Université de Caen

Département d'informatique
UFR de Sciences

École doctorale SIMEM

Génération à délai polynomial pour le problème SAT

THÈSE

présentée et soutenue publiquement le 19 janvier 2000

pour l'obtention du

Doctorat de l'université de Caen

(spécialité informatique)

par

Emmanuel Benoist

Composition du jury

Rapporteurs :	Hans Kleine Büning	Universität-Gesamthochschule Paderborn
	Marie-Catherine Vilarem	Université de Montpellier
Examinateurs :	Malik Ghallab	LAAS Toulouse
	Étienne Grandjean	Université de Caen (co-directeur)
	Jean-Jacques Hébrard	Université de Caen (co-directeur)
	Vangelis Paschos	Université Paris-Dauphine

Groupe de Recherches en Informatique, Image, Instrumentation de Caen - CNRS UPRESA 6072

Mis en page avec la classe thloria.

Table des matières

Table des figures

Introduction générale

1 La problématique

1.1 Intelligence artificielle et logique

La logique a été très étudiée au fil des siècles, que ce soit par les philosophes ou les mathématiciens, et maintenant par les informaticiens. Dans cette thèse, nous nous sommes intéressés plus particulièrement à l'aspect symbolique de la logique. Nous voulons utiliser la logique symbolique pour modéliser et résoudre des problèmes difficiles (et souvent même impossibles) à résoudre pour un être humain.

En informatique, la logique symbolique est particulièrement utilisée en Intelligence Artificielle [13, 25, 43], elle semble en effet pouvoir modéliser un grand nombre de phénomènes. Durant la seconde moitié des années 1960 l'Intelligence Artificielle a beaucoup profité des progrès faits dans le domaine de la démonstration automatique. L'intérêt pour la démonstration automatique était dû pour une part à la prise de conscience de ce que les déductions logiques sont une part importante de l'intelligence humaine, mais aussi aux techniques de démonstration automatique de théorèmes mises au point à la fin des années 1960. En parallèle aux progrès techniques sur la démonstration automatique de théorèmes, on trouve les progrès dans l'application de ces techniques pour résoudre divers problèmes de l'intelligence artificielle.

Durant les années 70, les systèmes experts ont popularisé les systèmes à bases de règles. De nombreux systèmes de résolution de problèmes concrets ont ainsi pu être construits. Le principe de ces systèmes est de recueillir les connaissances d'un expert d'un domaine bien précis, pour en faire des règles. Ces règles sont ensuite appliquées aux cas qui doivent être étudiés.

Depuis quelques années, une autre utilisation de la logique est faite par les chercheurs en Intelligence Artificielle [7, 17, 22]. On utilise de nouveaux résultats en logique (principalement propositionnelle) pour construire des systèmes de résolution de problèmes basés non plus sur des règles saisies par un expert, mais sur le résultat d'observations passées. On se sert ici des résultats passés pour construire des règles qui servent à prédire le futur. Par exemple Boros et al. [7] présentent le système LAD (Logical

Analysis of Data) qui permet de classifier de nouvelles observations, de façon que cette classification soit cohérente avec celle des observations passées. Les informations utilisées par ce système consistent en une archive des observations passées et de leur classification. Ce type de systèmes permet de résoudre des problèmes concrets. Les banques peuvent s'en servir pour déterminer si elles accordent ou non un crédit à quelqu'un, elles se servent dans ce cas des informations sur l'âge, la profession, les revenus des personnes. Les médecins peuvent s'en servir pour déterminer le caractère malin d'une tumeur, en utilisant des paramètres telles que la taille des cellules, l'âge du patient, la couleur des cellules ...

Le développement de l'intelligence artificielle et en particulier du Data Mining semble donc lié aux recherches qui sont effectuées sur les aspects théoriques de la logique, et en particulier sur la logique propositionnelle.

1.2 Le problème SAT

Dans cette thèse, nous nous intéressons à la logique propositionnelle, et plus précisément au problème central de cette logique qui est l'étude de la satisfaisabilité d'une formule donnée.

Nous allons maintenant considérer un exemple très simple pour montrer comment la logique propositionnelle peut être utilisée pour modéliser des problèmes. Comme nous n'avons pas encore défini les notions de bases, le lecteur devra pour le moment se référer à son intuition.

Nous allons étudier les faits suivants :

1. Je n'arrive pas en retard au bureau ;
2. Si je me lève tard et que je ne roule pas vite, alors j'arrive en retard au bureau ;
3. Si je me lève tard et que je roule vite, alors je ne suis pas en retard au bureau ;
4. Si je roule vite, et que la police fait un contrôle de vitesse, alors je n'ai plus de permis ;
5. Je dois garder mon permis.

On recherche une situation dans laquelle tous ces faits sont vérifiés. Comme ces faits sont donnés en français, nous avons à les représenter avec des symboles. On va décrire L l'action « Je me lève tard », A l'action « J'arrive en retard au bureau », R « Je roule vite », Po « La police effectue un contrôle de vitesse » et Pe « Je garde mon permis ». Toutes ces variables peuvent prendre la valeur Vrai ou la valeur Faux. De plus,

nous avons besoin de quelques symboles logiques $\wedge$ représente ET, $\vee$ représente OU, $\neg$ représente NON et $\rightarrow$ représente IMPLIQUE. Les faits énoncés ci-dessus peuvent donc se traduire de la manière suivante.

1. $\neg A$
2. $L \wedge \neg R \rightarrow A$
3. $L \wedge R \rightarrow \neg A$
4. $R \wedge Po \rightarrow \neg Pe$
5. Pe

À l'aide de règles de transcriptions, on peut transformer cette formule en une formule ne contenant qu'une conjonction de clauses, où chaque clause est elle même une disjonction de symboles, précédés ou non de la négation. Cela nous donne une formule en Forme Normale Conjonctive (CNF en anglais).

1. $\neg A$
2. $\neg L \vee R \vee A$
3. $\neg L \vee \neg R \vee \neg A$
4. $\neg R \vee \neg Po \vee \neg Pe$
5. Pe

Résoudre le problème de la satisfaisabilité d'une formule (dit problème SAT) revient à regarder s'il est possible de trouver une valeur à chacun des symboles de telle façon que toutes les clauses soient vérifiées en même temps. Ici, une des solutions possibles est $L = faux$, $A = faux$, $R = vrai$, $Po = faux$, $Pe = vrai$. Elle signifie : Je ne me lève pas en retard, je n'arrive pas en retard, je roule vite, la police n'est pas là et je garde mon permis. Cette solution vérifie les contraintes posées, mais elle n'est pas satisfaisante (il est stupide de se lever tôt et de rouler vite tout de même).

1.3 NP-complétude du problème de satisfaisabilité

La logique propositionnelle peut donc être utilisée pour modéliser de très nombreux problèmes. Malheureusement, il n'est actuellement pas possible de résoudre efficacement tous ces problèmes. Il n'existe pas d'algorithme polynômial pour résoudre le problème de la satisfaisabilité d'une formule quelconque. Cook [16] a de plus prouvé, que dans la classe de tous les problèmes dont on peut tester une solution en temps polynômial

(la classe NP), le problème de la satisfaisabilité fait partie des problèmes les plus difficiles (qui sont appelés NP-complets). Si on connaît un algorithme efficace pour résoudre le problème de satisfaisabilité, alors il est possible de construire des algorithmes efficaces pour tous les problèmes de la classe NP.

1.4 Formules de Horn et extensions

Déterminer la satisfaisabilité d'une formule est un problème difficile dans le cas général, il est utile d'étudier des classes de formules pour lesquelles le test de satisfaisabilité peut se faire en temps polynômial [36, 37]. Parmi l'ensemble des classes pour lesquelles on connaît un algorithme polynômial pour résoudre le problème SAT, la plus importante est sans doute la classe des formules de Horn. Il s'agit des formules en forme normale conjonctive dont chaque clause contient au plus un littéral positif.

Par exemple $\mathcal{F} = \{\{\neg x_1, x_2, \neg x_4\}, \{\neg x_1, \neg x_3\}, \{x_5\}\}$ est une formule de Horn.

Le test de satisfaisabilité d'une formule de Horn est basé sur la mécanique bien connue de la résolution unitaire. On remarque que si une formule $\mathcal{F}$ contient la clause unitaire $\{x\}$ (resp. $\{\neg x\}$), alors la seule valeur possible pour x est vrai (resp. faux), on peut donc simplifier $\mathcal{F}$ en affectant cette valeur à la variable x. On définit $\mathcal{F}_x$ (resp. $\mathcal{F}_{\neg x}$) comme la formule $\mathcal{F}$ dans laquelle on a effacé toutes les clauses contenant x (resp. $\neg x$) et telle que $\neg x$ (resp. x) est ôté de toutes les clauses qui le contenaient. Ces simplifications de $\mathcal{F}$ correspondent aux simplifications que l'on fait lorsque l'on affecte la valeur vrai (resp. faux) à la variable x. On peut aisément prouver que si la formule $\mathcal{F}$ contient la clause $\{x\}$ (resp. $\{\neg x\}$), $\mathcal{F}$ sera satisfaisable si et seulement si, la formule $\mathcal{F}_x$ (resp. $\mathcal{F}_{\neg x}$) est satisfaisable et ne contient pas la clause vide.

Par exemple si $\mathcal{F} = \{\{x_1, x_2\}, \{\neg x_1, x_3\}, \{x_4, x_5\}\}$, alors $\mathcal{F}_{x_1} = \{\{x_3\}, \{x_4, x_5\}\}$.

Si on répète cette simplification ($\mathcal{F}_x$ contient peut-être une clause unitaire), on obtient une formule dont toutes les clauses sont de longueur deux au moins. On va appeler cette formule $Noyau(\mathcal{F})$. $\mathcal{F}$ est satisfaisable, si et seulement si $Noyau(\mathcal{F})$ est satisfaisable et la résolution unitaire n'a pas généré de clause vide. Ce processus peut être effectué en temps linéaire pour toute formule $\mathcal{F}$.

Pour les formules appartenant à la classe Horn, la résolution unitaire est suffisante pour déterminer la satisfaisabilité de $\mathcal{F}$. En effet, $Noyau(\mathcal{F})$ (dont toutes les clauses sont de longueur supérieure ou égale à deux) est satisfaisable puisque si on donne la valeur faux à toutes les variables de cette formule, toutes les clauses sont satisfaites (chacune est moins de longueur 2 et contient au plus un littéral positif, donc chacune contient au moins un littéral négatif). Tester la satisfaisabilité d'une formule Horn,

revient donc à tester si la résolution unitaire sur cette formule a généré la clause vide.

Le renommage d'une formule $\mathcal{F}$ est obtenu en choisissant un ensemble de variables U et en remplaçant pour tout $x \in U$ les occurrences de x par $\neg x$ et les occurrences de $\neg x$ par x. L'intérêt de cette manipulation vient de ce qu'elle ne change pas la satisfaisabilité de la formule, ni même le nombre de solutions ; si la formule originale admet un modèle, alors la formule renommée admet comme modèle le modèle de la formule originale renommé.

Si on peut trouver un renommage qui transforme une formule $\mathcal{F}$ en une formule appartenant à une classe polynômiale, alors on peut tester la satisfaisabilité de $\mathcal{F}$ en temps polynômial. Pour la classe Horn, il existe des algorithmes linéaires [31, 38] permettant de tester si une formule peut se renommer en une formule de Horn.

D'autres extensions ont été proposées autour des formules de Horn. Chandru et Hooker [12] ont présenté la classe des formules Horn étendues, pour laquelle le test de satisfaisabilité est exactement le même que pour les formules de Horn, c'est à dire uniquement basé sur la résolution unitaire. Malheureusement, il n'existe actuellement aucun algorithme polynômial permettant de tester si une formule est Horn étendue. C'est pourquoi, Swaminathan et Wagner [46], ont présenté les formules Horn étendues simples qui elles aussi admettent le même algorithme pour tester si une formule est satisfaisable, mais pour laquelle ils donnent un algorithme de reconnaissance qui est quadratique. Conforti et Cornuéjols [14, 15] ont présenté les formules équilibrées qui étendent elles aussi les formules de Horn. Pour les formules équilibrées, le test de satisfaisabilité se fait aussi de la même façon que pour les formules de Horn, à l'aide de la résolution unitaire.

Les formules de Horn ont aussi été étendues dans d'autres directions. Par exemple Yamasaki et Doshita [48] ont défini une classe de formules appelées depuis Horn généralisées. Kleine Büning [35] a donné un test de satisfaisabilité pour cette classe basé sur la 2-résolution et dont la complexité est quadratique. Gallo et Scutella [27] ont étendu cette classe en une hiérarchie : les classes Γ_k ($k \geq 1$). Kleine Büning [35] a prouvé que l'on peut tester la satisfaisabilité d'une formule appartenant à la classe Γ_i en utilisant la $(i+1)$-résolution. Malheureusement, Eiter et al. [23] ont prouvé que tester si une formule est Γ_k-renommable est NP complet pour tout k supérieur au égal à 1.

D'autres classes étendent les formules de Horn. On peut citer les hiérarchies $\{\Omega\}$ et $\{\Delta\}$ présentées par Dalal et Etherington [20] ainsi que la classe Quad présentée par Dalal [19], mais les méthodes utilisées pour tester la satisfaisabilité de ces formules sont spécifiques à ces classes et sont éloignées de celles utilisées pour Horn.

1.5 Génération à délai polynômial

Générer toutes les solutions qui satisfont un ensemble de contraintes est un problème très étudié en algorithmique combinatoire. Par exemple la théorie des graphes offre de nombreux problèmes intéressants à étudier de ce point de vue. Johnson et al [33] ont étudié la génération d'ensembles maximaux indépendants. Read et Tarjan [42], Gabon et Myers [26] ou Shioura et al. [45] se sont intéressés aux arbres couvrants des graphes non orientés. Hariharan et al. [30] ont quant à eux étudié la génération des arbres couvrants des graphes orientés. Kalvin et Varol [34], Pruesse et Ruskey [41] ou Canfield et Williamson [9] ont étudié comment on pouvait générer efficacement tous les tris topologiques pour un ordre partiel donné.

Il a fallu trouver une nouvelle notion pour décrire la complexité de tels algorithmes. En effet, dans les cas les plus intéressants, le nombre de solutions à générer est potentiellement exponentiel en fonction de la taille de la donnée. C'est le cas pour les ensembles maximaux indépendants ou pour les solutions d'une formule de logique propositionnelle. La notion de performance que nous utiliserons dans cette thèse devra tenir compte de cette remarque. Johnson et al [33] ont proposé une notion de complexité, qu'ils ont appelée *génération à délai polynômial*.

Un algorithme génère les solutions d'un problème à délai polynômial, s'il génère toutes les solutions, les unes à la suite des autres, de telle façon que le délai avant la première solution, ensuite entre deux solutions consécutives, et après la dernière solution, est borné par un polynôme en fonction de la taille de la donnée.

Johnson et al [33] ont proposé un algorithme de génération à délai polynômial de tous les ensembles maximaux indépendants d'un graphe. Creignou et Hébrard [18] ont étudiés la génération à délai polynômial de toutes les solutions de certaines classes du problème SAT généralisé.

Pour le problème de la satisfaisabilité d'une formule de logique propositionnelle, trouver une solution est déjà difficile, (exponentiel dans le cas le pire). Les chercheurs se sont donc concentrés sur certaines classes de formules pour lesquelles ils ont proposé des algorithmes polynômiaux permettant de tester la satisfaisabilité et le plus souvent de retourner une solution. Nous allons dans cette thèse étudier la plupart de ces classes et voir celles pour lesquelles la génération à délai polynômial est possible. Dans la partie I, nous allons considérer les principales classes polynômiales connues et voir comment on peut adapter ces résultats pour la génération de toutes les solutions à délai polynômial. Dans les parties II et III, nous étudions plus en détail la classe des formules Horn étendues, et la nouvelle classe des formules ordonnées. Pour ces classes, on prouve que la résolution unitaire suffit à générer efficacement toutes les solutions.

Dans l'exemple que nous avons présenté à la section 1.2, la solution qui est présentée n'est pas satisfaisante, se lever tôt et tout de même rouler

vite, ne semble pas très intelligent. L'utilisateur de notre système peut donc demander à voir successivement toutes les autres solutions. Nous avons quatre solutions possibles : La première consiste à rouler vite, il n'y a pas de police et se lever tard. La seconde est la même sauf que l'on se lève tôt. La troisième consiste à ne pas rouler vite, ne pas se lever tard, et la police effectue son contrôle de vitesse. La quatrième est la même mais la police n'effectue pas de contrôle. La connaissance de toutes les solutions nous permet donc de choisir la meilleure option. Cela laisse à l'utilisateur le choix, ce qui peut être très important pour un système expert de ce type.

2 Les résultats

2.1 Génération à délai polynômial

Dans la partie I, nous exposons un algorithme très simple pour générer tous les modèles d'une formule. Ensuite, nous étudions la plupart des classes polynômiales connues et regardons si on peut utiliser notre algorithme pour générer toutes les solutions des formules appartenant à ces classes. En étudiant les propriétés de la résolution unitaire, on peut prouver que notre algorithme génère toutes les solutions avec un délai $O(nN)$ (où n est le nombre de variables dans la formule et N la longueur totale de la formule, c'est à dire la somme des longueurs des clauses) pour les formules Horn, Horn renommables et binaires (dont toutes les clauses sont de longueur deux) ainsi que pour les formules équilibrées introduites par Conforti et Conuéjols [14, 15].

Ensuite, nous étudions la classe des formules presque Horn qui est issue des travaux de Hébrard et Luquet sur la base de Horn [32]. Nous prouvons que si on prend la peine de pré-calculer un ordre sur les variables d'une telle formule (ce qui peut être fait en temps $O(nN)$), alors on peut générer tous ses modèles avec un délai $O(nN)$ avec pour seul outil la résolution unitaire. On peut ensuite utiliser ce résultat pour démontrer la possibilité de générer à délai $O(nN)$ les solutions des formules de la classe q-Horn.

Nous étudions ensuite les classes de formules pour lesquelles la résolution unitaire ne suffit pas pour générer toutes les solutions. On utilise dans ce cas les résultats établis pour résoudre le problème de la satisfaisabilité des formules de ces classes. Nous présentons un algorithme pour générer toutes les solutions d'une formule appartenant à l'une des classes Γ_k ($k \geq 0$) (présentées par Gallo et Scutella [27] à partir d'une classe décrite par Yamasaki et Doshita [48]) avec un délai $O(n^k N)$. Le même type de méthode peut être appliqué pour générer à délai polynômial les solutions des formules appartenant aux classes faisant partie des hiérarchies présentées par Pretolani [40].

Nous prouvons par contre que pour la classe Quad présentée par Dalal [19], ainsi que pour les classes appartenant aux hiérarchies $\{\Omega\}$ et $\{\Delta\}$, présentées par Dalal et Etherington [20], si P$\neq$NP, il est impossible de trouver un algorithme à délai polynômial pour générer toutes les solutions de ces formules.

2.2 Formules Horn étendues (simples)

Les parties II et III sont consacrées à l'étude de classes pour lesquelles la résolution unitaire est le seul moteur utilisé pour la génération de toutes les solutions.

Dans la partie II, nous étudions les propriétés des formules Horn étendues présentées par Chandru et Hooker [12] et ensuite étudiées par Swaminathan et Wagner [46] ainsi que Schlipf et al. [44]. Nous présentons tout d'abord les formules Horn étendues. Une formule $\mathcal{F}$ est dite *Horn étendue*, s'il existe une arborescence dont les arcs sont étiquetés de manière unique par les variables de $\mathcal{F}$ et telle que : pour chaque clause de la formule, les variables apparaissant positivement dans la clause, forment un chemin dans l'arborescence, et les variables apparaissant négativement dans cette même clause forment des chemins disjoints commençant tous à la racine de l'arborescence plus un chemin commençant au même sommet que le chemin positif. Nous étudions dans cette thèse l'origine de cette définition et aussi comment il est possible de générer à délai $O(nN)$ tous les modèles d'une telle formule. Malheureusement, il n'existe pas actuellement d'algorithme polynômial permettant de tester si une formule donnée appartient ou non à la classe Horn étendue.

C'est la raison de la création des formules *Horn étendues simples* par Swaminathan et Wagner [46]. Ils ont remarqué que si on simplifie un peu la définition donnée par Chandru et Hooker, on peut obtenir une classe de formules gardant les mêmes propriétés : on peut tester la satisfaisabilité en temps linéaire. Nous montrons que l'on peut générer tous les modèles avec un délai $O(nN)$. Ils proposent un algorithme *quadratique* pour tester si une formule appartient ou non à leur classe. Nous étudions cette classe, et mettons en avant une structure intrinsèque à toute formule. Il s'agit d'une partition de l'ensemble des variables en classes d'équivalences appelées agrégats. Ensuite, nous étudions les agrégats des formules Horn étendues simples et découvrons des relations à l'intérieur de ces agrégats et entre ces agrégats. Cela nous permet de construire un algorithme *linéaire* pour la reconnaissance des formules Horn étendues simples. Cela nous permet aussi de proposer un algorithme *linéaire* pour la reconnaissance des formules de la classe Horn élargie simple que nous avons créée à la suite d'une remarque de Schlipf et al. [44].

2.3 Formules ordonnées

Dans la partie III, nous présentons une nouvelle classe de formules, les *formules ordonnées*. La classe des formules ordonnées étend, d'une façon très naturelle, la classe des formules de Horn, et tout comme celles de Horn, elles possèdent de très nombreuses propriétés intéressantes.

On peut tester si une formule est ordonnée en temps $O(nN)$. Soit $\mathcal{F}$ une formule ordonnée, il est possible de déterminer si $\mathcal{F}$ est satisfaisable avec un algorithme linéaire basé uniquement sur la résolution unitaire. On peut générer tous les modèles de $\mathcal{F}$ avec un délai $O(nN)$.

Mais là où cette classe est particulièrement intéressante, c'est que l'on peut tester en temps $O(nN)$ si une formule $\mathcal{F}$ est ordonnée-renommable, c'est à dire, s'il existe un ensemble X de variables tel que si on remplace les occurrences de x par $\neg x$ et les occurrences de $\neg x$ par x pout tout $x \in X$, on transforme $\mathcal{F}$ en une formule ordonnée. Cette propriété n'est pas triviale, puisque par exemple tester si une formule peut être renommée en une formule de la classe Γ_k (pour tout $k \geq 1$) est au contraire NP-complet [23].

