.

Ensuite, nous complétons et simplifions des résultats de dichotomie de Bagan et al. qui établissent un lien étroit entre la facilité d'une requête conjonctive et une notion d'acyclicité d'hypergraphe. On prouve alors, grâce à un nouvel algorithme, des résultats similaires pour la classe duale de celle des requêtes conjonctives. Finalement, en généralisant le résultat classique de combinatoire de Brouwer et Kolen, on unifie l'ensemble de ces résultats sous forme d'*une* dichotomie pour l'énumération des requêtes conjonctives dites signées, qui établit un lien fort entre facilité de l'énumération et facilité de la décision.

Mots-clefs : Complexité de calcul, logique informatique, hypergraphes, base de données. (indexation libre : complexité descriptive)

Abstract

Beyond the decision of satisfiability problems, we investigate the problem of enumerating all their solutions.

In a first part, we consider the enumeration problem in the framework of the propositional satisfiability problem. Creignou and Hébrard proved that the polynomial classes for the non-trivial sat problem are exactly those for the enumeration problem. We give optimal enumeration algorithms for each of these classes, that generalize any non-trivial decision algorithm for this class. This suggests that enumeration is *the* relevant problem in this case, rather than the decision problem.

In a second part, we simplify and complete some results of Bagan et al. that establish a strong connection between the tractability of a conjunctive query and a notion of hypergraph acyclicity. We establish similar results for the dual class of the class of conjunctive queries, thanks to a new algorithm. Finally, we generalize all these results through a single dichotomy for the enumeration problem of conjunctive signed queries, by generalizing some classical combinatorial result by Brouwer and Kolen. This dichotomy establishes a close connection between enumeration strong tractability and decision strong tractability.

Keywords: Computational complexity, computer logic, hypergraphs, database. (free index: descriptive complexity)

Discipline : Informatique et applications.

GREYC CNRS UMR 6072, Normandie Université, 14032 Caen, France.