Comme le test de renommage utilise les mêmes techniques que pour les formules de Horn, on peut étendre les résultats obtenus par Hébrard et Luquet [32] sur la base de Horn et développés dans cette thèse avec la définition et l'étude des formules presque Horn (Partie I). On peut ainsi définir la classe des formules presque ordonnées. Nous présentons en outre une méthode pour générer tous les modèles d'une telle formule qui est très similaire à la méthode proposée pour générer les modèles d'une formule presque Horn présentée dans la Partie I. Sous réserve de disposer d'un ordre sur les variables (qui peut être calculé en temps $O(nN)$), notre algorithme peut générer tous les modèles d'une formule presque ordonnée en n'utilisant que la résolution unitaire.

Première partie

Le problème SAT et la génération à délai polynomial

Chapitre 1

Introduction

Sommaire

1.1 Préliminaires

Le problème de satisfaisabilité d'une formule de logique propositionnelle est un des problèmes centraux de l'informatique théorique. Il s'agit de trouver pour une formule donnée si oui ou non, il existe une solution pouvant rendre cette formule vraie.

Cook [16] a prouvé que tout problème de la classe NP (i.e. dont on peut tester si une donnée est solution en temps polynômial) pouvait se réduire en une instance du problème de satisfaisabilité. Depuis, de très nombreux problèmes ont été prouvés être de complexité équivalente au problème SAT et ils forment la classe des problèmes NP-complets.

L'utilisation de formules de logique propositionnelle est très utile, par exemple en intelligence artificielle. Ces formules peuvent être utilisées pour faire de la démonstration automatique. On peut aussi voir les systèmes de déduction à base de règles comme une application du problème SAT. La montée en puissance du Data Mining, a bien montré l'intérêt pratique de l'étude de la complexité des algorithmes et la maîtrise de cette complexité.

Nous allons étudier dans cette partie et dans cette thèse une extension du problème de satisfaisabilité d'une formule de logique propositionnelle : « la génération de toutes les solutions qui satisfont une formule ceci à délai raisonnable ».

En effet les utilisateurs de systèmes en Intelligence Artificielle, ne se contentent pas de savoir qu'une réponse existe, ils veulent la connaître. Si celle-ci ne leur convient pas, ils veulent en connaître une autre, puis peut-être une troisième. De plus entre deux réponses consécutives, le temps doit être borné par un délai polynômial.

Johnson et al. [33] ont présenté une notion de complexité qui s'adapte bien à ce type de problèmes. Comme le nombre des solutions peut être exponentiel, il est impossible de définir une complexité dépendant uniquement du temps d'exécution total de l'algorithme. C'est pourquoi ils ont défini la notion de délai polynômial. Le temps avant la première solution, entre deux solutions et pour dire qu'il n'existe pas d'autre solution doit être borné par un polynôme en fonction de la taille de la formule.

Nous présentons ici un algorithme qui permet de générer à délai polynômial toutes les solutions de certaines formules. On prouve ensuite que cet algorithme fonctionne avec les formules appartenant à presque toutes les classes pour lesquelles on peut tester la satisfaisabilité en temps polynômial.

1.2 Définitions

On va définir ici les principaux concepts et notations qui seront utilisés tout au long de cette thèse.

Rappelons ici qu'un *littéral* est soit une variable propositionnelle x (littéral *positif*), soit sa négation $\neg x$ (littéral *négatif*). Une *clause* est un ensemble fini de littéraux. Si une clause C contient au plus un littéral positif, C est appelée *clause de Horn*, et si $card(C) = 1$, on dit que C est une *clause unitaire*. Une formule est un ensemble fini de clauses. Tout au long de cette thèse, $\mathcal{F}$ désignera une formule, $V = \{x_1, \ldots, x_n\}$ son ensemble de variables et $N = \Sigma_{C \in \mathcal{F}} card(C)$. Dans la Partie II, nous utiliserons la notation F pour représenter une formule. $\mathcal{F}$ est une *formule de Horn* si toutes ses clauses sont des clauses de Horn.
Exemple : Soit $\mathcal{F}_1 = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$, avec $C_1 = \{\neg x_1, \neg x_2, x_4, x_5, \neg x_7\}$, $C_2 = \{\neg x_1, \neg x_2, \neg x_3\}$, $C_3 = \{\neg x_1, \neg x_4, \neg x_5\}$, $C_4 = \{x_1, x_2, x_3, \neg x_5, \neg x_6\}$, $C_5 = \{\neg x_6, x_7\}$, $C_6 = \{x_6\}$, $C_7 = \{x_2, x_3\}$. $\mathcal{F}_1$ est une formule et C_3 en est la troisième clause.

Soit x une variable et l un littéral. Si on a $l = x$ ou $l = \neg x$, on note $var(l) = x$. Si $l = x$ (resp. $l = \neg x$), on écrit $\overline{l} = \neg x$ (resp. $\overline{l} = x$). Soit L un ensemble de littéraux. On écrit $\overline{L}$ l'ensemble $\{\overline{l} \mid l \in L\}$. L est *cohérent* s'il ne contient pas à la fois l et $\overline{l}$ quelque soit le littéral l, sinon il est *incohérent*.

Soit $X \subseteq V$. $Lit(X)$ représente l'ensemble des littéraux définis sur X, i.e. $Lit(X) = X \cup \{\neg x \mid x \in X\}$. Si $C \subseteq Lit(X)$, on dit que C est une clause sur X. Un ensemble de littéraux L est *complet pour* X si

$L \cup \overline{L} = Lit(X)$. L est *complet* pour une formule s'il est complet pour V si V désigne l'ensemble des variables de la formule.

On dit qu'on *renomme* x dans $\mathcal{F}$ si on remplace toute occurrence de x par $\neg x$ et toute occurrence de $\neg x$ par x. Un *renommage* est un ensemble de littéraux qui est complet et cohérent. Soit R un renommage et $x \in V$ une variable, on a $R(x) = x$ et $R(\neg x) = \neg x$ si $x \in R$ (x n'est pas renommé), on a aussi $R(x) = \neg x$ et $R(\neg x) = x$ si $\neg x \in R$ (x est renommé). On remarque que pour tout renommage R, et tout littéral l, $R(l)$ est un littéral positif, si et seulement si $l \in R$. Pour toute clause C, $R(C)$ décrit l'ensemble $\{R(l) \mid l \in C\}$. De même pour toute formule $\mathcal{F}$, on aura $R(\mathcal{F}) = \{R(C) \mid C \in \mathcal{F}\}$.
Exemple : Soit $R = \{\neg x_1, \neg x_2, \neg x_3, x_4, x_5, x_6, x_7\}$ un renommage. $R(\mathcal{F}_1) = \{\{x_1, x_2, x_4, x_5, \neg x_7\}, \{x_1, x_2, x_3\}, \{x_1, \neg x_4, \neg x_5\}, \{\neg x_1, \neg x_2, \neg x_3, \neg x_5, \neg x_6\}, \{\neg x_6, x_7\}, \{x_6\}, \{\neg x_2, \neg x_3\}\}$.

$\mathcal{F}$ est dite *satisfaisable* s'il existe un ensemble de littéraux cohérent L tel que pour toute clause $C \in \mathcal{F}$, $C \cap L \neq \emptyset$. Soit M un ensemble cohérent et complet. M est un *modèle* de $\mathcal{F}$ si pour toute clause $C \in \mathcal{F}$, $C \cap M \neq \emptyset$.
Exemple : L'ensemble de littéraux $M = \{\neg x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$ est un modèle de la formule $\mathcal{F}_1$.

Soit C une clause, on notera $pos(C)$ l'ensemble $\{x \in V \mid x \in C\}$, et $neg(C)$ l'ensemble $\{x \in V \mid \neg x \in C\}$. Soit x une variable, on note $CNeg(x)$ l'ensemble $\{C \in \mathcal{F} \mid \neg x \in C\}$ (clauses contenant $\neg x$), et $CPos(x)$ l'ensemble $\{C \in \mathcal{F} \mid x \in C\}$ (clauses contenant x). Soit l un littéral, on écrit $Occ_{\mathcal{F}}(l)$ l'ensemble $\{C \in \mathcal{F} \mid l \in C\}$ des clauses de $\mathcal{F}$ contenant l ; lorsqu'aucune confusion ne peut être faite, on note cet ensemble $Occ(x)$. On peut remarquer que $CPos(x) = Occ(x)$ et $CNeg(x) = Occ(\neg x)$. Si C est un ensemble de littéraux, on note $var(C)$ l'ensemble $\{x \in V \mid x \in C \, ou \, \neg x \in C\}$, pour toute formule $\mathcal{G} = \{C_1, \ldots, C_k\}$ on note $var(\mathcal{G}) = var(C_1) \cup \ldots \cup var(C_k)$.
Exemple : Pour la formule $\mathcal{F}_1$ définie ci-dessus, $pos(C_1) = \{x_4, x_5\}$, $pos(C_2) = \emptyset$, $neg(C_1) = \{x_1, x_2, x_7\}$, $CNeg(x_1) = \{C_1, C_2, C_3\}$, $CPos(x_2) = \{C_4, C_7\}$, $Occ(\neg x_3) = \{C_2\}$.

Soit x une variable et l un littéral, on note $\mathcal{F} \backslash x = \{C \backslash x, \neg x \mid C \in \mathcal{F}\}$ et $\mathcal{F}_l = \{C \setminus \{\overline{l}\} \mid C \in \mathcal{F}, l \notin C\}$.
Exemple : On a $\mathcal{F}_1 \setminus x_1 = \{ \{\neg x_2, x_4, x_5, \neg x_7\}, \{\neg x_2, \neg x_3\}, \{\neg x_4, \neg x_5\}, \{x_2, x_3, \neg x_5, \neg x_6\}, \{\neg x_6, x_7\}, \{x_6\}, \{x_2, x_3\}\}$, et $(\mathcal{F}_1)_{\neg x_2} = \{ \{\neg x_1, \neg x_4, \neg x_5\}, \{x_1, x_3, \neg x_5, \neg x_6\}, \{\neg x_6, x_7\}, \{x_6\}, \{x_3\}\}$.

On dit qu'une clause C est *dérivable* de $\mathcal{F}$ par *résolution unitaire* s'il existe une suite de clauses $C_1, \ldots, C_p$ avec $C_p = C$ telle que pour tout i $(1 \leq i \leq p)$; ou bien $C_i \in \mathcal{F}$ ou bien il existe $j, k < i$ et un littéral

l vérifiant $C_j = C_i \cup \{\bar{l}\}$ et $C_k = \{l\}$. Soit $\sigma = (C_1, \ldots, C_p)$, σ est une *dérivation unitaire*.

Soit $Unit(\mathcal{F})$ l'ensemble des clauses unitaires dérivables de $\mathcal{F}$ par résolution unitaire.

On sait que si $Unit(\mathcal{F})$ n'est pas cohérent alors $\mathcal{F}$ est non satisfaisable. La réciproque est en général fausse.

Soit $Noyau(\mathcal{F})$ l'ensemble des clauses obtenues à partir de $\mathcal{F}$ en ôtant les clauses qui contiennent un élément de $Unit(\mathcal{F})$, et en supprimant dans les clauses restantes tous les complémentaires des littéraux de $Unit(\mathcal{F})$.
Exemple : En reprenant la formule $\mathcal{F}_1$ de l'exemple ci-dessus, on obtient : $Unit(\mathcal{F}_1) = \{x_6, x_7\}$ et $Noyau(\mathcal{F}_1) = \{\{\neg x_1, \neg x_2, x_4, x_5\}, \{\neg x_1, \neg x_2, \neg x_3\}, \{\neg x_1, \neg x_4, \neg x_5\}, \{x_1, x_2, x_3, \neg x_5\}, \{x_2, x_3\}\}$.

La proposition suivante est le rappel d'un résultat de base [39].

Proposition 1 *$\mathcal{F}$ est satisfaisable si et seulement si $Unit(\mathcal{F})$ est cohérent et $Noyau(\mathcal{F})$ est satisfaisable.*

Preuve : Si $\mathcal{F}$ est satisfaisable, alors évidemment $Unit(\mathcal{F})$ est cohérent et $Noyau(\mathcal{F})$ est satisfaisable. Maintenant supposons que $Unit(\mathcal{F})$ est cohérent et $Noyau(\mathcal{F})$ est satisfaisable. Soit M_N un modèle de $Noyau(\mathcal{F})$, l'ensemble $M_N \cup Unit(\mathcal{F})$ est un modèle de $\mathcal{F}$. □

Il est bien connu que l'ensemble $Unit(\mathcal{F})$ et la formule $Noyau(\mathcal{F})$ peuvent être calculés en temps $O(N)$ (voir [20] par exemple).

Par abus de notation, on utilisera parfois la notation $\mathcal{U}$ dans laquelle $\mathcal{U}$ est un ensemble de littéraux pour représenter l'ensemble de clauses unitaires $\mathcal{U}' = \{\{l\} \mid l \in \mathcal{U}\}$.

1.3 Algorithme générique

Nous présentons ici un algorithme pour la génération à délai polynômial des modèles pour les formules de la plupart des classes polynômiales connues.

Proposition 2 *Si pour tout ensemble de littéraux $\mathcal{U}$, on peut tester en temps $O(f(N))$ si $\mathcal{F} \cup \mathcal{U}$ est satisfaisable, alors on peut générer les modèles de $\mathcal{F}$ à délai $O(nf(N))$.*

Preuve : Nous allons utiliser l'algorithme Génération (Fig.1.1) pour générer tous les modèles de $\mathcal{F}$. Pour tout couple $(\mathcal{U}, i)$ empilé dans P, il existe au moins un modèle satisfaisant $\mathcal{F} \cup \mathcal{U}$ qui contient $\mathcal{U}$, sinon $\mathcal{F} \cup \mathcal{U}$ ne serait pas satisfaisable, et donc ne serait pas empilé. Donc, si cet algorithme retourne un ensemble de littéraux, on est sûr que cet ensemble est un modèle de $\mathcal{F}$. Comme cet algorithme examine implicitement tous les

Algorithme Génération
Entrée : Une formule $\mathcal{F}$, satisfaisant le conditions de la proposition 2 ;
et une permutation $(x_1, \ldots, x_n)$ des variables de $\mathcal{F}$;
Sortie : Les modèles de $\mathcal{F}$;
début
Pile $P \leftarrow \emptyset$
si $\mathcal{F}$ est satisfaisable **alors** P.empiler $(\emptyset, 1)$
tant que $P \neq \emptyset$ **faire**
$(\mathcal{U}, i) \leftarrow P$.dépiler() ;
si $i = n + 1$ **alors sortir**$(\mathcal{U})$
sinon
si $\mathcal{F} \cup \mathcal{U} \cup \{\{x_i\}\}$ est satisfaisable **alors**
P.empiler$(\mathcal{U} \cup \{\{x_i\}\}, i + 1)$;
fin si ;
si $\mathcal{F} \cup \mathcal{U} \cup \{\{\neg x_i\}\}$ est satisfaisable **alors**
P.empiler$(\mathcal{U} \cup \{\{\neg x_i\}\}, i + 1)$;
fin si ;
fin sinon ;
fin tant que ;
fin.

FIG. 1.1 – *Algorithme génération*

ensembles de littéraux cohérents et complets pour V, on peut conclure que cet algorithme génère tous les modèles de $\mathcal{F}$.

Le délai entre deux modèles consécutifs est polynômial :

- lors de chaque exécution de la boucle « tant que », soit un modèle est généré, soit la valeur de i au sommet de la pile est augmentée de 1, le nombre de boucles entre deux générations est donc borné par n.
- Le coût du test de satisfaisabilité est $O(f(N))$.

La complexité totale de cet algorithme est donc : $O(nf(N))$. □

1.4 Sur quelles formules appliquer cet algorithme

L'algorithme que nous avons présenté ici fonctionne avec presque toutes les classes de formules connues. Nous allons maintenant voir quels sont les mécanismes qui font que si on connaît un algorithme polynômial

pour tester la satisfaisabilité d'une formule, alors, la plupart du temps, on peut générer toutes les solutions de cette formule avec un délai polynômial.

Au Chapitre 2 nous allons étudier un ensemble de classes, telles que les formules de Horn, Horn renommables, binaires ou les formules équilibrées introduites par Conforti et Conuéjols [14, 15]. Le seul outil utilisé pour la génération à délai polynômial avec ces formules est la résolution unitaire.

Au Chapitre 3, nous montrons que moyennant le pré-calcul d'un ordre particulier sur les variables, il est possible d'utiliser notre algorithme, en utilisant uniquement la résolution unitaire, pour générer les solutions de deux classes supplémentaires. La classe des formules q-Horn présentée et étudiée par Boros et al. [6, 8] et une nouvelle classe, les formules presque Horn, qui est issue des travaux de Hébrard et Luquet [32] sur la notion de base de Horn.

Au Chapitre 4 nous présentons un ensemble de classes de formules pour lesquelles la résolution unitaire n'est pas suffisante, mais pour lesquelles on sait utiliser les résultats obtenus sur le test de satisfaisabilité pour générer toutes les solutions à délai polynômial. C'est le cas des formules Horn généralisées introduites par Yamasaki et Doshita [48], de la hiérarchie Γ présentée par Gallo et Scutella [27], ainsi que des hiérarchies présentées par Pretolani [40].

Au Chapitre 5, nous prouvons que pour certaines classes de formules, par exemples les formules Quad présentées par Dalal [19] ou les hiérarchies Ω et Δ présentées par Dalal et Etherington [20], si P$\neq$NP, il n'existe pas d'algorithme de génération des solutions à délai polynômial.

Chapitre 2

Utilisation de la résolution unitaire seule

Sommaire

2.1 Introduction

La résolution unitaire joue un rôle essentiel dans l'étude du problème SAT. Elle est efficace et permet de simplifier une formule très rapidement.

Nous présentons ici un ensemble de classes de formules pour lesquelles la résolution unitaire suffit à déterminer si $(\mathcal{F} \cup \mathcal{U})$ est satisfaisable (où $\mathcal{F}$ est une formule de la classe concernée et $\mathcal{U}$ un ensemble de littéraux vu comme un ensemble de clauses unitaires).

La première de ces classes est la classe des formules de Horn. On sait que tester la satisfaisabilité d'une telle formule $\mathcal{F}$ revient à étudier si l'ensemble $Unit(\mathcal{F})$ est cohérent. Cela nous permet de proposer un algorithme à délai $O(nN)$ pour générer tous les modèles d'une formule de Horn. Nous obtenons les mêmes résultats pour les formules Horn renommables puisque les modèles d'une formule Horn renommables sont juste les modèles de la formule de Horn correspondante que l'on a renommés.

On remarque en outre que si une formule binaire est satisfaisable alors elle est Horn renommable, donc générer tous les modèles d'une formule binaire revient à tester sa satisfaisabilité, et dans le cas positif à appliquer

l'algorithme trouvé pour les formules Horn renommables. Cela peut être fait là aussi avec un algorithme à délai $O(nN)$.

Une autre classe pour laquelle la résolution unitaire suffit à tester la satisfaisabilité est la classe des formules équilibrées. On utilise cette propriété, ainsi que la stabilité de la classe par la résolution unitaire, pour prouver que l'on peut générer à délai $O(nN)$ tous les modèles d'une formule équilibrée.

Soit $\mathcal{C}$ une classe de formules.

P 1 *$\mathcal{C}$ vérifie P1, si pour toute formule $\mathcal{F} \in \mathcal{C}$, et pour tout ensemble de littéraux $\mathcal{U}$, on a $Noyau(\mathcal{F} \cup \mathcal{U}) \in \mathcal{C}$.*

On remarque que $\mathcal{C}$ vérifie P1 implique que la classe $\mathcal{C}$ est stable par la résolution unitaire.

P 2 *$\mathcal{C}$ vérifie P2, lorsque pour toute formule $\mathcal{F} \in \mathcal{C}$, le fait que toute clause de $\mathcal{F}$ soit de longueur supérieure ou égale à deux, implique que $\mathcal{F}$ soit satisfaisable.*

Proposition 3 *Si $\mathcal{C}$ vérifie les propriétés P1 et P2, alors on peut tester si $\mathcal{F} \cup \mathcal{U}$ est satisfaisable en temps $O(N)$ (où N est la longueur de $\mathcal{F} \cup \mathcal{U}$).*

Preuve : On peut calculer $Noyau(\mathcal{F} \cup \mathcal{U})$ et $Unit(\mathcal{F} \cup \mathcal{U})$ en temps linéaire ($O(N)$). On sait que $\mathcal{F} \cup \mathcal{U}$ est satisfaisable si et seulement si $Unit(\mathcal{F} \cup \mathcal{U})$ est cohérent et $Noyau(\mathcal{F} \cup \mathcal{U})$ est satisfaisable (Prop. 1). Le test de la cohérence de $Unit(\mathcal{F} \cup \mathcal{U})$ est immédiat, comme en plus on sait que $Noyau(\mathcal{F} \cup \mathcal{U})$ est élément de $\mathcal{C}$ (P1), et qu'elle ne contient que des clauses dont la longueur est supérieure ou égale à deux, on sait (P2) que $Noyau(\mathcal{F} \cup \mathcal{U})$ est satisfaisable. □

Des Propositions 2 et 3 ont déduit immédiatement :

Corollaire 4 *Si $\mathcal{C}$ vérifie les propriétés P1 et P2, alors pour toute formule $\mathcal{F} \in \mathcal{C}$, on peut générer toutes les solutions de $\mathcal{F}$ avec un délai $O(nN)$.*

Nous allons maintenant étudier plusieurs classes qui vérifient les deux propriétés P1 et P2. Parmi celles-ci, on va trouver les classes polynômiales les plus connues comme la classe des formules de Horn, les formules Horn renommables, les formules binaires, mais aussi la classe des formules équilibrées. On verra dans la Partie II que les classes des formules Horn étendues et Horn étendues simples vérifient cette propriété, on découvrira aussi dans la Partie III de cette thèse la classe des formules ordonnées qui vérifie elle aussi ces deux propriétés.

2.2 Formules de Horn

La classe des formules de Horn est la plus connue des classes de formules de logique propositionnelle. C'est la classe qui sert de base au langage Prolog ainsi qu'aux systèmes d'apprentissage à base de règles.

Définition 1 (formule de Horn) *Une clause C est dite* clause de Horn *si elle contient au plus un littéral positif. Une formule est dite de* Horn *si toutes ses clauses sont des clauses de Horn.*

Exemple : La formule $\mathcal{F}_1 = \{\{x_1, \neg x_2, \neg x_3\}, \{\neg x_4, \neg x_5\}, \{x_3, \neg x_5\}\}$ est une formule de Horn, mais la clause $\{x_4, x_5\}$ n'est pas une clause de Horn.

On peut tester en temps linéaire si une formule est une formule de Horn, il suffit de compter pour chaque clause le nombre de littéraux positifs.

Proposition 5 *Si $\mathcal{F}$ est une formule de Horn, alors pour tout ensemble de clauses unitaires $\mathcal{U}$, $Noyau(\mathcal{F} \cup \mathcal{U})$ est une formule de Horn.*

Preuve : Soit C' une clause de $\mathcal{F}' = Noyau(\mathcal{F} \cup \mathcal{U})$, il existe une clause $C \in \mathcal{F} \cup \mathcal{U}$ telle que $C' \subseteq C$ (par définition de $Noyau$). L'ensemble $\mathcal{U}$ ne contient que des clauses unitaires, donc $C \in \mathcal{F}$ et C est une clause de Horn. La clause C ne contient donc pas plus d'un littéral positif, il en est alors de même pour C'. □

Proposition 6 *Si $\mathcal{F}$ est une formule de Horn, dont toutes les clauses sont de longueur supérieure ou égale à deux, alors $\mathcal{F}$ est satisfaisable.*

Preuve : Soit $M = \{\neg x \mid x \in V\}$. L'ensemble M est un modèle pour $\mathcal{F}$, car toute clause de $\mathcal{F}$ contient au moins un littéral négatif et est donc satisfaite par M. □

Exemple : La formule $\mathcal{F}_1$ présentée ci-dessus admet l'ensemble de littéraux $\{\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5\}$ comme modèle.

La classe des formules de Horn vérifie donc les deux propriétés P1 et P2, on peut donc générer tous les modèles de toute formule de Horn à délai $O(nN)$ (Corollaire 4).

2.3 Formules Horn-renommables

La satisfaisabilité d'une formule n'est pas altérée par un renommage de ses variables. Dans les formules de Horn, littéraux positifs et négatifs

jouent des rôles dissymétriques, il est donc naturel d'étudier la classe des formules qu'un renommage de certaines de ses variables transforme en une formule de Horn.

Définition 2 (formule Horn-renommable) *Une formule est dite* Horn-renommable *si on peut la transformer en une formule de Horn en renommant certaines de ses variables.*

Exemple : La formule $\mathcal{F}_2 = \{\{x_1, \neg x_2, x_3\}, \{\neg x_4, x_5\}, \{\neg x_3, x_5\}\}$ est Horn-renommable. Si on renomme les variables x_3 et x_5, on obtient la formule de Horn $\mathcal{F}_1$ présentée dans l'exemple précédent.

Plusieurs algorithmes permettant de tester si une formule est Horn-renommable ont été proposés [1, 11, 31]. La complexité des meilleurs algorithmes est linéaire en la longueur de la formule ($O(N)$).

Proposition 7 *Si $\mathcal{F}$ est Horn-renommable, alors pour tout ensemble de littéraux $\mathcal{U}$, la formule $Noyau(\mathcal{F} \cup \mathcal{U})$ est Horn-renommable.*

Preuve : On rappelle qu'un renommage est un ensemble de littéraux qui est cohérent et complet (cf. définitions Sec. 1.2).

Comme $\mathcal{F}$ est Horn-renommable, il existe un renommage R tel que $R(\mathcal{F})$ est Horn. Soit $\mathcal{U}' = R(\mathcal{U})$, on a donc $Noyau(R(\mathcal{F}) \cup \mathcal{U}') = Noyau(R(\mathcal{F} \cup \mathcal{U}))$ est Horn (Prop. 5). D'où $Noyau(\mathcal{F} \cup \mathcal{U})$ est Horn-renommable. □

Proposition 8 *Si $\mathcal{F}$ est Horn-renommable et que toutes ses clauses sont de longueur supérieure ou égale à deux, alors $\mathcal{F}$ est satisfaisable.*

Preuve : Évident car le renommage des variables ne change pas la satisfaisabilité d'une formule. □

Exemple : La formule $\mathcal{F}_2$ définie ci-dessus est satisfaite par le modèle de $\mathcal{F}_1$ dans lequel on a renommé les variables x_3 et x_5, ce qui donne le modèle $\{\neg x_1, \neg x_2, x_3, \neg x_4, x_5\}$.

La classe des formules Horn-renommables vérifie donc les deux propriétés P1 et P2, on peut donc comme pour les formules de Horn générer tous les modèles à délai $O(nN)$ (Corollaire 4).

2.4 Formules binaires

Nous étudions ici l'autre grande classe de formules de logique propositionnelle. Les formules binaires sont intéressantes, car on peut tester leur

satisfaisabilité en temps linéaire, et il est trivial de tester si une formule appartient à la classe.

Définition 3 (formule binaire) *Une formule est dite* binaire *si toutes ses clauses sont de longueur inférieure ou égale à deux.*

Exemple : Soit $\mathcal{F}_3 = \{\{x_1, x_2\}, \{x_1, \neg x_3\}, \{\neg x_2, x_3\}, \{\neg x_2, \neg x_3\}\}$. La formule $\mathcal{F}_3$ est binaire, car toutes ses clauses ont une longueur ≤ 2.

La reconnaissance des formules binaires est trivialement linéaire, il suffit de compter pour chaque clause de $\mathcal{F}$ le nombre de littéraux.

De plus, Even et al. [24] ont donné un algorithme efficace pour tester la satisfaisabilité d'une formule binaire.

Proposition 9 *On peut tester en temps linéaire si une formule $\mathcal{F}$ dont toutes les clauses sont de longueur deux est satisfaisable.*

Preuve : On peut voir toute clause de deux littéraux $\{l_1, l_2\}$ comme une double implication, $\overline{l_1} \rightarrow l_2$ et $\overline{l_2} \rightarrow l_1$. On peut construire le graphe G dont les sommets sont les littéraux de la formule et dont les arcs correspondent aux implications données ci-dessus.

Nous allons maintenant prouver que $\mathcal{F}$ est satisfaisable si et seulement si il n'existe pas de couple de littéraux complémentaires appartenant à la même composante fortement connexe. Pour tout modèle, si un littéral appartient à M, alors il en sera de même pour tous les littéraux de la même composante fortement connexe. Donc, si deux littéraux complémentaires apparaissent dans la même composante fortement connexe, la formule $\mathcal{F}$ est forcément insatisfaisable.

A l'inverse, supposons qu'il n'existe pas de couple de littéraux complémentaires apparaissant dans une même composante fortement connexe de G. Considérons le graphe quotient G' obtenu en rassemblant en un noeud chaque composante fortement connexe de G. Ce graphe est forcément acyclique (par définition des composantes fortement connexes), il définit donc un ordre partiel sur ses éléments. On peut étendre cet ordre en un ordre total. Pour chaque variable x, si la composante fortement connexe de x apparaît avant celle de $\neg x$, on ajoute $\neg x$ dans M, sinon c'est x qui est ajouté à M. On peut prouver que ce modèle satisfait la formule.

Comme il est possible de calculer en temps linéaire les composantes fortement connexes de G en utilisant l'algorithme de Tarjan [47], comme en plus l'algorithme de Tarjan calcule les composantes fortement connexes dans l'ordre topologique inverse, nous obtenons donc un algorithme linéaire pour calculer la satisfaisabilité des formules binaires. □

Exemple : Le graphe G (Fig. 2.1) est le graphe d'implication associé à la formule $\mathcal{F}_3$. Dans ce graphe, la composante connexe de chaque

noeud est ce noeud lui même. On peut donc obtenir l'ordre total suivant : $(\neg x_1, x_2, \neg x_3, x_3, \neg x_2, x_1)$. On obtient pour cet ordre le modèle $\{x_1, \neg x_2, x_3\}$. On peut vérifier que ce modèle satisfait $\mathcal{F}_3$.

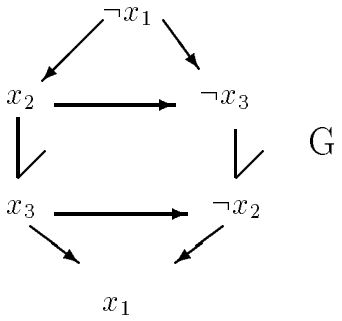

FIG. 2.1 – *Graphe d'implication de $\mathcal{F}_3$*

Proposition 10 *Une formule binaire dont toutes les clauses sont de longueur deux est satisfaisable si et seulement si elle est Horn-renommable.*

Preuve : ($\Rightarrow$) La formule $\mathcal{F}$ est satisfaisable et binaire. Soit M le modèle de $\mathcal{F}$. M est cohérent et complet, donc $\overline{M} = \{l \mid \overline{l} \in M\}$ est un renommage. $\overline{M}(l)$ est positif si et seulement si $l \in \overline{M}$, donc ssi $l \notin M$. Comme pour chaque clause $C \in \mathcal{F}$, $C \cap M \neq \emptyset$, c'est à dire $\mid C \cap M \mid \geq 1$, d'où $\mid C \cap \overline{M} \mid \leq 1$, ce qui signifie que $M(C)$ contient au plus un littéral positif. $\mathcal{F}$ est donc Horn-renommable.

($\Leftarrow$) Supposons $\mathcal{F}$ Horn-renommable. Il existe donc un renommage R tel que $R(\mathcal{F})$ est une formule de Horn. Donc pour toute clause $C \in \mathcal{F}$, on a $R(C)$ contient au plus un littéral positif donc $\mid R \cap C \mid \leq 1$. Soit $M = \{l \mid \overline{l} \in R\}$. On a que $\mid M \cap C \mid \geq 1$ donc M est un modèle pour $\mathcal{F}$. □

Exemple : La formule $\mathcal{F}_3$ est Horn-renommable. Si on renomme x_1 et x_3, on obtient la formule $\mathcal{F}_4 = \{ \{\neg x_1, x_2\}, \{\neg x_1, x_3\}, \{\neg x_2, \neg x_3\}, \{\neg x_2, x_3\} \}$ qui est une formule de Horn.

Proposition 11 *Si $\mathcal{F}$ est une formule binaire, alors on peut générer tous ses modèles à délai $O(nN)$.*

Preuve : Il suffit d'appliquer le test de satisfaisabilité (Prop. 9) sur $\mathcal{F}$. Si celui-ci est négatif, alors on peut dire en temps linéaire que $\mathcal{F}$ n'admet aucun modèle. Si ce test est positif, alors la Proposition 10 implique que $\mathcal{F}$ est Horn-renommable, on applique donc l'algorithme Génération. Le corollaire 4 ainsi que les résultats de la section 2.3 impliquent le résultat. □

2.5 Formules équilibrées

Conforti et Cornuéjols [14] ont présenté une classe de formules pour laquelle le test de satisfaisabilité se fait à l'aide uniquement de la résolution unitaire. Nous allons ici prouver que l'on peut générer toutes les solutions de telles formules à délai $O(nN)$.

Définition 4 (formule équilibrée) *Considérons la formule $\mathcal{F} = \{C_1, \ldots, C_m\}$ sur un ensemble $V = \{x_1, \ldots, x_n\}$. Associons à $\mathcal{F}$ la $(0, \pm 1)$-matrice M suivante. Les lignes de M sont indexées par les clauses de $\mathcal{F}$ et les colonnes sont indexées par les variables de telle façon que M_{ij} soit un $+1$ si $x_j \in C_i$, un -1 si $\neg x_j \in C_i$ et un 0 sinon. La formule $\mathcal{F}$ est* équilibrée *si pour chaque sous-matrice carrée de M ayant exactement deux entrées non nulles par ligne et par colonne, la somme des entrées est un multiple de quatre.*

Exemple : Soit $\mathcal{F}_5 = \{\{\neg x_1, x_2\}, \{\neg x_1, x_2, x_3\}, \{x_2, x_3\}\}$. On va voir ici que $\mathcal{F}_5$ est équilibrée. La matrice associée à la formule $\mathcal{F}_5$ est :

$$\begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{pmatrix} -1 & 1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix} \end{array} \tag{2.1}$$

et ses seules sous-matrices carrées ayant exactement deux entrées non nulles par ligne et par colonne sont :

$$\begin{array}{c} \\ C_1 \\ C_2 \end{array} \begin{array}{c} \begin{array}{cc} x_1 & x_2 \end{array} \\ \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix} \end{array} \tag{2.2}$$

$$\begin{array}{c} \\ C_2 \\ C_3 \end{array} \begin{array}{c} \begin{array}{cc} x_2 & x_3 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array} \tag{2.3}$$

La sous-matrice 2.2 correspondant aux variables x_1, x_2 et aux clauses C_1, C_2 a une somme égale à zéro ($= 0 * 4$). La sous-matrice 2.3 correspondant aux variables x_2, x_3 et aux clauses C_2, C_3 est telle que la somme de ses entrées est égale à quatre. $\mathcal{F}_5$ est donc une formule équilibrée.

On peut tester en temps polynômial si une formule est équilibrée. En effet Conforti et al. [15] ont proposé un algorithme polynomial pour reconnaitre si une matrice $0, \pm 1$ est équilibrée. Comme la construction de la matrice se fait en temps $O(n^2)$ et son remplissage en $O(N)$, il est donc possible de tester efficacement si une formule est équilibrée.

Proposition 12 *Soit $\mathcal{U}$ un ensemble fini de clauses unitaires, si $\mathcal{F}$ est équilibrée, alors $Noyau(\mathcal{F} \cup \mathcal{U})$ est équilibrée.*

Preuve : Soit M' la matrice correspondant à la formule $\mathcal{F}' = Noyau(\mathcal{F} \cup \mathcal{U})$. La formule $\mathcal{F}'$ est égale à $\mathcal{F}$ sauf que l'on a effacé des clauses et certains littéraux. M' est donc une sous-matrice de M (on a effacé des lignes et des colonnes à M pour construire M'). Donc toute sous-matrice carrée de M' est sous-matrice carrée de M ce qui implique que la propriété est vérifiée pour $\mathcal{F}'$, donc $\mathcal{F}'$ est équilibrée. □

Proposition 13 *Soit $\mathcal{F}$ un formule équilibrée, si $\mathcal{F}$ ne contient pas de clause unitaire, alors $\mathcal{F}$ est satisfaisable.*

Preuve : Conforti et Cornuéjuols [14] ont prouvé que pour toute formule équilibrée $\mathcal{F}$, si toute clause de $\mathcal{F}$ contient au moins deux littéraux, alors pour toute variable x_j, il existe au moins deux modèles satisfaisant $\mathcal{F}$, un contenant x_j et un autre contenant $\neg x_j$. ([14] Page 673 remark 3.3) □

Exemple : $M = \{\neg x_1, x_2, x_3\}$ est un modèle de $\mathcal{F}_5$.

La classe des formules équilibrées vérifie donc les deux propriétés P1 et P2, on peut donc générer tous les modèles de toute formule équilibrée à délai $O(nN)$ (Corollaire 4).

2.6 Conclusion

Dans les parties II et III de cette thèse nous allons nous intéresser à des classes de formules qui vérifient les propriétés P1 et P2. C'est à dire des formules pour lesquelles on a un algorithme à délai $O(nN)$ pour générer toutes les solutions. Le lecteur peut passer les chapitres suivants et poursuivre sa lecture directement avec les parties II et III.

Chandru et Hooker [12] ont présenté la classe des formules Horn étendues. Nous étudions cette classe dans la partie II. Dans le premier chapitre de cette partie, nous étudions la définition et l'origine de cette classe de formules. C'est aussi l'occasion de vérifier que l'on peut générer tous les modèles de telles formules efficacement. Malheureusement il n'existe pas encore d'algorithme polynômial permettant de tester si une formule est Horn étendue. C'est la raison pour laquelle, Swaminathan et Wagner [46] ont présenté la classe des formules Horn étendues simples. Cette classe est une restriction de la classe des formules Horn étendues, qui vérifie elle aussi les propriétés P1 et P2, mais pour laquelle ils ont présenté un algorithme de reconnaissance quadratique. Nous présentons même dans cette thèse (partie II) un algorithme de reconnaissance linéaire.

Dans la partie III nous présentons une nouvelle classe, les formules ordonnées. Cette classe vérifie à la fois P1 et P2, donc on peut générer ses modèles avec un délai $O(nN)$.

Chapitre 3

Générer en utilisant un ordre sur les variables

Sommaire

3.1 Introduction

On a vu au Chap. 2 qu'il était possible de générer toutes les solutions des formules de certaines classes en utilisant uniquement la résolution unitaire. Pour que cela soit possible, il faut que la classe vérifie les propriétés P1 et P2. P1 signifie que si $\mathcal{F}$ appartient à la classe, alors pour tout ensemble de clauses unitaires $\mathcal{U}$, $Noyau(\mathcal{F} \cup \mathcal{U})$ appartient aussi à la classe. P2 signifie que si $\mathcal{F}$ appartient à la classe et ne contient pas de clause unitaire, alors $\mathcal{F}$ est satisfaisable.

Nous allons étudier une nouvelle extension de Horn dans ce chapitre. Il s'agit de la classe presque Horn, qui est issue des travaux de Hébrard et Luquet [32] sur la base de Horn. Ils ont remarqué que même si une formule n'était pas Horn renommable, il était possible de renommer partiellement cette formule ce qui permet de simplifier la formule et donc de résoudre plus facilement le problème de satisfaisabilité. En itérant ce principe, on peut trouver une classe de formules (que nous avons appelée presque Horn) pour laquelle on détermine la satisfaisabilité des formules en temps linéaire. Malheureusement, la classe presque Horn ne vérifie pas la propriété P1, c'est à dire qu'il existe des formules presque Horn et des ensembles $\mathcal{U}$ tels que $Noyau(\mathcal{F} \cup \mathcal{U})$ n'est pas presque Horn. Mais dans l'algorithme de génération (Fig. 1.1) les ensembles $\mathcal{U}$ ne sont pas quelconques, ils sont construits en utilisant un ordre arbitraire donné sur

les variables. On va prouver ici que pour toute formule $\mathcal{F}$, sous réserve du calcul d'un ordre acceptable sur les variables, pour tout ensemble $\mathcal{U}$ utilisé dans l'algorithme, si $\mathcal{F}$ est presque Horn alors $Noyau(\mathcal{F} \cup \mathcal{U})$ est aussi presque Horn. Ceci nous permet donc de générer à délai $O(nN)$ (où N représente la taille totale de la formule et n le nombre de ses variables) tous les modèles de toute formule presque Horn.

Nous étudions ensuite la classe q-Horn qui a été introduite par Boros et al. [6]. Cette classe généralise à la fois les formules binaires et les formules de Horn. On remarque que toute formule q-Horn satisfaisable est aussi presque Horn. Comme il existe un algorithme linéaire pour tester la satisfaisabilité d'une formule q-Horn, nous présentons un algorithme à délai $O(nN)$ permettant de générer tous les modèles d'une formule q-Horn.

3.2 Formules presque Horn

Si une formule n'est pas Horn renommable, elle peut être partiellement renommable. Hébrard et Luquet [32] ont étudié le phénomène des formules partiellement renommables, ils en ont extrait le concept de base de Horn. Nous utilisons ici ce concept pour introduire une nouvelle classe de formules, les formules *presque Horn*. Nous présentons un algorithme linéaire permettant de tester la satisfaisabilité d'une telle formule. Dans cette section, nous allons rappeler les principaux résultats de Hébrard et Luquet et nous allons montrer que l'on peut, moyennant l'introduction d'un ordre sur les variables, calculer tous les modèles d'une formule appartenant à cette classe avec un délai $O(nN)$.

Définition 5 (formule X-Horn) *Soit $X \subseteq V$. $\mathcal{F}$ est* X-Horn *si toute clause de $\mathcal{F}$ qui contient un littéral positif de $Lit(X)$ est une clause de Horn sur X (i.e. ne contient que des littéraux de $Lit(X)$ et contient au maximum un littéral positif).*

Exemple : Soit $\mathcal{F}_1 = \{ \{x_5, x_6\}, \{x_5, x_6, \neg x_7\}, \{x_3, \neg x_4\}, \{\neg x_3, x_4\}, \{x_1, x_2, \neg x_3\}, \{\neg x_1, \neg x_2, x_6\}, \{\neg x_1, x_2\}, \{x_1, \neg x_2\}\}$. $\mathcal{F}_1$ est $\{x_3, x_4\}$-Horn, en effet les clauses qui contiennent des littéraux positifs sur x_3 et x_4, sont des clauses de Horn sur les variables x_3 et x_4.

On remarque que les clauses d'une formule X-Horn peuvent être de trois types :
• clauses de Horn sur X ;
• clauses sur V contenant des littéraux négatifs de $Lit(X)$ avec des littéraux de $Lit(V \setminus X)$, mais ne contenant aucun littéral positif de $Lit(X)$;
• clauses sur $V \setminus X$.

Définition 6 (formule X-Horn-renommable, Reste$(\mathcal{F}, X)$) *Soit $X \subseteq V$. $\mathcal{F}$ est* X-Horn-renommable *si on peut transformer $\mathcal{F}$ en une formule X-Horn en renommant des variables de X. On utilise la notation $Reste(\mathcal{F}, X)$ pour l'ensemble $\{C \in \mathcal{F} \mid C \cap Lit(X) = \emptyset\}$.*

Exemple : La formule $\mathcal{F}_1$ est $\{x_5, x_6, x_7\}$-Horn-renommable (on renomme la variable x_6).

Hébrard et Luquet [32] ont étudié la notion de X-Horn-renommabilité, ils ont prouvé les deux propositions suivantes et ont découvert la notion de base de Horn. Ils ont remaqué que la X-Horn-renommabilité d'une formule permettait, pour l'étude du problème de satisfaisabilité, de se ramener à l'étude d'une formule plus simple.

Proposition 14 *Si $\mathcal{F}$ est X-Horn-renommable et ne contient pas de clause unitaire, alors $\mathcal{F}$ est satisfaisable ssi $Reste(\mathcal{F}, X)$ est satisfaisable.*

Preuve : ($\Rightarrow$) Immédiat, car $Reste(\mathcal{F}, X) \subseteq \mathcal{F}$.

($\Leftarrow$) Cas particulier : $\mathcal{F}$ est X-Horn. Soit M un modèle (sur $V \setminus X$) de $Reste(\mathcal{F}, X)$ et $I = \{\neg p \mid p \in X\}$. Alors $I \cup M$ est un modèle de $\mathcal{F}$. En effet, pour tout $C \in \mathcal{F}$, soit on a $C \in Reste(\mathcal{F}, X)$ et $C \cap M \neq \emptyset$, ou alors $C \cap I \neq \emptyset$ (car $\mathcal{F}$ est X-Horn et $card(C) \geq 2$). Cas général : $\mathcal{F}$ est X-Horn-renommable. Se déduit du cas précédent car le renommage préserve la satisfaisabilité. □

Proposition 15 *Soient $X_1 \subseteq V$ et $X_2 \subseteq V$. Si $\mathcal{F}$ est X_1-Horn-renommable et X_2-Horn-renommable, alors $\mathcal{F}$ est $(X_1 \cup X_2)$-Horn-renommable.*

Preuve : On peut trouver une preuve de cette proposition dans [32], une démonstration du même type de résultat est en outre présentée en Partie III Chap. 2 Prop. 83. □

La proposition précédente, implique que pour toute formule $\mathcal{F}$, il existe un ensemble canonique B ($B \subseteq V$) tel que $\mathcal{F}$ est B-Horn-renommable et tel que pour tout X, si $\mathcal{F}$ est X-Horn-renommable, alors $X \subseteq B$.

Définition 7 (base de Horn, Reste$(\mathcal{F})$) *Soient $X_1, \ldots, X_k$ tous les sous-ensembles de V tels que $\mathcal{F}$ est X_i-Horn-renommable et $B = X_1 \cup \ldots \cup X_k$. La Proposition 15 implique que $\mathcal{F}$ est B-Horn-renommable. L'ensemble B sera appelé* la base de Horn de $\mathcal{F}$ *et noté $Base(\mathcal{F})$. On définit en outre l'ensemble, $Reste(\mathcal{F}) = Reste(\mathcal{F}, Base(\mathcal{F}))$.*

Exemple : On a vu que $\mathcal{F}_1$ est $\{x_3, x_4\}$-Horn (donc $\{x_3, x_4\}$-Horn-renommable) et $\{x_5, x_6, x_7\}$-Horn-renommable, $\mathcal{F}_1$ est donc $\{x_3, x_4, x_5, x_6, x_7\}$-Horn-renommable. On peut prouver que $Base(\mathcal{F}_1) = \{x_3, x_4, x_5, x_6, x_7\}$. On

peut donc calculer $Reste(\mathcal{F}_1) = \{ \{\neg x_1, x_2\}, \{x_1, \neg x_2\} \}$.

Corollaire 16 *Si $\mathcal{F}$ ne contient pas de clause unitaire, alors $\mathcal{F}$ est satisfaisable si et seulement si $Reste(\mathcal{F})$ est satisfaisable.*

On remarque donc que si $Reste(\mathcal{F}) = \emptyset$ et que $\mathcal{F}$ ne contient pas de clause unitaire, alors la formule $\mathcal{F}$ est satisfaisable. On va se servir de ce résultat pour définir une nouvelle classe de formules, les formules presque Horn.

Si $Base(Reste(\mathcal{F}))$ est non vide, alors $Reste(Reste(\mathcal{F}))$ est un sous ensemble strict de $Reste(\mathcal{F})$. On peut répéter ce processus tant que l'on obtient des formules dont la base de Horn est non vide.

Définition 8 (formule presque Horn) *Soit* Reste-itéré($\mathcal{F}$) *le sous-ensemble de $\mathcal{F}$ défini récursivement par : si $Base(\mathcal{F}) = \emptyset$ alors* Reste-itéré($\mathcal{F}$) = $\mathcal{F}$ *sinon* Reste-itéré($\mathcal{F}$) = Reste-itéré($Reste(\mathcal{F})$).

Une formule $\mathcal{F}$ est presque Horn, *si Reste-itéré($\mathcal{F}$) = $\emptyset$.*

Exemple : $\mathcal{F}_1$ est presque Horn, car $Reste(\mathcal{F}_1)$ est une formule de Horn, donc $Reste(Reste(\mathcal{F}_1)) = \emptyset$.

Corollaire 17 *Soit $\mathcal{F}$ une formule presque Horn, si $\mathcal{F}$ ne contient pas de clause unitaire, alors $\mathcal{F}$ est satisfaisable.*

Malheureusement, la propriété P1 n'est pas vérifiée pour la classe presque Horn.
Exemple : Soit $\mathcal{U} = \{\{x_3\}, \{\neg x_6\}\}$, $Unit(\mathcal{F}_1 \cup \mathcal{U}) = \{x_3, x_4, x_5, \neg x_6\}$ et donc $Noyau(\mathcal{F}_1 \cup \mathcal{U}) = \{\{x_1, x_2\}, \{x_1, \neg x_2\}, \{\neg x_1, x_2\}, \{\neg x_1, \neg x_2\}\}$. Quel que soit l'ensemble $X \subseteq \{x_1, x_2\}$, cette formule n'est pas X-Horn-renommable, donc $Base(Noyau(\mathcal{F}_1 \cup \mathcal{U})) = \emptyset$, et donc $Noyau(\mathcal{F}_1 \cup \mathcal{U})$ n'est pas une formule presque Horn.

Nous présentons maintenant une adaptation de l'algorithme de génération pour qu'il puisse donner tous les modèles de toute formule presque Horn.

Nous effectuons le pré-calcul d'un ordre sur les variables qui nous permet ensuite d'appliquer l'algorithme de génération sans risque de tomber dans le problème soulevé par l'exemple ci-dessus.

Nous présentons des résultats intermédiaires qui permettent de mieux apréhender la notion de formule presque Horn. Tout d'abord, on remarque que si une formule est incluse dans une autre, alors son *Reste* est inclus dans le *Reste* de l'autre formule.

Proposition 18 *Si $\mathcal{F}' \subseteq \mathcal{F}$ alors $Reste(\mathcal{F}') \subseteq Reste(\mathcal{F})$.*

Preuve : On va faire une preuve par l'absurde. Supposons que $Reste(\mathcal{F}') \not\subseteq Reste(\mathcal{F})$. Alors il existe une clause C telle que $C \in Reste(\mathcal{F}')$ et $C \notin Reste(\mathcal{F})$. Mais comme $Reste(\mathcal{F}') \subseteq \mathcal{F}'$ (par définition du *Reste*), on a $C \in \mathcal{F}'$, donc $C \in \mathcal{F}$ (car $\mathcal{F}' \subseteq \mathcal{F}$). Comme $C \notin Reste(\mathcal{F})$, on a $C \cap Lit(Base(\mathcal{F})) \neq \emptyset$. Soit $Y = Base(\mathcal{F}) \cap var(\mathcal{F}')$ (on a $Y \neq \emptyset$ car $C \in \mathcal{F}'$ et $C \cap Lit(Base(\mathcal{F})) \neq \emptyset$). Par définition de la X-Horn-renommabilité, pour toute clause $C \in \mathcal{F}$, $pos(C) \cap Base(\mathcal{F}) \neq \emptyset$ implique que $var(C) \subseteq Base(\mathcal{F})$. Or $\mathcal{F}' \subseteq \mathcal{F}$, donc pour toute clause $C \in \mathcal{F}'$, $pos(C) \cap Base(\mathcal{F}) \neq \emptyset$ implique que $var(C) \subseteq Base(\mathcal{F})$. Comme $Y = Base(\mathcal{F}) \cap var(\mathcal{F}')$, pour toute clause $C \in \mathcal{F}'$, $pos(C) \cap Y \neq \emptyset$ implique que $var(C) \subseteq Base(\mathcal{F})$. Mais pour toute clause $C \in \mathcal{F}'$, $var(C) \subseteq var(\mathcal{F}')$, donc $var(C) \subseteq Base(\mathcal{F})$ est équivalent à $var(C) \subseteq Y$. Donc pour toute clause $C \in \mathcal{F}'$, $pos(C) \cap Y \neq \emptyset$ implique que $var(C) \subseteq Y$. Donc $\mathcal{F}'$ est Y-Horn-renommable. Donc $Y \subseteq Base(\mathcal{F}')$ (par définition de la base de Horn). De plus $var(Reste(\mathcal{F}')) \cap Base(\mathcal{F}') = \emptyset$ (par définition du Reste). D'où $var(Reste(\mathcal{F}')) \cap Y = \emptyset$ et finalement $var(Reste(\mathcal{F}')) \cap Base(\mathcal{F}) = \emptyset$. Contradiction. □

Proposition 19 *Si $\mathcal{F}' \subseteq \mathcal{F}$ et $\mathcal{F}$ est presque Horn, alors $\mathcal{F}'$ est presque Horn.*

Preuve : Si *Reste-itéré*$(\mathcal{F}) = \emptyset$, alors la Prop. 18 implique que *Reste-itéré*$(\mathcal{F}') = \emptyset$. □

Nous présentons une définition alternative pour les formules presque Horn, cette définition n'utilise pas le concept de base de Horn, mais se contente de la notion plus faible de X-Horn-renommabilité.

Proposition 20 *$\mathcal{F}$ est presque Horn ssi il existe $X_1, \ldots, X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ $(1 \leq i \leq k)$, tels que $\mathcal{F}_1 = \mathcal{F}$, $\mathcal{F}_i$ est X_i-Horn-renommable $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$.*

Preuve : ($\Rightarrow$) Immédiat, il suffit de prendre $X_i = Base(\mathcal{F}_i)$ pour tout i, avec $\mathcal{F}_1 = \mathcal{F}$ et $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i)$.

($\Leftarrow$) On raisonne par récurrence sur k.

$\underline{k = 1}$ On a $\mathcal{F} = \emptyset$, $Reste(\mathcal{F}) = \emptyset$ et $\mathcal{F}$ est presque Horn.

$\underline{k > 1}$ Par hypothèse de récurrence, $\mathcal{F}_2$ est presque Horn. Par définition de la base de Horn, $Reste(\mathcal{F}_1) \subseteq \mathcal{F}_2$. Donc $Reste(\mathcal{F}_1)$ est presque Horn (Prop. 19), *Reste-itéré*$(\mathcal{F}_1) = \emptyset$ et $\mathcal{F}_1$ est presque Horn. □

Remarque 1 *Soit $\mathcal{F}'$ une formule obtenue en renommant des variables dans $\mathcal{F}$. $\mathcal{F}$ est presque Horn ssi $\mathcal{F}'$ est presque Horn.*

On a vu que même si $\mathcal{F}$ est presque Horn, il existe des ensembles de clauses unitaires $\mathcal{U}$ tels que $Noyau(\mathcal{F} \cup \mathcal{U})$ n'est pas presque Horn. Nous présentons ici une caractérisation d'ensembles $\mathcal{U}$ tels que $Noyau(\mathcal{F} \cup \mathcal{U})$ soit toujours presque Horn.

Définition 9 (permutation convenable, ensemble convenable) *Supposons que $\mathcal{F}$ est presque Horn. Il existe $X_1, \ldots, X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ $(1 \leq i \leq k)$, tels que $\mathcal{F}_1 = \mathcal{F}$, $X_i = Base(\mathcal{F}_i)$ $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Soit $W = V \setminus (X_1 \cup \ldots \cup X_k)$. Une permutation $(x_1, \ldots, x_n)$ des variables de $\mathcal{F}$ est dite* convenable *si pour tout j $(1 \leq j \leq n)$, $\{x_1, \ldots, x_j\} \subseteq W$ ou il existe i tel que $\{x_1, \ldots, x_j\} = W \cup X_k \cup X_{k-1} \cup \ldots \cup X_{i+1} \cup X$ avec $X \subseteq X_i$. Un ensemble $\mathcal{U}$ de clauses unitaires est dit* convenable *s'il existe un permutation convenable $(x_1, \ldots, x_n)$ et un entier $i \in \{1, \ldots, n\}$, tels que $var(\mathcal{U}) = \{x_j \mid 1 \leq j \leq i\}$.*

Exemple : Pour la formule $\mathcal{F}_1$ qui nous sert d'exemple, on a $W = \emptyset$, $X_1 = \{x_3, x_4, x_5, x_6, x_7\}$ et $X_2 = \{x_1, x_2\}$. L'ensemble $\mathcal{U} = \{\{x_3\}, \{\neg x_6\}\}$ n'est pas un ensemble convenable, car pour toute permutation π, si x_3 et x_6 sont les deux premiers éléments de π, alors π n'est pas convenable, car on a $\{x_3, x_6\} \subseteq X_1$, donc ne vérifie pas la définition d'une permutation convenable. On a remarqué à l'exemple précédent, que dans ce cas, $Noyau(\mathcal{F}_1 \cup \mathcal{U})$ n'est pas presque Horn. L'ensemble $\mathcal{V} = \{\{x_1\}, \{x_2\}, \{\neg x_3\}\}$ est un ensemble convenable, car $var(\mathcal{V}) = X_2 \cup X$ avec $X \subseteq X_1$. On remarque qu'un ensemble est convenable, si lorsqu'il contient un élément de $Lit(Base(\mathcal{F}_1))$, alors il contient un littéral correspondant à chaque variable n'appartenant pas à la base de Horn. On remarque que $Noyau(\mathcal{F}_1 \cup \mathcal{V}) = \{\ \{x_5, x_6\}, \{x_5, x_6, \neg x_7\}, \{x_6\}\ \}$ est une formule presque Horn (elle est même Horn renommable).

Proposition 21 *Si $\mathcal{F}$ est une formule presque Horn, et ne contient pas de clause unitaire, alors pour tout ensemble convenable de littéraux $\mathcal{U}$, $Noyau(\mathcal{F} \cup \mathcal{U})$ est presque Horn.*

Preuve : Nous allons prouver que $Noyau(\mathcal{F} \cup \mathcal{U})$ est presque Horn. La proposition 20 implique qu'il existe $X_1, \ldots, X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ $(1 \leq i \leq k)$, tels que $\mathcal{F}_1 = \mathcal{F}$, $X_i = Base(\mathcal{F}_i)$ $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Pour tout i $(1 \leq i \leq k)$, $\mathcal{F}_i$ est X_i-Horn-renommable. Sans perte de généralité, on peut supposer que $\mathcal{F}_i$ est X_i-Horn $(1 \leq i \leq k)$ (cf Remarque 1). Soit $\mathcal{G}_i$ $(1 \leq i \leq k)$ la formule obtenue, a partir de $\mathcal{F}_i$ en supprimant toutes les clauses $C \in \mathcal{F}$ telles que $C \cap Unit(\mathcal{F} \cup \mathcal{U}) \neq \emptyset$, et en ôtant des clauses restantes tout littéral t tel que $\overline{t} \in Unit(\mathcal{F} \cup \mathcal{U})$. Soit $Y_i = X_i \cap var(\mathcal{G}_i)$ $(1 \leq i \leq k)$. On obtient donc $\mathcal{G}_1 = Noyau(\mathcal{F} \cup \mathcal{U})$, $\mathcal{G}_i \subseteq \mathcal{G}_1$ $(1 \leq i \leq k)$

et $\mathcal{G}_k = \emptyset$. Il suffit de prouver que $\mathcal{G}_i$ est Y_i-Horn $(1 \leq i \leq k)$ et que $\mathcal{G}_{i+1} = Reste(\mathcal{G}_i, Y_i)$ $(1 \leq i \leq k-1)$ (Prop. 20).

On prouve dans un premier temps que $\mathcal{G}_i$ est Y_i-Horn $(1 \leq i \leq k)$. Soit $C' \in \mathcal{G}_i$. La définition de $\mathcal{G}_i$ nous donne qu'il existe $C \in \mathcal{F}_i$ tel que $C' \subseteq C$, $C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$, et pour tout $t \in C \setminus C'$, $\overline{t} \in Unit(\mathcal{F} \cup \mathcal{U})$. Soit $l \in C'$. Si $l \in Lit(Y_i)$ et l est un littéral positif,alors par définition de Y_i, $l \in Lit(X_i)$, et $C \subseteq Lit(X_i)$ puisque $\mathcal{F}_i$ est X_i-Horn ; par conséquent $C' \subseteq Lit(Y_i)$. Par définition des formules X_i-Horn, on sait que C ne contient qu'un seul littéral positif. Puisque $C' \subseteq C$, on peut conclure que C' est une clause de Horn sur Y_i. Donc $\mathcal{G}_i$ est une formule Y_i-Horn.

On prouve maintenant que $\mathcal{G}_{i+1} = Reste(\mathcal{G}_i, Y_i)$ $(1 \leq i \leq k-1)$. $(\subseteq)$ On a $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$. Donc $\mathcal{F}_{i+1} \subseteq \mathcal{F}_i$ et $\mathcal{G}_{i+1} \subseteq \mathcal{G}_i$. Soit $C' \in \mathcal{G}_{i+1}$. Il existe $C \in \mathcal{F}_{i+1}$ tel que $C' \subseteq C$. On a $C \in Reste(\mathcal{F}_i, X_i)$, d'où $C \cap Lit(X_i) = \emptyset$, $C' \cap Lit(Y_i) = \emptyset$ et $C' \in Reste(\mathcal{G}_i, Y_i)$. $(\supseteq)$ Soit $C' \in Reste(\mathcal{G}_i, Y_i)$. Il existe $C \in \mathcal{F}_i$ tel que $C' \subseteq C$, $C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$, et pour tout $t \in C \setminus C'$, $\overline{t} \in Unit(\mathcal{F} \cup \mathcal{U})$. Soit $l \in C'$. On a $var(l) \notin Y_i$, $l \in C$ et $var(l) \notin X_i$. Donc $var(l) \notin X_j$ $(1 \leq j \leq i)$. De plus $var(l) \notin var(\mathcal{U})$ puisque $l \notin Unit(\mathcal{F} \cup \mathcal{U})$ et $\overline{l} \notin Unit(\mathcal{F} \cup \mathcal{U})$. Donc $var(\mathcal{U}) \cap X_j = \emptyset$ $(1 \leq j \leq i)$ (permet d'utiliser le lemme 22). Or $\mathcal{U}$ est convenable. Supposons que $C \notin Reste(\mathcal{F}_i, X_i)$. Alors il existe $\neg x \in C$ tel que $x \in X_i$. On a $\neg x \notin C'$, d'où $x \in Unit(\mathcal{F} \cup \mathcal{U})$. Impossible (Lemme 22). D'où on déduit que $C \in Reste(\mathcal{F}_i, X_i)$, $C \in \mathcal{F}_{i+1}$ et $C' \in \mathcal{G}_{i+1}$. □

Lemme 22 *Supposons que $\mathcal{F}$ ne contienne pas de clause unitaire, et qu'il existe $X_1, \ldots, X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ $(1 \leq i \leq k)$, tels que $\mathcal{F}_1 = \mathcal{F}$, $\mathcal{F}_i$ est X_i-Horn $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Soit $\mathcal{U}$ un ensemble fini de clauses unitaires tel que $Unit(\mathcal{F} \cup \mathcal{U})$ est cohérent, et $i_0 \in \{1, \ldots, k\}$ tel que pour tout j $(1 \leq j \leq i_0)$ $var(\mathcal{U}) \cap X_j = \emptyset$. Alors $X_{i_0} \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$.*

Preuve : On va prouver par récurrence sur p que pour toute dérivation unitaire σ de longueur p de $\mathcal{F} \cup \mathcal{U}$, σ ne contient pas de clause unitaire positive $\{x\}$ telle que $x \in X_{i_0} \cap Unit(\mathcal{F} \cup \mathcal{U})$.

$\underline{p = 1}$ Les dérivations de longueur 1 ne contiennent que des clauses de $\mathcal{F} \cup \mathcal{U}$, l'hypothèse de récurence est donc vérifiée pour $p = 1$.

$\underline{p > 1}$ Supposons qu'il existe x un littéral positif et σ une dérivation de longueur p, $\sigma = (C_1, \ldots, C_p)$ tels que $C_p = \{x\}$ et $x \in X_{i_0}$. σ contient une clause $C = \{x, l_1, \ldots, l_h\}$ (avec $h \geq 1$) et les clauses unitaires $\{\overline{l_1}\}, \ldots, \{\overline{l_h}\}$. On va prouver que C ne contient pas de littéraux négatifs. Soit $\neg z \in C$ tel que $z \in X_j$ et j est minimal. On a $j \leq i_0$ puisque $x \in X_{i_0}$ et σ contient une clause unitaire positive $C_i = \{z\}$ avec $i < p$, ce qui est impossible par hypothèse de récurrence. C ne contient donc que des littéraux positifs. $C \in \mathcal{F}_{i_0}$ donc C est Horn sur X_{i_0}, ce qui est impossible puisque $h \geq 1$.

La propriété est donc vraie pour tout p. □

Proposition 23 *Si $\mathcal{F}$ est presque Horn et ne contient pas de clause unitaire alors pour tout ensemble convenable de littéraux $\mathcal{U}$, $\mathcal{F} \cup \mathcal{U}$ est satisfaisable si et seulement si $Unit(\mathcal{F} \cup \mathcal{U})$ est cohérent.*

Preuve : ($\Longrightarrow$) Si $\mathcal{F} \cup \mathcal{U}$ est satisfaisable, alors $Unit(\mathcal{F} \cup \mathcal{U})$ est cohérent (Rem. 1).
($\Longleftarrow$) $Noyau(\mathcal{F} \cup \mathcal{U})$ est presque Horn (Prop.21), donc satisfaisable (Coro.17), $Unit(\mathcal{F} \cup \mathcal{U})$ est cohérent (par hypothèse). La remarque 1 implique que $\mathcal{F} \cup \mathcal{U}$ est satisfaisable. □

Proposition 24 *On peut tester si une formule $\mathcal{F}$ est presque Horn et, si c'est le cas, construire une permutation convenable en temps $O(nN)$.*

Preuve : Hébrard and Luquet[32] ont présenté un algorithme pour le calcul de la base de Horn d'une formule. Ils ont montré qu'on pouvait calculer la base de Horn d'une formule en temps linéaire. On peut observer que si $Base(\mathcal{F})$ est connue, alors il est facile de calculer $Reste(\mathcal{F})$ en temps $O(N)$. On obtient donc $Reste(\mathcal{F})$ en temps $O(N)$. Si $Base(\mathcal{F})$ n'est pas vide, alors l'ensemble des variables de $Reste(\mathcal{F})$ est strictement inclus dans V. En conséquence, le calcul du Reste-itéré($\mathcal{F}$) requière au maximum n étapes et est donc exécuté en temps $O(nN)$. On obtient, comme sous-produit de ce calcul, les ensembles $X_1, \ldots, X_k$ et les formules $\mathcal{F}_1, \ldots, \mathcal{F}_k$ tels que $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Il est facile de construire en temps $O(n)$ une permutation convenable à partir des ensembles $X_1, \ldots, X_k$ et de $V \setminus (X_1 \cup \ldots \cup X_k)$. □

Proposition 25 *On peut générer à délai $O(nN)$ les modèles de toute formule presque Horn.*

Preuve : L'algorithme Génération (Fig. 1.1) peut être appelé avec n'importe quelle permutation. Si on utilise une permutation convenable, à chaque pas de l'algorithme $\mathcal{F} \cup \mathcal{U} \cup \{\{l_i\}\}$ est satisfaisable ssi $Unit(\mathcal{F} \cup \mathcal{U} \cup \{\{l_i\}\})$ est cohérent (où l_i est x_i ou $\neg x_i$) (Prop. 23). On peut en outre tester en temps $O(N)$ si $Unit(\mathcal{F} \cup \mathcal{U} \cup \{\{l_i\}\})$ est cohérent. Le reste de la preuve est similaire à celle de la Prop. 2. L'algorithme Génération donne donc tous les modèles d'une formule presque Horn avec un délai $O(nN)$. □

3.3 Formules q-Horn

La classe des formules *q-Horn* a été présentée par Boros et al. [6]. Cette classe est à la fois une généralisation des formules de Horn et des formules binaires. Boros et al. ont présenté un algorithme linéaire pour résoudre le problème de la satisfaisabilité d'une formule q-Horn. Boros et al. [8] ont par ailleurs proposé un algorithme linéaire assez compliqué pour tester si une formule est q-Horn. Toutefois, Hébrard et Luquet [32] ont remarqué le lien entre une formule q-Horn et la notion de base de Horn présentée à la Section 3.2.

Boros et al. [6] donnent une définition fonctionnelle pour les formules q-Horn, nous utiliserons de préférence ici la caractérisation syntaxique mise en évidence dans ce même papier.

Définition 10 (q-Horn) *$\mathcal{F}$ est q-Horn si V (son ensemble de variables) peut être découpé en deux ensembles X et Y tels que les clauses de $\mathcal{F}$ appartiennent à un des deux types suivants :*

- *clauses contenant au plus un littéral positif sur X et pas de littéral sur Y.*
- *clauses ne contenant aucun littéral positif sur X, et au plus deux littéraux (positifs ou négatifs) sur Y.*

Pour se rapprocher de notre caractérisation des formules X-Horn, on peut aussi l'écrire de la façon suivante. Chaque clause de $\mathcal{F}$ appartient à l'un de ces trois ensembles :

- clauses contenant au plus un littéral positif sur X et pas de littéral sur Y.
- clauses ne contenant aucun littéral positif sur X, mais au moins un littéral négatif sur X, et au plus deux littéraux (positifs ou négatifs) sur Y.
- clauses contenant au maximum deux littéraux (positifs ou négatifs) sur Y, et aucun littéral sur X.

On peut donc remarquer que si $\mathcal{F}$ est q-Horn, alors $\mathcal{F}$ est X-Horn. Cela nous conduit donc à une caractérisation des formules q-Horn présentée dans [32] qui utilise la notion de base de Horn. Une formule $\mathcal{F}$ est q-Horn, si et seulement si toute clause $C \in \mathcal{F}$ contient au maximum deux littéraux n'appartenant pas à la base de Horn de $\mathcal{F}$.
Exemple : Soit $\mathcal{F}_2 = \{ \{x_1, \neg x_2, \neg x_3\}, \{\neg x_1, \neg x_2, \neg x_3\}, \{\neg x_3, x_4\}, \{\neg x_1, \neg x_2, x_5, x_6\}, \{\neg x_1, \neg x_3, \neg x_6, x_7\}, \{\neg x_2, \neg x_4, x_6, x_7\}, \{x_6, \neg x_7\}, \{x_5, x_6\}\}$. La base de Horn de $\mathcal{F}_2$ est $\{x_1, x_2, x_3, x_4\}$. On peut vérifier que les clauses se répartissent en trois groupes : les trois premières clauses ne contiennent

que des littéraux sur $\{x_1, x_2, x_3, x_4\}$; les trois suivantes contiennent des littéraux négatifs sur $\{x_1, x_2, x_3, x_4\}$ et un ou deux littéraux (négatifs ou positifs) sur $\{x_5, x_6, x_7\}$; les deux dernières clauses contiennent deux littéraux sur l'ensemble $\{x_5, x_6, x_7\}$. Ces deux dernières clauses forment la formule $Reste(\mathcal{F}_2)$.

On peut en outre voir que pour toute formule q-Horn $\mathcal{F}$, $Reste(\mathcal{F})$ est une formule binaire. Si $Reste(\mathcal{F})$ est satisfaisable, alors $Reste(\mathcal{F})$ est Horn renommable (cf Section 2.4). Dans ce cas, $Reste(Reste(\mathcal{F})) = \emptyset$, donc $\mathcal{F}$ est presque Horn, et on peut appliquer l'algorithme proposé en section 3.2.

Proposition 26 *Si $\mathcal{F}$ est q-Horn, alors on peut générer à délai $O(nN)$ tous ses modèles.*

Preuve : Il suffit dans un premier temps de tester si $\mathcal{F}$ est satisfaisable (en utilisant l'algorithme de Boros et al. par exemple). Si la réponse est *non*, alors $\mathcal{F}$ n'admet aucun modèle, la proposition est donc vérifiée. Si la réponse est *oui*, alors $\mathcal{F}$ est presque Horn, donc la proposition 25 nous donne un algorithme permettant de générer tous les modèles de $\mathcal{F}$ à délai $O(nN)$. □

Chapitre 4

Générer en utilisant les résultats sur SAT

Sommaire

4.1 Introduction

Il existe des classes de formules pour lesquelles les propriétés P1 et P2 ne sont pas vérifiées. Nous allons dans cette section étudier les classes de formules dont la reconnaissance est polynômiale et qui vérifient la propriété P1, qui traduit une certaine stabilité de la classe par rapport à la résolution unitaire. C'est le cas des formules Horn généralisées ou encore des hiérarchies de Pretolani. On va voir que pour ces formules, il est aussi possible de générer toutes les solutions à délai polynômial. On ne peut malheureusement pas traiter ce problème en utilisant uniquement la résolution unitaire comme dans les chapitres précédents, mais il est tout de même possible en utilisant les résultats obtenus sur ces formules pour la résolution du problème SAT, de générer efficacement tous les modèles.

4.2 Formules Horn généralisées

Soient l un littéral et x une variable, dans cette section, on rappelle que l'on note $\mathcal{F}_l = \{C \setminus \{\bar{l}\} \mid C \in F, l \notin C\}$, cette formule correspond aux simplifications de la formule $\mathcal{F}$ dans laquelle on a affecté la valeur vraie au littéral l. On note $\mathcal{F} \setminus x = \{C \setminus \{x, \neg x\} \mid C \in F\}$ la formule dans laquelle on a effacé toutes les occurences de x et de $\neg x$.

Nous présentons ici les formules GHorn, qui ont été présentées par Yamasaki et Doshita [48].

Définition 11 (formule GHorn) *$\mathcal{F}$ est* GHorn *si $\mathcal{F}$ est Horn ou s'il existe une variable x (appelée candidat) telle que :*

1. *$\mathcal{F}_x$ est Horn.*
2. *$\mathcal{F} \setminus x$ est GHorn*

Exemple : Soit $\mathcal{F}_6 = \{\{x_1, x_2, \neg x_4, x_5\}, \{x_2, \neg x_3, x_4\}, \{\neg x_1, \neg x_3, \neg x_4, \neg x_5\}, \{\neg x_2, x_3, \neg x_4\}\}$. On va vérifier ici que $\mathcal{F}_6$ est GHorn. C'est la variable x_2 qui va nous servir de premier candidat. On vérifie que $(\mathcal{F}_6)_{x_2} = \{\{\neg x_1, \neg x_3, \neg x_4, \neg x_5\}, \{x_3, \neg x_4\}\}$ est bien Horn. Il reste à montrer que $\mathcal{F}_6 \setminus x_2$ est bien GHorn, avec $\mathcal{F}_6 \setminus x_2 = \{\{x_1, \neg x_4, x_5\}, \{\neg x_3, x_4\}, \{\neg x_1, \neg x_3, \neg x_4, \neg x_5\}, \{\neg x_3, \neg x_4\}\}$. On prend x_1 comme candidat, et on obtient : $(\mathcal{F}_6 \setminus x_2)_{x_1} = \{\{\neg x_3, x_4\}, \{\neg x_3, \neg x_4, \neg x_5\}, \{\neg x_3, \neg x_4\}\}$ est Horn, et la formule $\mathcal{F}_6 \setminus x_2, x_1 = \{\{\neg x_4, x_5\}, \{\neg x_3, x_4\}, \{\neg x_3, \neg x_4, \neg x_5\}, \{\neg x_3, \neg x_4\}\}$ est elle aussi Horn. On a donc bien que $\mathcal{F}_6$ est GHorn.

Gallo et Scutella [27] ont élargi cette classe et en ont tiré la hiérarchie de classes $\{\Gamma\}$ dont la classe de base est GHorn (i.e. Horn= Γ_0 et GHorn= Γ_1).

Définition 12 (hiérarchie $\{\Gamma\}$) *$\mathcal{F}$ appartient à la classe Γ_i si $\mathcal{F} \in \Gamma_{i-1}$ ou s'il existe une variable x telle que :*

1. *$\mathcal{F}_x \in \Gamma_{i-1}$*
2. *$\mathcal{F} \setminus x \in \Gamma_i$*

Gallo et Scutella [27] donnent un algorithme polynômial (pour k fixé) pour la reconnaissance des formules de la classe Γ_k.

Proposition 27 *Si $\mathcal{F}$ est une formule de la classe Γ_i, alors toute formule $\mathcal{F}'$ telle que pour toute clause $C' \in \mathcal{F}'$, il existe une clause $C \in \mathcal{F}$ avec $C' \subseteq C$, alors $\mathcal{F} \in \Gamma_i$.*

Preuve : Cette proposition est évidente au niveau 0, puisque Γ_0 =Horn et que si C contient au plus un littéral positif, alors il en est forcement de même pour C'.

Supposons cette proposition vraie pour tout $i \leq k-1$, on va prouver par récurrence qu'elle est vraie pour k.

La formule $\mathcal{F}$ est Γ_k, donc il existe une suite de variables $(x_1, \ldots, x_j)$ telle que : $\mathcal{F}_{x_1} \in \Gamma_{k-1}$, $(\mathcal{F} \setminus x_1)_{x_2} \in \Gamma_{k-1}$, $\ldots (\mathcal{F} \setminus x_1 \ldots \setminus x_{j-1})_{x_j} \in \Gamma_{k-1}$ et $\mathcal{F} \setminus x_1 \ldots \setminus x_j \in \Gamma_{k-1}$. $\mathcal{F}'_{x_1}$ est une formule dont chaque clause est incluse dans une clause de $\mathcal{F}_{x_1}$ (par hypothèse de la proposition), donc

par hypothèse de récurrence ${\mathcal{F}'}_{x_1}$ est Γ_{k-1}, il en est de même pour toutes les formules $(\mathcal{F}' \setminus x_1 \ldots \setminus x_{h-1})_{x_h}$ $(h \leq j)$ et pour $(\mathcal{F}' \setminus x_1 \ldots \setminus x_{j-1})_{x_j}$, ainsi que pour la formule $\mathcal{F}' \setminus x_1 \ldots \setminus x_j$, ce qui implique que la formule $\mathcal{F}'$ est bien élément de la classe Γ_k. □

Proposition 28 *Si $\mathcal{F}$ est une formule de la classe Γ_i, alors pour tout ensemble de clauses unitaires $\mathcal{U}$, on a $Noyau(\mathcal{F} \cup \mathcal{U}) \in \Gamma_i$*

Preuve : Conséquence immédiate de la Prop. 27. □

Proposition 29 *Pour tout k, soit $\mathcal{F} \in \Gamma_k$ et $\mathcal{U}$ un ensemble de clauses unitaires, on peut tester en temps $O(N^{k+1})$ si $\mathcal{F} \cup \mathcal{U}$ est satisfaisable (où N représente la longueur de $\mathcal{F} \cup \mathcal{U}$).*

Preuve : Kleine-Büning a prouvé [35] que la k-résolution était complète pour les formules de Γ_{k-1}. Ce qui donne un algorithme $O(N^{k+1})$ pour tester la satisfaisabilité de toute formule de Γ_k.

Une formule est satisfaisable ssi $Unit$ est cohérent et $Noyau$ est satisfaisable (Prop. 1). Pour la formule $\mathcal{F} \cup \mathcal{U}$ il est possible de calculer $Noyau(\mathcal{F} \cup \mathcal{U})$ et l'ensemble $Unit(\mathcal{F} \cup \mathcal{U})$ en temps linéaire. Comme $Noyau(\mathcal{F} \cup \mathcal{U})$ est Γ_k (Prop. 28), on peut tester sa satisfaisabilité en temps $O(N^{k+1})$. Ceci implique que l'on peut tester la satisfaisabilité de $\mathcal{F} \cup \mathcal{U}$ en temps $O(N^{k+1})$. □

La proposition 2 implique que l'on peut générer avec un délai polynômial pour un k fixé, toutes les solutions d'une formule appartenant à la classe Γ_k.

Malheureusement Eiter et al. [23] ont prouvé que tester si une formule quelconque pouvait être renommée en une formule Γ_k était NP-complet pour tout $k \geq 1$. Mais si on sait qu'une formule est Γ_k renommable (par construction par exemple) et que l'on connaît le renommage, alors générer toutes ses solutions revient à générer toutes les solutions d'une formule Γ_k, et peut donc être fait avec un délai polynômial.

4.3 Hiérarchies de Pretolani

Pretolani [40] a généralisé les travaux de Gallo et Scutella [27]. Il améliore le schéma de décomposition de Gallo et Scutella, il étudie son application à une classe de base, générique, dont la satisfaisabilité peut être testée en temps polynômial, autre que Horn. Il propose une famille de hiérarchies polynômiales : une hiérarchie polynômiale $\{\mathcal{C}\}$ est une suite de classes imbriquées (i.e. $\mathcal{C}_i \subseteq \mathcal{C}_{i+1}$ pour tout $i > 0$) telle qu'une formule avec n variables appartienne à toute classe $\mathcal{C}_i$ avec $i \geq n-1$. Résoudre la

satisfaisabilité d'une formule de $\mathcal{C}_i$ se ramène à résoudre la satisfaisabilité de $O(n^i)$ formules de la classe de base $\mathcal{C}$. Si on utilise les formules de Horn comme classe de base, on obtient la hiérarchie $\{\Delta\}$ qui est telle que $\Gamma_i \subsetneq \Delta_i$ pour tout i positif.

Nous utilisons les notations présentées à la section précédente. Soient l un littéral et x une variable, on note $\mathcal{F}_l = \{C \setminus \{l\} \mid C \in F, \overline{l} \notin C\}$ et $\mathcal{F} \setminus x = \{C \setminus x, \neg x \mid C \in F\}$.

Soit $\mathcal{C}$ une classe de formules. On définit la hiérarchie polynômiale $\{\mathcal{C}\}$ de la façon suivante.

Définition 13 (hiérarchie polynômiale $\{C\}$) *Soit $\mathcal{C}_0 = \mathcal{C}$. Pour tout $i > 0$, $\mathcal{F} \in \mathcal{C}_i$ si $\mathcal{F} \in \mathcal{C}_{i-1}$ ou s'il existe une variable $x \in V$ (appelée le candidat) telle que une des deux conditions suivantes soit vérifiée.*

- *$\mathcal{F}_x \in \mathcal{C}_{i-1}$ et $\mathcal{F}_{\neg x} \in \mathcal{C}_i$*
- *$\mathcal{F}_{\neg x} \in \mathcal{C}_{i-1}$ et $\mathcal{F}_x \in \mathcal{C}_i$*

P 3 (classe close par fixation) *Une classe $\mathcal{C}$ est dite close par fixation, si pour toute formule $\mathcal{F} \in C$, on a $\mathcal{F}_{\neg x} \in \mathcal{C}$ $\mathcal{F}_x \in \mathcal{C}$ pour tout $x \in V$.*

On voit facilement que si la classe $\mathcal{C}$ est close par fixation, alors toute formule de $\mathcal{C}$ a la propriété P1. La réciproque n'étant pas toujours vraie. Cette propriété est vérifiée par les classes des formules Horn, Horn renommables, binaires, que nous avons déjà vues et aussi par les classes Horn étendues, Horn étendues simples et ordonnées que nous verrons dans les sections et chapitres suivants.

Supposons que la classe $\mathcal{C}$ est close par fixation, et qu'il existe un algorithme de reconnaissance pour les formules de $\mathcal{C}$ qui est de complexité $O(g(N))$. Dans ce cas Pretolani [40] propose un algorithme $O(n^i g(N))$ pour la reconnaissance des formules de la classe $\mathcal{C}_i$. Cet algorithme consiste en n^i tests de reconnaissance pour la classe de base $\mathcal{C}$. Comme sous-produit de cet algorithme de reconnaissance on récupère une séquence de candidats correspondant à la formule. L'algorithme de reconnaissance utilise le théorème suivant.

Théorème 30 *Dans une hiérarchie polynômiale $\{\mathcal{C}_i\}$, si $\mathcal{C}$ est close par fixation, alors il en est de même pour tous les $\mathcal{C}_i$.*

Pretolani propose ensuite un résultat sur le test de satisfaisabilité d'une formule appartenant à une classe $\mathcal{C}_i$ de la hiérarchie polynômiale $\{\mathcal{C}\}$.

Proposition 31 *Si $\mathcal{F} \in \mathcal{C}_i$ et que l'on connaît un ensemble de séquences de candidats pour $\mathcal{F}$, si le test de satisfaisabilité pour la classe $\mathcal{C}$ a une complexité $O(h(N))$, alors on peut tester la satisfaisabilité de $\mathcal{F}$ en temps $O(n^i h(N))$.*

Preuve : La preuve de cette proposition est donnée par Pretolani dans [40], nous en reprenons ici les grandes lignes.

Pour toute formule $\mathcal{G}$ appartenant à la classe $\mathcal{C}$, on note $Resoud_{\mathcal{C}}(\mathcal{G})$ le résultat de l'algorithme qui test de satisfaisabilité de toute formule de la classe $\mathcal{C}$ appliqué à la formule $\mathcal{G}$. L'algorithme PSAT (Fig. 4.1) permet de déterminer pour une formule appartenant à $\mathcal{C}_i$ si elle est satisfaisable. On peut noter que PSAT consiste en n^i appels à $Resoud_{\mathcal{C}}$. Mais le principal coût de cet algorithme provient du calcul de séquences de littéraux candidats. Ce calcul est effectué lors du test de reconnaissance de la classe $\mathcal{C}_i$, donc cet algorithme a bien une complexité $O(n^i h(N))$. □

Algorithme PSAT
Entrée : Un nombre i, une formule $\mathcal{F} \in \mathcal{C}_i$;
Sortie : Vrai si $\mathcal{F}$ est satisfaisable, Faux sinon ;
début
  Si $i = 0$ **alors retourner** $Resoud_{\mathcal{C}}(\mathcal{F})$;
  $h \leftarrow 0$;
  Trouver une séquence de littéraux candidats $(l_1, \ldots, l_k)$ pour $\mathcal{F}$;
  tant que $(h < k)$ **do** ;
    $h++$;
    $l \leftarrow l_h$;
    si PSAT$(i-1, \mathcal{F}_l)$ = Vrai **alors retourner** Vrai ;
    $\mathcal{F} = \mathcal{F}_{\neg l}$;
  fin tant que ;
  retourner PSAT$(i-1, \mathcal{F})$;
fin.

FIG. 4.1 – *Algorithme PSAT*

Si $\mathcal{C}$ est une classe dont l'algorithme de reconnaissance a une complexité en temps qui est $O(g(N))$, et que l'algorithme de test de satisfaisabilité $Resoud_{\mathcal{C}}$ a une complexité $O(h(N))$ et que $\mathcal{C}$ est close par fixation, alors on peut tester en temps $O(n^i(g(N) + h(N)))$ si une formule de la classe $\mathcal{C}_i$ est satisfaisable.

Proposition 32 *Soit $\mathcal{C}$ une classe close par fixation, avec un algorithme de reconnaissance dont la complexité est $O(g(N))$ et un algorithme de test de satisfaisabilité de complexité $O(h(N))$. Pour toute formule $\mathcal{F} \in \mathcal{C}_i$, on peut générer toutes les solutions de $\mathcal{F}$ avec un délai $O(n^{i+1}(g(N) + h(N)))$.*

Preuve : Comme $\mathcal{C}_i$ est clos par fixation, $Noyau(\mathcal{F} \cup \mathcal{U}) \in \mathcal{C}_i$. Donc on possède un algorithme $O(n^i(g(N)+h(N)))$ pour tester si $Noyau(\mathcal{F} \cup \mathcal{U})$

est satisfaisable. La proposition 2 implique le résultat. □

Chapitre 5

Impossibilité de générer à délai polynômial

Sommaire

5.1 Classes trivialement satisfaisables

On peut remarquer que si une formule $\mathcal{F}$ ne contient pas de clause totalement négative (resp. totalement positive), alors elle est trivialement satisfaisable. Elle admet pour modèle l'ensemble $\{x \mid x \in V\}$ (resp. $\{\neg x \mid x \in V\}$).

Malheureusement ces deux classes ne sont formées sur aucune propriété structurelle de la formule, et si $\mathcal{F}$ appartient à une telle classe, il n'y a aucune raison pour que la même formule $\mathcal{F}$ dans laquelle on a donné la valeur vraie à un littéral quelconque l ($\mathcal{F}_l$) en fasse aussi partie. Nous montrons ici que si P≠NP, générer tous les modèles des formules de ces classes ne peut être fait avec un délai polynômial.

Définition 14 ($\mathcal{T}_+, \mathcal{T}_-$) *On appelle classe $\mathcal{T}_+$ la classe des formules dont toutes les clauses contiennent au moins un littéral positif.*

On appelle classe $\mathcal{T}_-$ la classe de formules dont toutes les clauses contiennent au moins un littéral négatif.

Exemple : La formule $\{\{x_1, \neg x_2, x_3\}, \{x_1, x_2, x_3\}, \{x_2, \neg x_3\}\}$ appartient à la classe $\mathcal{T}_+$. La formule $\{\{\neg x_1, x_2, x_3, x_4\}, \{x_2, \neg x_3\}, \{\neg x_1\}\}$ appartient à la classe $\mathcal{T}_-$.

Proposition 33 *S'il existe un algorithme de génération à délai polynômial pour les formules de $\mathcal{T}_-$ (resp. $\mathcal{T}_+$), alors il existe un algorithme de génération à délai polynômial pour toute formule de logique propositionnelle.*

Preuve : Soit $\mathcal{F}$ une formule quelconque avec un ensemble de variables V. Soit $z \notin V$. On construit la formule $\mathcal{G} = \{C \mid C \in \mathcal{F} et neg(C) \neq \emptyset\} \cup \{C' \mid \exists C \in \mathcal{F}, neg(C) = \emptyset et C' = C \cup \{\neg z\}\} \cup \{\{z, \neg x_i\} \mid \forall x_i \in V\}$. Soit M un modèle de $\mathcal{G}$. Si $\neg z \in M$, alors nécessairement M doit contenir les littéraux $\neg x_i$ pour tout i $(1 \leq i \leq n)$ car les clauses $\{z, \neg x_i\}$ appartiennent à $\mathcal{G}$. Comme toutes les clauses de $\mathcal{G}$ contiennent un littéral négatif, l'ensemble $M = \{\neg x_i \mid 1 \leq i \leq n\} \cup \{\neg z\}$ est un modèle de $\mathcal{G}$. Cet ensemble est donc le seul modèle de $\mathcal{G}$ contenant le littéral $\neg z$. Tous les autres modèles de $\mathcal{G}$ contiennent donc z. On peut remarquer que $\mathcal{G}_z = \mathcal{F}$, donc si M est un modèle de $\mathcal{G}$ contenant z, alors $M \setminus \{z\}$ est un modèle de $\mathcal{F}$. Si on peut générer à délai polynômial tous les modèles de $\mathcal{G}$, alors on obtient un moyen de générer à délai polynômial les modèles de $\mathcal{F}$ (puisque le délai entre deux modèles de $\mathcal{F}$ est au plus égal à deux fois le délai entre deux modèles de $\mathcal{G}$, ce qui reste polynômial). Comme $\mathcal{F}$ est une formule quelconque, ceci n'est possible que si P=NP. □

Exemple : Soit $\mathcal{F}_1 = \{ \{x_1, x_2, x_3\}, \{x_1, \neg x_2, \neg x_3\}, \{\neg x_1, \neg x_4, \neg x_5\}, \{x_1, x_2, x_5\}, \{x_2, \neg x_3, \neg x_4\}\}$ on peut construire la formule $\mathcal{G}_1 = \{\{\neg z, x_1, x_2, x_3\}, \{x_1, \neg x_2, \neg x_3\}, \{\neg x_1, \neg x_4, \neg x_5\}, \{\neg z, x_1, x_2, x_5\}, \{x_2, \neg x_3, \neg x_4\}, \{z, \neg x_1\}, \{z, \neg x_2\}, \{z, \neg x_3\}, \{z, \neg x_4\}, \{z, \neg x_5\}\}$. On peut vérifier que pour tout modèle M de $\mathcal{G}_1$, on a soit $\neg z \in M$, dans ce cas, le seul modèle possible est $M = \{\neg x_1, \neg x_2, \neg x_3, \neg x_4, \neg x_5, \neg z\}$, soit $z \in M$ et $M \setminus \{z\}$ est un modèle de $\mathcal{F}_1$.

A moins que P ne soit égal à NP, si une classe $\mathcal{C}$ contient la classe $\mathcal{T}_-$ (resp. $\mathcal{T}_+$), il est impossible de donner un algorithme de génération à délai polynômial pour les formules de $\mathcal{C}$.

5.2 Quad

Dalal [19] a présenté une classe de formules pour lesquelles il propose un algorithme quadratique de résolution du problème Sat, malheureusement, on ne peut pas générer les modèles d'une formule Quad avec un délai polynômial, car Quad contient les classes $\mathcal{T}_+$ et $\mathcal{T}_-$.

Définition 15 (classe Racine) *Une formule $\mathcal{F}$ appartient à la classe* Racine *si l'une des conditions suivantes est satisfaite :*

- *$\mathcal{F}$ contient la clause vide,*
- *$\mathcal{F} \in \mathcal{T}_+$,*

- $\mathcal{F} \in \mathcal{T}_-$,
- *Toute clause de $\mathcal{F}$ est de longueur deux.*

Une clause C' est une sous-clause de la clause C si $C' \subseteq C$. Une sous-clause C' de C est dite maximale, si $\mid C \mid - \mid C' \mid = 1$. Pour toute clause C, la clause $\neg C$ est définie comme l'ensemble $\{\overline{l} \mid l \in C\}$, et la formule $\sim C$ est l'ensemble $\{\{\overline{l}\} \mid l \in C\}$.
Exemple : La clause $\{x_1, x_3\}$ est une sous-clause maximale de la clause $C = \{x_1, \neg x_2, x_3\}$, alors que la clause $\{x_1\}$ n'est qu'une sous-clause (pas maximale) de C. De même, $\neg C$ est la clause $\{\neg x_1, x_2, \neg x_3\}$ et $\sim C$ est la formule $\{\{\neg x_1\}, \{x_2\}, \{\neg x_3\}\}$.

Pour définir les formules Quad, Dalal utilise un ordre total sur l'ensemble de toutes les clauses. Soit $\prec$ un ordre total arbitraire sur un ensemble de littéraux, cet ordre induit un ordre total sur les clauses : $C' \prec C$ ssi $C' \subsetneq C$ ou il existe un littéral l dans $C \setminus C'$ tel que $l \prec q$ pour tout littéral q de $C' \setminus C$ (on remarque que la relation n'est pas réflexive $C \not\prec C$). $\prec$ détermine un ordre total sur les clauses de $\mathcal{F}$. Pour toute clause de $\mathcal{F}$, $\prec$ détermine en outre un ordre total sur ses sous-clauses maximales. Dans la définition présentée ci-dessous, on utilisera les notations première ou suivante dans le contexte de cet ordre sur les clauses et sur les sous-clauses maximales.

Définition 16 (Quad) *Une formule $\mathcal{F}$ appartient à la classe Quad si l'une des conditions suivantes est vérifiée.*

- *$Noyau(\mathcal{F})$ appartient à la classe Racine,*
- *Soit C' la première sous-clause maximale de la première clause C de $Noyau(\mathcal{F})$ pour laquelle $Noyau(\mathcal{F} \cup \{\neg C'\})$ appartient à la classe Racine. Une des deux propriétés suivantes est vérifiée.*
  - *$Noyau(\mathcal{F} \cup \{\neg C'\})$ est satisfaisable,*
  - *La formule $(\mathcal{F} \setminus \{C\}) \cup \{C'\}$ appartient à Quad.*

Dalal [19] a présenté un algorithme quadratique ($O(N^2k)$ où N est la longueur totale de la formule et k la longueur de la plus grande clause) permettant de tester si une formule appartient à la classe Quad. Il propose en outre un algorithme de la même complexité ($O(N^2k)$) pour tester la satisfaisabilité de toute formule Quad.

Malheureusement la classe Quad contient les classes $\mathcal{T}_-$ et $\mathcal{T}_+$, la proposition 33 implique donc que si P est différent de NP, il n'existe aucun algorithme à délai polynômial pouvant générer les modèles de toutes les formules de la classe Quad.

5.3 Hiérarchies $\{\Omega\}$ et $\{\Delta\}$

Dalal et Etherington [20] ont étendu les résultats de Gallo et Scutella [27] (Chap. 4), ils ont présenté deux hiérarchies de formules $\{\Omega\}$ et $\{\Delta\}$ dont la satisfaisabilité peut être testée en temps polynômial. Malheureusement, pour tout $k \geq 0$, $\mathcal{T}_+$ et $\mathcal{T}_-$ sont incluses dans Δ_k et Ω_k, donc il est impossible de générer à délai polynômial les modèles des formules appartenant aux classes Δ_k et Ω_k $(k \geq 0)$.

Dalal et Etherington utilisent pour définir les hiérarchies $\{\Omega\}$ et $\{\Delta\}$ la notion de multi-sous-ensemble $\sqsubseteq$ à la place de la notion classique de sous-ensemble $\subseteq$. On dit que $\{A_1, A_2, \ldots, A_k\} \sqsubseteq \{B_1, B_2, \ldots, B_l\}$ ssi pour tout i $(1 \leq i \leq k)$, il existe j $(1 \leq j \leq l)$ tel que $A_i \subseteq B_j$.

On rappelle que l'on note $Lit(\mathcal{F})$ l'ensemble des littéraux apparaissant dans la formule $\mathcal{F}$.

Définition 17 (hiérarchies $\{\Delta\}$ et $\{\Omega\}$) *Dalal et Etherington ont défini récursivement les classes Δ_k et Ω_k de la façon suivante.*

- *$\mathcal{F} \in \Delta_0$ ssi une des propriétés suivantes est vérifiée :*
  - *la clause vide appartient à $\mathcal{F}$;*
  - *$\mathcal{F} \in \mathcal{T}_+$;*
  - *$\mathcal{F} \in \mathcal{T}_-$;*
  - *Il existe une clause unitaire positive $\{x\} \in \mathcal{F}$ telle que $\mathcal{F}_x \in \Delta_0$.*
- *Pour tout k, $\mathcal{F} \in \Omega_k$ ssi l'une des propriété suivante est vérifiée :*
  - *$\mathcal{F} \in \Delta_k$;*
  - *pour tout littéral $l \in Lit(\mathcal{F})$, on a $Noyau(\mathcal{F} \cup \{l\}) \in \Delta_k$, et $Noyau(\mathcal{F} \cup \{\bar{l}\}) \in \Omega_k$;*
  - *pour tout littéral $l \in Lit(\mathcal{F})$, on a $Noyau(\mathcal{F} \cup \{l\}) \sqsubseteq \mathcal{F}$ et $Noyau(\mathcal{F} \cup \{x\}) \in \Delta_k$*

Δ_0 est l'ensemble des formules pour lesquelles on peut déterminer la satisfaisabilité immédiatement. La classe Ω_k contient les formules pour lesquelles propager la valeur vrai pour n'importe quel littéral produit une formule appartenant soit à Ω_k, soit à Δ_k. Les classes Δ_k contiennent toutes les formules pour lesquelles il existe un littéral dont la propagation de la valeur produit une formule appartenant soit à Ω_{k-1} soit à Δ_k.

On remarque que $\mathcal{T}_+ \subsetneq \Delta_0$ et $\mathcal{T}_- \subsetneq \Delta_0$ comme en plus $\Delta_k \subsetneq \Omega_k \subsetneq \Delta_{k+1}$, on obtient que pour tout k,$\mathcal{T}_+ \cup \mathcal{T}_- \subsetneq \Delta_k$ et $\mathcal{T}_+ \cup \mathcal{T}_- \subsetneq \Omega_k$. La proposition 33 implique donc que si P est différent de NP, il n'existe pas d'algorithme polynômial pour générer tous les modèles des formules appartenant aux classes Δ_k et Ω_k (pour $k \geq 0$).

Chapitre 6

Conclusion

Dans cette partie, on a pu voir que pour la quasi totalité des classes polynômiales, on peut trouver un algorithme à délai polynômial pour générer toutes les solutions.

Si on connaît une méthode pour tester la satisfaisabilité des formules d'une classe, alors cette méthode utilise une propriété structurelle de la classe. On s'appuie sur cette structuration de la formule pour pouvoir en générer tous les modèles.

Les classes polynômiales pour lesquelles nous ne pouvons pas générer les modèles à délai polynômial, sont celles contenant les classes $\mathcal{T}_+$ (les formules ayant au moins un littéral positif dans chaque clause) ou/et $\mathcal{T}_-$ (les formules ayant au moins un littéral négatif dans chaque clause). A moins que NP ne soit égal à P, il n'existe pas d'algorithme de génération à délai polynômial pour ces formules.

Classe	Reconnaissance	Satisfaisabilité	Génération
Horn	$O(N)$	$O(N)$	$O(nN)$
Horn renommable	$O(N)$	$O(N)$	$O(nN)$
Binaires	$O(N)$	$O(N)$	$O(nN)$
Équilibrées	polynômial	$O(N)$	$O(nN)$
Γ_k	$O(n^k N)$	$O(n^k N)$	$O(n^{k+1} N)$
Γ_k-renommable	NP-complet	$O(n^k N)$	$O(n^{k+1} N)$
Quad	$O(N^2)$	$O(N^2)$	impossible si P$\neq$NP
Ω_k et Δ_k	$O(n^{k+1})$	$O(n^{k+1})$	impossible si P$\neq$NP
Presque Horn(*)	$O(nN)$	$O(1)$	$O(nN)$
q-Horn	$O(N)$	$O(N)$	$O(nN)$

(*) sans clause unitaire

FIG. 6.1 – *Génération à délai polynômial: tableau provisoire*

Dans les parties suivantes, nous étudions d'autres classes de formules pour lesquelles on peut générer toutes les solutions avec un délai po-

lyômial. Un tableau complet est présenté dans la conclusion générale de cette thèse.

Deuxième partie

Formules Horn étendues

Chapitre 1

Présentation

Nous avons vu dans le Chapitre 2 de la Partie I que si une classe de formules vérifie les deux propriétés P1 et P2, alors on peut générer toutes les solutions des formules de cette classe avec un délai $O(nN)$ (où n est le nombre de variables et N la longueur totale de la formule) avec comme unique outil la résolution unitaire. La première de ces propriétés repose sur une stabilité de la classe par la résolution unitaire, la seconde requiert que toute formule de la classe ne comportant pas de clause unitaire soit satisfaisable.

Nous étudions dans cette partie un ensemble de classes de formules, qui vérifient encore ces deux propriétés. La première de ces classes, la classe des formules Horn étendues a été introduite par Chandru et Hooker [12]. Nous présentons dans le Chapitre 2 la définition de cette classe, nous montrons comment on peut tester en temps linéaire la satisfaisabilité d'une telle formule, et nous montrons qu'on peut générer à délai polynômial les modèles de telles formules. Nous présentons aussi l'origine de ces formules. Elles sont issues des résultats de Chandrasekaran [10] sur la programmation linéaire ; nous expliquons ici comment ce résultat s'applique aux formules Horn étendues.

Malheureusement, il n'existe pas actuellement d'algorithme polynômial pour tester si une formule est Horn étendue. Swaminathan et Wagner [46] ont donc présenté une sous-classe des formules Horn étendues qu'ils ont appelée classe Horn étendue simple. Ils ont proposé un algorithme de reconnaissance quadratique pour les formules de cette classe. Schlipf et al. [44] ont remarqué que si on relâchait une des contraintes de la définition des formules Horn étendues (resp. Horn étendues simples), on obtenait une classe plus grande qui avait les mêmes propriétés. Nous avons appelé cette classe les formules Horn élargies (resp. Horn élargies simples). Nous présentons dans le Chapitre 3, une étude de la structure des formules Horn élargies simples qui nous conduit à un algorithme linéaire de reconnaissance de ces formules. Nous présentons en outre un algorithme linéaire permettant de tester si une formule est Horn étendue simple.

Chapitre 2

Les formules Horn étendues

Sommaire

2.1 Présentation

Chandru et Hooker[12] ont présenté la classe des formules Horn étendues, il s'agit d'une généralisation de la classe des formules de Horn pour laquelle le problème SAT est aussi facile à traiter que pour Horn, en utilisant comme pour Horn la résolution unitaire. Nous allons voir que cette propriété permet aussi de générer toutes les solutions pour de telles formules avec un délai $O(nN)$ (où N est la longueur totale de la formule et n le nombre de ses variables).

Une formule est Horn étendue si ses variables correspondent aux arcs d'une arborescence enracinée (i.e. un arbre orienté dans lequel tous les arcs sont orientés en partant de la racine), de telle façon que pour chaque clause les variables apparaissant positivement étiquettent un chemin et les variables apparaissant négativement étiquettent un ensemble de chemins commençant à la racine plus, le cas échéant, un chemin commençant au même sommet que le chemin positif. Les formules de Horn sont celles

pour lesquelles l'arbre associé est une étoile (i.e. l'arborescence enracinée dont tous les arcs partent de la racine).

La découverte de ces formules découle du théorème de Chandrasekaran. Ce théorème caractérise les ensembles d'inéquations linéaires pour lesquelles un solution 0-1 peut toujours être trouvée (s'il en existe une) en arrondissant une solution réelle, qui elle peut être calculée par programmation linéaire. Ils ont prouvé qu'un ensemble d'inégalités avec des coefficients dans 0, 1, -1 correspondant à une arborescence telle que décrite ci-dessus satisfait les conditions du théorème de Chandrasekaran.

2.2 Définitions

Nous rappelons les notations présentées dans la Partie I et présentons ensuite la définition des formules Horn étendues.

Une *arborescence* est un graphe orienté T, dont le graphe sous-jacent est un arbre, et qui a exactement un noeud de degré entrant égal à zéro. Cet sommet est appelé la *racine* de T.

On rappelle que dans toute cette partie, F désigne une formule et V l'ensemble de ses variables. Soit $U \subseteq V$ et T une arborescence dont les arcs sont étiquetés (de manière unique) par les éléments de U. Soit $x \in U$, on notera $\overrightarrow{x}$ l'arc de T étiqueté par x. On note T_W le sous-graphe orienté de T formé par les arcs de T étiquetés par les éléments de W.

On rappelle que nous travaillons en utilisant la forme normale conjonctive ; une formule est donc vue comme un ensemble de clauses, et une clause comme un ensemble de littéraux. Si C est une clause, $pos(C) = \{x \in V \mid x \in C\}$, et $neg(C) = \{x \in V \mid \neg x \in C\}$, si x est une variable on a aussi $CNeg(x) = \{C \in F / \neg x \in C\}$ et $CPos(x) = \{C \in F / x \in C\}$.

Définition 18 *Soit T une arborescence avec une racine r et des arcs étiquetés par les variables de V (de manière unique). Soit $C \in F$ une clause telle que $T_{pos(C)}$ est un chemin orienté (qui peut être vide).*

La clause C est Horn étendue *par rapport à T, si $neg(C) = N_1 \cup \ldots \cup N_k$ avec $N_i \cap N_j = \emptyset$ $(1 \leq i < j \leq k)$, T_{N_i} est un chemin orienté qui commence à la racine r (pour $1 \leq i \leq k-1$), et T_{N_k} est un chemin orienté (qui peut être vide) qui commence au même sommet que $T_{pos(C)}$.*

F est *Horn étendue* par rapport à T, si chaque clause $C \in F$ est Horn étendue par rapport à T. Une formule est Horn étendue si elle est Horn étendue par rapport à une arborescence.

Exemple : Soit F_1 la formule telle que $F_1 = \{C_1, C_2, C_3, C_4\}$. Où $C_1 = \{\neg x_1, \neg x_2, x_4, x_5, \neg x_7\}$, $C_2 = \{\neg x_1, \neg x_3, \neg x_4, x_5\}$, $C_3 = \{x_1, x_2, \neg x_3, \neg x_4, \neg x_5\}$ et $C_4 = \{\neg x_1, \neg x_3, \neg x_6\}$. On peut voir (Fig. 2.1) que la clause C_1 est Horn étendue par rapport à T. Le chemin comportant l'arc $\overrightarrow{x_7}$ commence au même sommet que le chemin correspondant aux variables x_4 et x_5, on

peut en outre voir que le chemin $\overrightarrow{x_1}$,$\overrightarrow{x_2}$ commence à la racine de l'arborescence. Le lecteur pourra vérifier sur la Figure 2.2 que toutes les clauses de F_1 sont Horn étendues par rapport à T. $\mathcal{F}_1$ est donc Horn étendue par rapport à T. On peut donc dire que F_1 est Horn étendue.

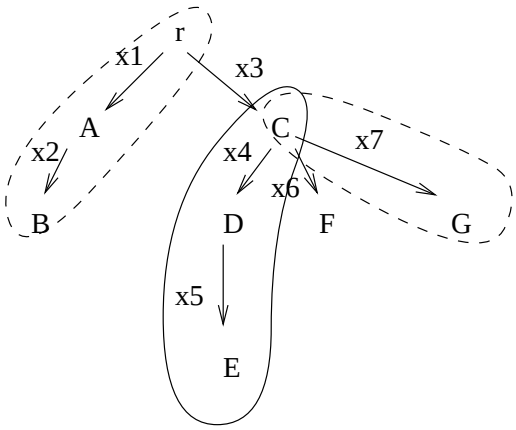

FIG. 2.1 – *La clause C_1 est Horn étendue par rapport à T*

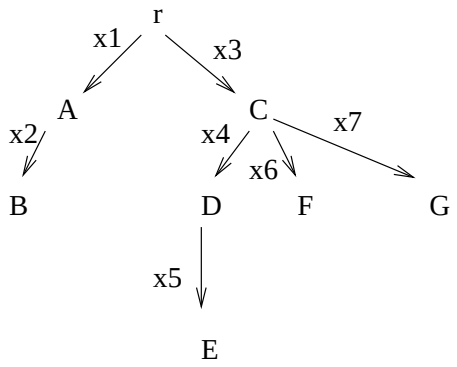

FIG. 2.2 – *Arborescence T dont les arcs sont étiquetés par les variables de F_1*

2.3 Satisfaisabilité et génération à délai polynômial

Chandru et Hooker [12] ont remarqué que la résolution unitaire, permet comme pour Horn, de tester si les formules Horn étendues sont satisfaisables. Nous expliquons ici comment ce processus fonctionne. Nous prouvons en outre que l'on peut générer toutes les solutions des formules Horn étendues avec un délai $O(nN)$.

On rappelle le principe de la résolution unitaire, on dit qu'une clause C est *dérivable* de $\mathcal{F}$ par *résolution unitaire* s'il existe une suite de clauses $C_1, \ldots, C_p$ telle que $C_p = C$ et pour tout i $(1 \leq i \leq p)$; ou bien $C_i \in \mathcal{F}$ ou bien il existe $j, k < i$ et un littéral l vérifiant $C_j = C_i \cup \{\overline{l}\}$ et $C_k = \{l\}$.

Soit $Unit(\mathcal{F})$ l'ensemble des clauses unitaires dérivables de $\mathcal{F}$ par résolution unitaire. On sait que si $Unit(\mathcal{F})$ n'est pas cohérent alors $\mathcal{F}$ est non satisfaisable. La réciproque est fausse dans le cas général.

Soit $Noyau(\mathcal{F})$ l'ensemble des clauses obtenues à partir de $\mathcal{F}$ en ôtant les clauses qui contiennent un élément de $Unit(\mathcal{F})$, et en supprimant dans les clauses restantes tous les complémentaires des littéraux de $Unit(\mathcal{F})$.
Exemple : Soit $F_2 = \{ \{\neg x_1, \neg x_3, \neg x_4, x_6, x_8\}, \{\neg x_1, x_3, x_5, \neg x_6\}, \{x_1, \neg x_2, \neg x_3, \neg x_4, x_7\}, \{x_2, x_6\}, \{\neg x_6\}\}$. On a $Unit(F_2) = \{x_2, \neg x_6\}$ et $Noyau(F_2) = \{\neg x_1, \neg x_3, \neg x_4, x_8\}, \{x_1 \neg x_3, \neg x_4, x_7\}$.

Proposition 34 *Si F est une formule Horn étendue, alors $Noyau(F)$ est aussi une formule Horn étendue.*

Preuve : Soit T une arborescence telle que F est Horn étendue par rapport à T. Soit V l'ensemble des variables de F et V_N l'ensemble des variables de $Noyau(F)$. Soit T_N l'arborescence étiquetée par les variables de V_N obtenue en supprimant de T tous les arcs étiquetés par les variables de $V \setminus V_N$ et en fusionnant l'origine et l'extrémité de tels arcs. Soit X un ensemble de variables, si les variables de X étiquettent un chemin dans T alors les variables de $X \cap V_N$ étiquettent un chemin dans T_N. Soit C_N une clause de $Noyau(F)$, il existe une clause $C \in F$ telle que $C_N = C \cap V_N$. Comme C est Horn étendue par rapport à T, on peut déduire que C_N est Horn étendue par rapport à T_N. Donc $Noyau(F)$ est Horn étendue. □

Exemple : Comme F_2 est Horn étendue par rapport à l'arborescence présentée Figure 2.3, on en déduit que $Noyau(F_2)$ est Horn étendue par rapport à l'arborescence de la Figure 2.4.

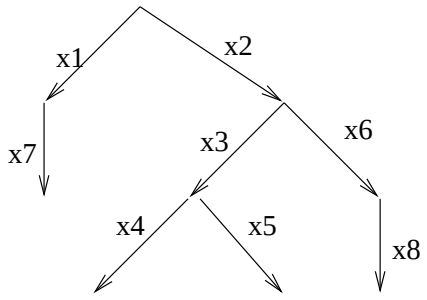

FIG. 2.3 – *F_2 est Horn étendue par rapport à T*

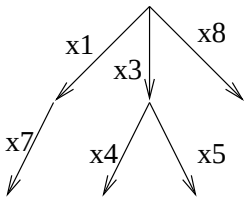

FIG. 2.4 – *$Noyau(F_2)$ est Horn étendue par rapport à T'*

Proposition 35 *Soit F une formule Horn étendue, si F ne contient pas de clause unitaire, alors F est satisfaisable.*

Preuve : Soit T une arborescence telle que F soit Horn étendue par rapport à T. Soit $profondeur_T(x)$ une fonction qui indique pour chaque variable x à quelle distance l'arc $\overrightarrow{x}$ est de la racine (on pose $profondeur_T(x) = 0$ pour toutes les variables telles que $\overrightarrow{x}$ est un arc sortant de la racine). Soit $M = \{x \mid profondeur_T(x)\, modulo\, 2 = 1\} \cup \{\neg x \mid profondeur_T(x)\, modulo\, 2 = 0\}$, On va prouver que M est un modèle pour F. Soit C une clause de F, on va prouver que $C \cap M \neq \emptyset$. Par hypothèse, C contient au moins deux littéraux. Comme F est Horn étendue par rapport à T on a : $neg(C) = N_1 \cup \ldots \cup N_k$ avec $N_i \cap N_j = \emptyset$ $(1 \leq i < j \leq k)$, T_{N_i} est un chemin orienté qui commence à la racine r (pour $1 \leq i \leq k-1$), et T_{N_k} est un chemin orienté (qui peut être vide) qui commence au même sommet que $T_{pos(C)}$.

- $neg(C) = \emptyset$. On sait que $T_{pos(C)}$ est un chemin dans T donc deux variables de $pos(C)$ apparaissant à deux niveaux successifs de T donc une au moins de ces variables est valuée positivement, donc C est satisfaite par M.

- $neg(C) \neq \emptyset$ and $N_k \neq \emptyset$.

  Si $pos(C) \neq \emptyset$, ceci implique qu'il existe un sommet s de T et deux variables x et y telles que $x \in pos(C)$, $y \in neg(C)$ et $\overrightarrow{x}$ a pour origine s tout comme $\overrightarrow{y}$. D'où $profondeur_T(x) = profondeur_T(y)$, donc un des deux littéraux est vrai. La clause C est donc satisfaite par M.

  Si $pos(C) = \emptyset$ alors soit $card(N_k) \geq 2$ dans ce cas T_{N_k} est un chemin dont deux arcs sont à des niveaux différents et donc l'un au moins est de profondeur paire, soit $k > 1$ et on est dans le cas suivant.

- $neg(C) \neq \emptyset$ and $k > 1$, ceci implique que T_{N_1} est un chemin qui commence à la racine de T, soit $\overrightarrow{x}$ le premier arc de T_{N_1}, $profondeur_T(x) = 0$, donc $\neg x \in M$. La clause C est donc satisfaite par M.

□

Exemple : $\{\neg x_1, \neg x_3, x_4, x_5, x_7, \neg x_8\}$ est donc un modèle de $Noyau(F_2)$.

Ces deux propositions conduisent naturellement à un algorithme pour tester la satisfaisabilité des formules Horn étendues. Il suffit de calculer $Unit(F)$, si cet ensemble est cohérent, alors F est satisfaisable puisque l'on sait que $Noyau(F)$ est Horn étendu et qu'il ne contient pas de clause unitaire.
Exemple : $\{\neg x_1, x_2, \neg x_3, x_4, x_5, \neg x_6, x_7, \neg x_8\}$ est un modèle de F_2.

Proposition 36 *Si F est Horn étendue, alors pour tout ensemble de clauses unitaires $\mathcal{U}$, $F \cup \mathcal{U}$ est une formule Horn étendue.*

Preuve : Soit T une arborescence reconnaissant F, toute clause unitaire C est Horn étendue par rapport à T. Si C est une clause unitaire positive, $C = \{x\}$, alors l'arc $\overrightarrow{x}$ forme obligatoirement un chemin dans T. Si $C = \{\neg x\}$ est une clause unitaire négative, alors $\overrightarrow{x}$ forme un chemin dans T. L'origine de $\overrightarrow{x}$ est le sommet d'origine du chemin (vide) correspondant aux littéraux positifs de C. □

On a prouvé (Corollaire. 4, Chap. 2, Part. I) que si une classe de formules vérifie les propriétés suivantes, alors on peut générer tous les modèles de toute formule de la classe avec un délai $O(nN)$ en n'utilisant que la résolution unitaire.

Propriété 1 (P1) *$\mathcal{C}$ vérifie P1, si pour toute formule $\mathcal{F} \in \mathcal{C}$, pour tout ensemble de clauses unitaires $\mathcal{U}$, on a $Noyau(\mathcal{F} \cup \mathcal{U}) \in \mathcal{C}$.*

Propriété 2 (P2) *$\mathcal{C}$ vérifie P2, lorsque pour toute formule $\mathcal{F} \in \mathcal{C}$, le fait que toute clause de $\mathcal{F}$ soit de longueur supérieure ou égale à deux, implique que $\mathcal{F}$ soit satisfaisable.*

On déduit des Prop. 34, 35 et 36 et du Corollaire 4 que l'on peut générer avec un délai $O(nN)$ tous les modèles d'une formule Horn étendue.

2.4 Origine

2.4.1 Préliminaires

La définition des formules Horn étendues [12] est motivée par des résultats en programmation linéaire appliqués à une modélisation possible du problème SAT. Chandru et Hooker ont créé les formules Horn étendues en recherchant une famille de formules dont la matrice associée vérifie toujours les propriétés du théorème de Chandrasekaran.

On rappelle que l'on travaille en Forme Normale Conjonctive et qu'une formule est donc vue comme un ensemble de clauses, qui sont elles mêmes des ensembles de littéraux.

Une clause, par exemple $\{x_1, \neg x_2\}$, peut être représentée par une inéquation sur des variables binaires, dans notre cas $x_1 + (1 - x_2) \geq 1$, où x_1 et x_2 doivent prendre la valeur 0 ou 1. On dira que x_j est vrai si $x_j = 1$, et faux si $x_j = 0$. L'inégalité ci-dessus peut s'écrire $x_1 - x_2 \geq 0$, plus généralement, on peut mettre ces inégalités sous la forme $ax \geq a_0$, où a est un vecteur ligne dont les composantes sont dans $\{0, 1, -1\}$, x est un vecteur colonne $(x_1, x_2, \ldots, x_n)$, et a_0 est égal à 1 moins le nombre de -1 dans a. Donc une formule contenant m clauses peut être représentée comme un système d'inéquations $\mathcal{H}x \geq b$, dans lequel $\mathcal{H}$ est une matrice $m \times n$, x est binaire. Il est clair que la formule est satisfaisable si et seulement si le système suivant a une solution :

$$\begin{aligned} \mathcal{H}x &\geq b \\ -x &\geq -e \\ x &\geq 0 \end{aligned} \tag{2.1}$$

Où e est un vecteur de n uns et x est un vecteur colonne d'entiers. La relaxation linéaire de (2.1) est obtenue en retirant la contrainte portant sur le caractère entier de x. La programmation linéaire trouve une solution de la relaxation linéaire si une telle solution existe.

2.4.2 Théorème de Chandrasekaran

Dans le cas général, un système d'équations du type (2.1) n'a pas forcément de solution entière, Chandrasekaran [10] a exposé une condition suffisante pour qu'un système d'inéquations ait une solution entière. C'est cette condition qui a été utilisée par Chandru et Hooker [12] pour créer les formules Horn étendues.

Soit $\lceil \alpha \rceil$ le plus petit entier supérieur ou égal à α, et pour un vecteur x, la $i$ème composante du vecteur $\lceil x \rceil$ est égale à $\lceil x_i \rceil$.

Théorème 37 (Chandrasekaran[10]) *Considérons le système linéaire $\mathcal{A}x \geq b, x \geq 0$, dans lequel $\mathcal{A}$ est une matrice entière $m \times n$ et b un vecteur entier. Soit $\mathcal{T}$ une matrice carrée $n \times n$ non singulière qui vérifie les conditions suivantes :*

1. *$\mathcal{T}$ et $\mathcal{T}^{-1}$ sont entières ;*
2. *Chaque ligne de $\mathcal{T}^{-1}$ contient au plus une entrée négative, et toutes ces entrées sont des -1 ;*
3. *Chaque ligne de $\mathcal{A}\mathcal{T}^{-1}$ contient au plus une entrée négative, et toutes ces entrées sont des -1.*

Alors si x est une solution du système linéaire, il en sera de même pour le vecteur $\mathcal{T}^{-1}\lceil \mathcal{T}x \rceil$.

2.4.3 Motivations des formules Horn étendues

On rappelle qu'une formule est satisfaisable si et seulement si un système d'inégalités de la forme (2.1) a une solution. Chandru et Hooker [12] utilisent le théorème 37 pour identifier les conditions pour lesquelles (2.1) a une solution si sa relaxation linéaire en possède une. Nous allons ici montrer que toute formule Horn étendue vérifie les conditions du théorème de Chandrasekaran.

Proposition 38 *Soit F une formule Horn étendue par rapport à un arbre T. Soit $\mathcal{H}$ la matrice correspondant à la formule F, et $\mathcal{A}$, la matrice telle que :*

$$\mathcal{A} = \begin{pmatrix} \mathcal{H} \\ -\mathcal{I} \end{pmatrix}$$

Où $\mathcal{I}$ est la matrice identité.

Si F est Horn étendue, alors $\mathcal{A}$ vérifie les propriétés du théorème de Chandrasekaran.

Preuve : On va prouver qu'il existe une matrice de transition $\mathcal{T}$ telle que $\mathcal{T}$ et $\mathcal{T}^{-1}$ sont entières, chaque ligne de $\mathcal{T}^{-1}$ et de $\mathcal{A}\mathcal{T}^{-1}$ contient au plus une entrée négative, qui de plus ne peut être différente de -1.

On remarque d'abord que

$$\mathcal{A}\mathcal{T}^{-1} = \begin{pmatrix} \mathcal{H}\mathcal{T}^{-1} \\ -\mathcal{T}^{-1} \end{pmatrix}$$

On doit donc prouver que chaque ligne de $\mathcal{T}^{-1}$ contient au plus une entrée positive (égale à 1) et une entrée négative (égale à -1), et que chaque ligne de $\mathcal{H}\mathcal{T}^{-1}$ contient au plus une entrée négative (qui doit être égale à -1).

Soit $\mathcal{T}^{-1}$ la matrice telle que ses colonnes sont indexées par les sommets de l'arborescence (sauf la racine) et ses lignes sont indexées par les arcs de l'arborescence (i.e. par les variables de la formule). $\mathcal{T}^{-1}_{sx} = -1$ si l'arc $\overrightarrow{x}$ a pour extrémité le sommet s. $\mathcal{T}^{-1}_{sx} = 1$ si l'arc $\overrightarrow{x}$ a pour origine le sommet s. $\mathcal{T}^{-1}_{sx} = 0$ dans tous les autres cas. Chaque arc n'ayant qu'une origine et une extrémité, il est facile de voir, que $\mathcal{T}^{-1}$ vérifie les conditions.

Les éléments de la matrice $\mathcal{H}\mathcal{T}^{-1}$, dont les lignes sont indexées par les clauses de F et les colonnes par les sommets de l'arborescence T, vérifient la propriété suivante pour tout sommet s de T et toute clause $C \in F$:

$$\begin{aligned} \mathcal{H}\mathcal{T}^{-1}_{C,s} = & \, card(\{x \in neg(C) \mid \overrightarrow{x} \text{ entrant en } s\}) \\ & -card(\{x \in neg(C) \mid \overrightarrow{x} \text{ sortant de } s\}) \\ & +card(\{x \in pos(C) \mid \overrightarrow{x} \text{ sortant de } s\}) \\ & -card(\{x \in pos(C) \mid \overrightarrow{x} \text{ entrant en } s\}) \end{aligned} \tag{2.2}$$

Pour chaque clause $C \in F$, il y a donc au plus un sommet pour lequel cette somme est négative (c'est le dernier sommet de $T_{pos(C)}$), mais il ne peut pas être inférieur à -1. Donc $\mathcal{H}\mathcal{T}^{-1}$ vérifie bien la propriété : au maximum une entrée négative par ligne et cette entrée doit être égale à -1.

$\mathcal{T}^{-1}$ est une matrice triangulaire (si on ordonne les sommets et les arcs selon un parcours en profondeur de l'arborescence), dont tous les éléments sur la diagonale sont égaux à -1. Son déterminant est donc ± 1. $\mathcal{T}$ est donc une matrice entière.

La matrice $\mathcal{A}$ vérifie donc bien les conditions du théorème de Chandrasekaran. □

Corollaire 39 *Si F est Horn étendue alors on peut tester en temps polynômial si F est satisfaisable.*

2.4.4 Exemple

Exemple : Pour la formule $F_1 = \{C_1, C_2, C_3, C_4\}$ (où $C_1 = \{\neg x_1, \neg x_2, x_4, x_5, \neg x_7\}$, $C_2 = \{\neg x_1, \neg x_3, \neg x_4, x_5\}$, $C_3 = \{x_1, x_2, \neg x_3, \neg x_4, \neg x_5\}$ et $C_4 = \{\neg x_1, \neg x_3, \neg x_6\}$) définie dans la section 2.2, on a les matrices suivantes :

La matrice $\mathcal{H}$ représente la formule elle même, chaque ligne représente une clause et chaque colonne représente une variable, un 1 indique que la variable apparaît dans $pos(C)$ un -1 indique que la variable apparaît dans $neg(C)$, un 0 signifie que la variable n'apparaît pas dans C.

$$\mathcal{H} = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 \end{array} \\ \begin{pmatrix} -1 & -1 & 0 & 1 & 1 & 0 & -1 \\ -1 & 0 & -1 & -1 & 1 & 0 & 0 \\ 1 & 1 & -1 & -1 & -1 & 0 & 0 \\ -1 & 0 & -1 & 0 & 0 & -1 & 0 \end{pmatrix} \end{array}$$

La matrice $\mathcal{T}^{-1}$ représente l'arbre T (Figure. 2.2). Les colonnes de $\mathcal{T}^{-1}$ sont les sommets de T, les lignes de $\mathcal{T}^{-1}$ sont les arcs de T. On affecte 1 lorsque l'arc a pour origine le sommet, -1 si le sommet est l'extrémité de l'arc et 0 si l'arc ne touche pas le sommet.

$$\mathcal{T}^{-1} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{pmatrix} \end{array}$$

La matrice $\mathcal{HT}^{-1}$ est la multiplication des deux précédentes matrices.

$$\mathcal{HT}^{-1} = \begin{array}{c} \\ C_1 \\ C_2 \\ C_3 \\ C_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & -1 & 0 & 1 \\ 1 & 0 & 0 & 2 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & -1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

2.5 Conclusion

Les formules Horn étendues sont intéressantes, on peut tester leur satisfaisabilité en temps linéaire et générer leurs solutions avec un délai $O(nN)$ en n'utilisant qu'un outil très simple, la résolution unitaire. Il n'y a malheureusement pas d'algorithme connu pour la reconnaissance des formules Horn étendues. Au chapitre suivant nous présentons la classe des formules Horn étendues simples introduite par Swaminathan et Wagner [46] pour laquelle nous proposons un algorithme de reconnaissance en temps linéaire, cette classe de formules est une restriction de la classe Horn étendue, qui garde les propriétés importantes de cette classe. Le test de satisfaisabilité et la génération à délai polynômial ne nécessitent pas d'autre outil que la résolution unitaire.

Chapitre 3

Les formules Horn étendues et élargies simples

Sommaire

3.1 Introduction

Comme on ne sait pas encore reconnaître efficacement les formules Horn étendues, Swaminathan et Wagner [46] ont introduit une restriction qu'ils ont appelée classe des formules Horn étendues simples pour laquelle ils ont proposé un algorithme quadratique de reconnaissance. Nous étudions ici cette classe ainsi qu'une extension proposée par Schlipf et al [44] que nous avons nommée classe des formules de Horn élargies simples. Nous proposons des algorithmes linéaires pour reconnaître les formules Horn étendues simples et Horn élargies simples. Les résultats présentés dans ce chapitre sont tirés d'un article de Benoist et Hébrard [4].

Dans ce chapitre, nous allons étudier les formules Horn étendues simples et Horn élargies simples. Dans un premier temps nous présentons les définitions de ces formules. Nous étudions ensuite les propriétés intéressantes de ces formules : le test de satisfaisabilité linéaire n'utilisant que la résolution unitaire, ainsi que la possibilité de générer avec un délai $O(nN)$ (où N représente la longueur totale de la formule et n le nombre de ses variables) toutes les solutions de ces formules. Nous présentons ensuite un algorithme linéaire de reconnaissance des formules Horn élargies simples. Cet algorithme est basé sur l'étude de classes d'équivalences sur les variables, que nous appelons : *les agrégats*. Nous étudions la structure des agrégats des formules Horn élargies simples, ce qui nous permet de proposer un algorithme linéaire de reconnaissance. Nous présentons ensuite une modification de cet algorithme qui permet de reconnaître les formules Horn étendues simples en un temps linéaire. Si de plus on suppose que la formule étudiée ne contient pas de variable monotone positive, alors cet algorithme n'utilise que des structures de données assez simples.

3.2 Définitions

Nous présentons les notations dont on va avoir besoin dans ce chapitre, ainsi que les définitions des formules Horn étendues et Horn élargies ainsi que des formules Horn étendues simples et Horn élargies simples qui vont être étudiées dans ce chapitre.

Une *arborescence* T est un graphe orienté dont le graphe non orienté sous-jacent est un arbre et qui a exactement un noeud dont le degré entrant est zéro : ce noeud est appelé la *racine* de T. Si T n'a qu'un arc dont l'origine est r, on dira que T a un *pied*, et on notera cet arc $foot(T)$.

Soit $U \subseteq V$ et T une arborescence avec des arcs étiquetés (de manière unique) par les éléments de U. Soit $x \in U$, on notera $\overrightarrow{x}$ l'arc de T étiqueté par x. Soit $x, y \in U$, on dit que $\overrightarrow{x}$ est le *parent* de $\overrightarrow{y}$ dans T, s'il existe un sommet v qui est l'extrémité de $\overrightarrow{x}$ et l'origine de $\overrightarrow{y}$. S'il existe un chemin orienté commençant avec $\overrightarrow{x}$ et finissant avec $\overrightarrow{y}$ on dit que $\overrightarrow{x}$ est un *ancêtre* de $\overrightarrow{y}$. Soit $W \subseteq U$. On note T_W le sous-graphe orienté de T formé par les arcs de T étiquetés par les éléments de W.

On rappelle que si C est une clause, $pos(C) = \{x \in V \mid x \in C\}$, et $neg(C) = \{x \in V \mid \neg x \in C\}$, si x est une variable on a aussi $CNeg(x) = \{C \in F / \neg x \in C\}$ et $CPos(x) = \{C \in F / x \in C\}$.

Définition 19 *Soit T une arborescence avec une racine r et des arcs étiquetés par les variables de V (de manière unique). Soit $C \in F$ une clause telle que $T_{pos(C)}$ est un chemin orienté (qui peut être vide).*

1. *La clause C est* Horn élargie simple *par rapport à T si $T_{neg(C)}$ est une arborescence dont la racine est r.*

2. *La clause C est* Horn élargie *par rapport à T si $neg(C) = N_1 \cup N_2$, avec T_{N_1} une arborescence dont la racine est r et T_{N_2} un chemin orienté (qui peut être vide) qui commence au même point que $T_{pos(C)}$.*

3. *La clause C est* Horn étendue simple *par rapport à T si $neg(C) = N_1 \cup \ldots \cup N_k$, avec $N_i \cap N_j = \emptyset$ $(1 \leq i < j \leq k)$, et T_{N_i} est un chemin orienté qui commence à la racine r $(1 \leq i \leq k)$.*

4. *La clause C est* Horn étendue *par rapport à T, si $neg(C) = N_1 \cup \ldots \cup N_k$ avec $N_i \cap N_j = \emptyset$ $(1 \leq i < j \leq k)$, T_{N_i} est un chemin orienté qui commence à la racine r (pour $1 \leq i \leq k-1$), et T_{N_k} est un chemin orienté (qui peut être vide) qui commence au même sommet que $T_{pos(C)}$.*

F est *Horn élargie simple (Horn élargie, Horn étendue simple, Horn étendue)* par rapport à T, si chaque clause $C \in F$ est Horn élargie simple (Horn élargie, Horn étendue simple, Horn étendue) par rapport à T. Une formule est *Horn élargie simple (Horn élargie, Horn étendue simple, Horn étendue)* si elle est Horn élargie simple (Horn élargie, Horn étendue simple, Horn étendue) par rapport à une arborescence. Lorsqu'aucune confusion ne sera possible, nous écrirons T *reconnaît* F à la place de F est Horn élargie simple (Horn élargie, Horn étendue simple, Horn étendue) par rapport à T.
Exemple : La clause $\{\neg x_1, x_3, x_4, \neg x_5, \neg x_6, \neg x_7, \neg x_8\}$ est Horn élargie simple par rapport à l'arborescence T en Fig. 3.1.

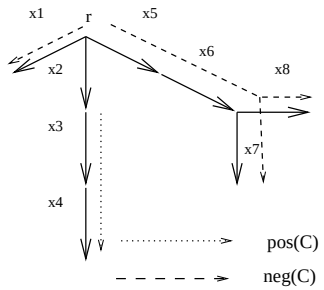

FIG. 3.1 – *Arborescence T*

Exemple : La clause $\{\neg x_1, x_3, x_4, \neg x_5, \neg x_6, \neg x_7\}$ est Horn étendue simple par rapport à l'arborescence T en Fig. 3.2.

On peut remarquer que toute formule Horn étendue est Horn élargie et que de même toute formule Horn étendue simple est Horn élargie simple.

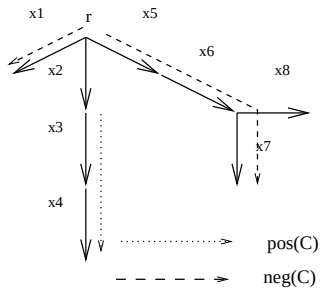

FIG. 3.2 – *Arborescence T*

3.3 Satisfaisabilité et génération à délai polynômial

Comme les formules Horn élargies simples ne sont pas un sous-ensemble des formules Horn étendues, nous devons prouver le même type de résultats que ceux obtenus au chapitre précédent. On va prouver que l'on peut tester en temps linéaire la satisfaisabilité d'une formule Horn élargie simple et qu'il est possible d'en générer toutes les solutions avec un délai $O(nN)$ en n'utilisant que la résolution unitaire.

Proposition 40 *Si F est une formule Horn élargie simple, alors pour tout $\mathcal{U}$ ensemble de clauses unitaires, $Noyau(F \cup \mathcal{U})$ est Horn élargie simple.*

Preuve : Soit T une arborescence reconnaissant F. Soit V l'ensemble des variables de F et V_N l'ensemble des variables de $Noyau(F \cup \mathcal{U})$. Soit T_N l'arborescence étiquetée par les variables de V_N obtenue en supprimant de T tous les arcs étiquetés par les variables de $V \setminus V_N$ et en fusionnant l'origine et l'extrémité de tels arcs. Soit X un ensemble de variables, si les variables de X étiquettent un chemin dans T, alors les variables de $X \cap V_N$ étiquettent un chemin dans T_N. Si les variables de X étiquettent une arborescence dans T, alors les variables de $X \cap V_N$ étiquettent une arborescence dans T_N. Soit C_N une clause de $Noyau(F \cup \mathcal{U})$, il existe une clause $C \in F$ telle que $C_N = C \cap V_N$ (car $\mathcal{U}$ ne contient que des clauses unitaires). Comme C est Horn élargie simple par rapport à T, on peut déduire que C_N est Horn élargie simple par rapport à T_N. Donc $Noyau(F \cup \mathcal{U})$ est Horn étendue simple. □

Proposition 41 *Si F est Horn élargie simple et F ne contient pas de clause unitaire, alors F est satifaisable.*

Preuve : Soit T une arborescence reconnaissant F. Soit $M_1 = \{\neg x \mid \overrightarrow{x}$ a pour origine $r\}$ et $M_2 = \{x \mid x \in V, \neg x \notin M_1\}$. On va prouver que

$M = M_1 \cup M_2$ est un modèle de F. Soit C une clause de F ($\mid C \mid \geq 2$ car F ne contient pas de clause unitaire). C est Horn élargie simple par rapport à T, donc $T_{pos(C)}$ est un chemin et $T_{neg(C)}$ est une arborescence dont la racine est r. Si $\mid pos(C) \mid \geq 2$ alors une au moins des deux variables est valuée positivement (même dans le cas ou le chemin correspondant à $pos(C)$ commence à la racine de T, un au moins de ces arcs n'a pas pour origine r). C est donc satisfaite dans ce cas. Si $\mid pos(C) \mid < 2$, alors $\mid neg(C) \mid \geq 1$ donc il existe $y \in neg(C)$ tel que $\overrightarrow{y}$ a pour origine r, donc C est satisfaite. □

On déduit des Prop. 40, 41 et du Corollaire 4 (Partie I, Chap. 2) que l'on peut générer avec un délai $O(nN)$ tous les modèles d'une formule Horn élargie simple.

La classe Horn étendue simple est incluse dans la classe Horn élargie simple, il existe donc un algorithme linéaire permettant de tester si une telle formule est satisfaisable et il existe aussi un algorithme à délai $O(nN)$ donnant toutes les solutions d'une formule Horn étendue simple.

3.4 Agrégats

Nous présentons dans cette section, la notion d'agrégat. Cette notion, intrinsèque à toute formule, regroupe les variables en classes d'équivalences. Nous étudions les propriétés de ces classes d'équivalences pour les formules Horn élargies simples. Cette étude servira de base à la construction d'un algorithme de reconnaissance des formules Horn élargies simples. On peut remarquer en outre que comme toute formule Horn étendue simple est Horn élargie simple, ces propriétés sont aussi vraies pour les formules Horn étendues simples.

Définition 20 (Agrégat) *Soit R la clôture transitive et réflexive de la relation $\{(x,y) \in V \times V / \exists C \in F$ telle que $x, y \in pos(C)$ et $card(CNeg(x)) = card(CNeg(y))\}$. R est une relation d'équivalence sur les éléments de V, et ses classes d'équivalences sont appelées les* agrégats *de F.*

Exemple : Soit $F_0 = \{C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8\}$ avec
$neg(C_1) = \{x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_{10}, x_{13}\}$, $pos(C_1) = \{x_{11}, x_{12}\}$;
$neg(C_2) = \{x_1, x_5, x_6, x_7, x_8, x_{10}, x_{13}\}$, $pos(C_2) = \{x_4, x_{12}\}$;
$neg(C_3) = \{x_1, x_4, x_5, x_6, x_7, x_8, x_9, x_{11}, x_{12}, x_{13}\}$, $pos(C_3) = \{x_{10}\}$;
$neg(C_4) = \{x_1, x_2, x_3, x_5, x_6, x_7, x_8, x_{13}\}$, $pos(C_4) = \{x_9, x_{12}\}$;
$neg(C_5) = \{x_2, x_3\}$, $pos(C_5) = \{x_1, x_5\}$;
$neg(C_6) = \{x_2, x_3\}$, $pos(C_6) = \{x_1, x_8, x_{12}, x_{13}\}$;
$neg(C_7) = \emptyset$, $pos(C_7) = \{x_1, x_7, x_8, x_{10}\}$;
$neg(C_8) = \emptyset$ $pos(C_8) = \{x_1, x_6\}$.
Pour $i = 1, 2, 3, 5, 6, 7, 8, 13$, $card(CNeg(x_i)) = 4$;
$card(CNeg(x_{10})) = 2$,

Pour $i = 4, 9, 11, 12$, $card(CNeg(x_i)) = 1$;
Les agrégats de F_0 sont : $A_1 = \{x_1, x_5, x_6, x_7, x_8, x_{13}\}$, $A_2 = \{x_{10}\}$, $A_3 = \{x_4, x_9, x_{11}, x_{12}\}$ et $A_4 = \{x_2, x_3\}$.

Les trois propositions suivantes sont des conséquences directes des définitions.

Proposition 42 *Supposons que F soit Horn élargie simple. Soient T une arborescence reconnaissant F et $x, y \in V$. Si $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ dans T, alors $CNeg(y) \subseteq CNeg(x)$.*

Preuve : Soit P le chemin orienté de T qui commence à la racine de T et dont le dernier arc est $\overrightarrow{y}$. Supposons que $CNeg(y) \neq \emptyset$. Soit $C \in CNeg(y)$. Par définition, tous les arcs de P sont étiquetés par des éléments de $neg(C)$, donc $x \in neg(C)$ et $C \in CNeg(x)$. Si $CNeg(y) = \emptyset$, le résultat est trivialement vérifié. □

Notation 1 *On utilisera $CNeg(A)$ pour décrire l'ensemble $\bigcup_{x \in A} CNeg(x)$.*

Proposition 43 *Soit A un agrégat et $x, y \in A$. Si F est Horn élargie simple, alors $CNeg(x) = CNeg(y) = CNeg(A)$.*

Preuve : Soit T une arborescence reconnaissant F. Supposons qu'il existe $C \in F$ telle que $x, y \in pos(C)$. Par définition il existe un chemin orienté de T qui contient $\overrightarrow{x}$ et $\overrightarrow{y}$. On déduit de la Prop. 42 que $CNeg(x) \subseteq CNeg(y)$ ou $CNeg(y) \subseteq CNeg(x)$. D'où $CNeg(x) = CNeg(y)$ car $card(CNeg(x)) = card(CNeg(y))$. Finalement, le résultat est obtenu par transitivité de l'égalité. □

Proposition 44 *Supposons que F est Horn élargie simple. Soit T une arborescence reconnaissant F et A un agrégat.*

1. *Pour toute clause $C \in F$, $T_{A \cap pos(C)}$ est un chemin orienté de T (qui peut être vide).*
2. *T_A est un sous-graphe connexe de T (T_A est une arborescence), et T_A a un pied.*

Preuve :

1. Soient $x, y \in A \cap pos(C)$. On peut supposer que $\overrightarrow{x}$ est un ancêtre de $\overrightarrow{y}$ dans T puisque $T_{pos(C)}$ est un chemin orienté. Soit $\overrightarrow{z}$ un arc appartenant au chemin orienté de T dont le premier arc est $\overrightarrow{x}$ et dont le dernier arc est $\overrightarrow{y}$. Il est suffisant de montrer que $z \in A \cap pos(C)$. On a $z \in pos(C)$ car $T_{pos(C)}$ est un chemin orienté. De plus $CNeg(y) \subseteq CNeg(z) \subseteq CNeg(x)$ (Prop. 42) et $card(CNeg(x)) = card(CNeg(y))$, donc $card(CNeg(z)) = card(CNeg(x))$ et $z \in A$.

2. Par définition d'un agrégat, il existe une permutation $(C_1, \ldots, C_k)$ des éléments de $\{C \in F / A \cap pos(C) \neq \emptyset\}$, telle que pour tout i $(1 < i \leq k)$ il existe j $(1 \leq j < i)$ tel que $pos(C_i) \cap pos(C_j) \neq \emptyset$. Pour tout i $(1 \leq i \leq k)$ $T_{A \cap pos(C_i)}$ est un chemin orienté (Prop. 44 (1)). Le résultat découle de ce que pour chaque $U \subseteq V$ et$W \subseteq V$, si T_U est une arborescence avec un pied, T_W est un chemin orienté, et $U \cap W \neq \emptyset$, alors $T_{U \cup W}$ est une arborescence avec un pied.

□

Supposons F Horn élargie simple, soient A et A' deux agrégats et $C \in F$. On observe que si $A \cap pos(C) \neq \emptyset$, $A' \cap pos(C) \neq \emptyset$ et $card(CNeg(A)) = card(CNeg(A'))$, alors $A = A'$.

Définition 21 (lAgrégats) *Soit $C \in F$. On définit $lAgregats(C)$ comme la liste ordonnée des agrégats $(A_1, \ldots, A_n)$ telle que $A_i \cap pos(C) \neq \emptyset$ $(1 \leq i \leq n)$, $pos(C) \subseteq (A_1 \cup \ldots \cup A_n)$ et $card(CNeg(A_i)) > card(CNeg(A_j))$ $(1 \leq i < j \leq n)$.*

Exemple : Pour la formule F_0, $lAgregats(C_1) = (A_3)$, $lAgregats(C_2) = (A_3)$, $lAgregats(C_3) = (A_2)$, $lAgregats(C_4) = (A_3)$, $lAgregats(C_5) = (A_1)$, $lAgregats(C_6) = (A_1, A_3)$, $lAgregats(C_7) = (A_1, A_2)$, $lAgregats(C_8) = (A_1)$.

Lemme 45 *Supposons que F est Horn élargie simple. Soit T une arborescence reconnaissant F. Soit $C \in F$, et A, A' deux agrégats tels que A est le prédécesseur de A' dans $lAgregats(C)$. Soit $\overrightarrow{x}$ le premier arc du chemin orienté $T_{A' \cap pos(C)}$. Alors le dernier arc de $T_{A \cap pos(C)}$ est le parent de $\overrightarrow{x}$ dans T, et $\overrightarrow{x} = foot(T_{A'})$.*

Preuve : Soit $q \in A \cap pos(C)$. L'arc $\overrightarrow{q}$ est un ancêtre de $\overrightarrow{x}$ puisque $T_{pos(C)}$ est un chemin orienté et $card(CNeg(q)) = card(CNeg(A)) > card(CNeg(A')) = card(CNeg(x))$ (Prop. 42). Soit $\overrightarrow{p}$ le parent de $\overrightarrow{x}$ dans T. On a deux possibilité, $\overrightarrow{q} = \overrightarrow{p}$ ou $\overrightarrow{q}$ est un ancêtre de $\overrightarrow{p}$. On a $p \in pos(C)$ car $T_{pos(C)}$ est un chemin orienté, et $p \notin A'$ car $\overrightarrow{x}$ est le premier arc de $T_{A' \cap pos(C)}$. La Prop. 42 nous donne $card(CNeg(p)) > card(CNeg(x))$. Soit A'' l'agrégat tel que $p \in A''$. On a $card(CNeg(A'')) > card(CNeg(A'))$ et ensuite $card(CNeg(A'')) \geq card(CNeg(A))$ par définition de $lAgregats(C)$. On a $card(CNeg(q)) \geq card(CNeg(p))$ (Prop. 42), et $card(CNeg(A)) \geq card(CNeg(A''))$. Donc $card(CNeg(A)) = card(CNeg(A''))$, $A = A''$ et $p \in A$. Pour tout $q \in A \cap pos(C)$, si $p \neq q$ alors $\overrightarrow{q}$ est un ancêtre de $\overrightarrow{p}$, d'où $\overrightarrow{p}$ est le dernier arc de $T_{A \cap pos(C)}$. Le parent de $\overrightarrow{x}$ n'appartient pas à A', donc $\overrightarrow{x} = foot(T_{A'})$. □

On montre maintenant que l'ensemble des agrégats d'une formule Horn élargie simple a une structure de forêt.

Proposition 46 *Soient A, A' et A'' des agrégats, et C_1, C_2 deux clauses de F tels que A' (resp. A'') est le prédécesseur de A dans $lAgregats(C_1)$ (resp. $lAgregats(C_2)$). Si F est Horn élargie simple alors $A' = A''$.*

Preuve : Soit T une arborescence reconnaissant F. Il existe des variables $x_1, y_1 \in pos(C_1)$ telles que $x_1 \in A'$, $y_1 \in A$, et $\overrightarrow{x_1}$ est le parent de $\overrightarrow{y_1}$ dans T (Lemme 45). Il existe deux autres variables $x_2, y_2 \in pos(C_2)$ telles que $x_2 \in A''$, $y_2 \in A$, et $\overrightarrow{x_2}$ est le parent de $\overrightarrow{y_2}$ dans T (Lemme 45). T_A est un sous graphe connexe de T (Prop. 44), d'où $x_1 = x_2$ et $A' = A''$. □

Notation 2 *Soit A un agrégat. On appelle* PRED(A) *l'ensemble $\{A' / \exists\, C \in F$ tel que A' est le prédécesseur de A dans $lAgregats(C)\}$.*

Remarque, si F est Horn élargie simple, alors pour tout agrégat A, $card(PRED(A)) \leq 1$ (Prop. 46).
Exemple : $PRED(A_1) = \emptyset$, $PRED(A_2) = \{A_1\}$, $PRED(A_3) = \{A_1\}$, $PRED(A_4) = \emptyset$. La forêt associée est dessinée en Fig. 3.3.

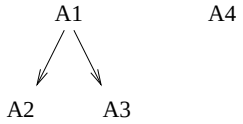

FIG. 3.3 – *Forêt associée aux agrégats de F_0*

Proposition 47 *Supposons que F est Horn élargie simple. Si A et A' sont deux agrégats tels que $PRED(A') = \{A\}$. Alors on a $CNeg(A') \subseteq CNeg(A)$.*

Preuve : Soit T une arborescence reconnaissant F. Soit $C \in F$ une clause telle que $A, A' \in lAgregats(C)$. Il existe deux variables $x, y \in pos(C)$ telles que $x \in A$, $y \in A'$, et $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ dans l'arborescence T (Lemme 45). On a $CNeg(y) \subseteq CNeg(x)$ (Prop. 42), $CNeg(A) = CNeg(x)$ et $CNeg(A') = CNeg(y)$ (Prop. 43), d'où $CNeg(A') \subseteq CNeg(A)$. □

Soit $F = \{C_1, \ldots, C_m\}$ et $N = card(C_1) + \ldots + card(C_m)$.

Proposition 48 *Les agrégats de F peuvent être calculés en temps $O(N)$.*

Preuve : Définissons dans un premier temps la procédure OrderPos (Fig. 3.4) qui calcule pour toute clause C, une liste ordonnée des élément de $pos(C)$, notée $ordPos(C)$, telle que pour tout couple de variables $x, y \in pos(C)$, x est placé avant y dans $ordPos(C)$ si $card(CNeg(x)) \geq card(CNeg(y))$.

Chaque $W[i]$ $(1 \leq i \leq card(F))$ représente l'ensemble $\{x \in V/\ card(CNeg(x)) = i\}$. Les ensembles $CNeg(x)$ et $CPos(x)$ $(x \in V)$, donnés en entrée à la procédure OrderPos peuvent être calculés à partir de F en temps $O(N)$. Il est en outre facile de vérifier que la procédure Order Pos a une complexité linéaire.

Soit $G(F)$ le graphe dont l'ensemble des sommets est V (on rappelle que V est l'ensemble des variables de F) et dont l'ensemble des arcs est $E = \{\{x, y\}/card(CNeg(x)) = card(CNeg(y))$, et il existe $C \in F$ tel que x et y apparaisse consécutivement dans $ordPos(C)\}$. Par définition, les agrégats de F sont des composantes connexes de $G(F)$. On peut observer que $card(E) \leq card(pos(C_1)) + \ldots + card(pos(C_m))$ avec $F = \{C_1, \ldots, C_m\}$. Donc les agrégats peuvent être construits en temps linéaire à l'aide d'un parcours du graphe (parcours en profondeur par exemple). □

Procédure OrderPos
Entrée : Les ensembles $CNeg(x)$ et $CPos(x)$ $(x \in V)$;
Sortie : Les listes $ordPos(C)(C \in F)$;
début
  pour tout $C \in F$ **faire**
    $ordPos(C) \leftarrow \emptyset$;
  pour $i = 1$ **jusqu'à** $card(F)$ **faire**
    $W[i] \leftarrow \emptyset$;
  pour tout $x \in V$ **faire**
    $W[card(CNeg(x))] \leftarrow W[card(CNeg(x))] \cup \{x\}$;
  pour $i = 1$ **jusqu'à** $card(F)$
    pour tout $x \in W[i]$ **faire**
      pour tout $C \in CPos(x)$ **faire**
        insérer x au début de $ordPos(C)$;
fin

FIG. 3.4 – *Procédure OrderPos*

3.5 Reconnaissance de formules Horn élargies simples

Dans cette section nous présentons un algorithme linéaire pour la reconnaissance des formules Horn élargies simples. On va voir que pour toute formule F Horn élargie simple, on peut construire une arborescence reconnaissant F, en utilisant la forêt associée à ses agrégats que nous avons mise en évidence à la section précédente.

Proposition 49 *Supposons que F est Horn élargie simple. Soit T une arborescence reconnaissant F et A un agrégat.*

1. *Pour toute clause $C \in F$, si $A \cap pos(C) \neq \emptyset$ et si A n'est pas le premier élément de $lAgregats(C)$, alors $foot(T_A)$ est le premier arc du chemin orienté $T_{A \cap pos(C)}$.*

2. *Pour tout agrégat A' tel que $PRED(A') = \{A\}$ et pour toute clause $C \in F$, si A et A' appartiennent à $lAgregats(C)$, alors le parent de $foot(T_{A'})$ est le dernier arc de $T_{A \cap pos(C)}$.*

Preuve : Découle directement du Lemme 45. □

Proposition 50 *Soit A un agrégat et T une arborescence étiquetée avec les éléments de A. Si pour toute clause $C \in F$, $T_{A \cap pos(C)}$ est un chemin orienté de T, alors T a un pied.*

Preuve : La preuve est la même que celle de la proposition 44(2).

Par définition d'un agrégat, il existe une permutation $(C_1, \ldots, C_k)$ des éléments de $\{C \in F / A \cap pos(C) \neq \emptyset\}$, telle que pour tout i $(1 < i \leq k)$ il existe j $(1 \leq j < i)$ tel que $pos(C_i) \cap pos(C_j) \neq \emptyset$. Pour tout i $(1 \leq i \leq k)$ $T_{A \cap pos(C_i)}$ est un chemin orienté (Prop. 44 (1)). Le résultat découle de ce que pour chaque $U \subseteq V$ et$W \subseteq V$, si T_U est une arborescence avec un pied, T_W est un chemin orienté, et $U \cap W \neq \emptyset$, alors $T_{U \cup W}$ est une arborescence avec un pied. □

Définition 22 (Arborescence acceptable) *Soient A un agrégat et T une arborescence étiquetée avec les éléments de A, tels que pour toute clause $C \in F$, $T_{A \cap pos(C)}$ est un chemin orienté. L'arborescence T est* acceptable *pour A si T satisfait les conditions suivantes*

1. *Pour toute clause $C \in F$, si $A \cap pos(C) \neq \emptyset$ et A n'est pas le premier élément de $lAgregats(C)$ alors $foot(T)$ est le premier arc de $T_{A \cap pos(C)}$.*

2. *Pour tout agrégat A' tel que $PRED(A') = \{A\}$, il existe $x_{A'} \in A$ tel que pour toute clause $C \in F$, si A et A' appartiennent à $lAgregats(C)$, alors $\overrightarrow{x_{A'}}$ est le dernier arc de $T_{A \cap pos(C)}$.*

L'arc $\overrightarrow{x_{A'}}$ sera appelé $anchor(A', T)$.

Exemple : La figure Fig. 3.5 représente une arborescence T acceptable pour l'agrégat A_1 de la formule F_0. On a $anchor(A_2, T) = \overrightarrow{x_7}$, et $anchor(A_3, T) = \overrightarrow{x_{13}}$. On peut vérifier que pour chaque clause $C \in F_0$, $A_1 \cap pos(C)$ est un chemin orienté de T (parfois vide) : $A_1 \cap pos(C_5) = \{x_1, x_5\}$,

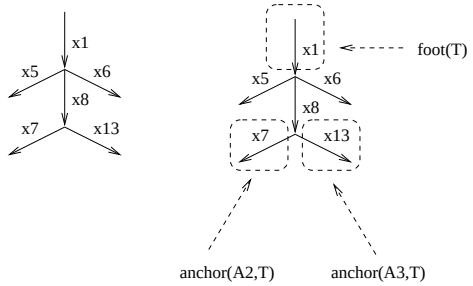

FIG. 3.5 – *T : Une arborescence acceptable pour A_1*

$A_1 \cap pos(C_6) = \{x_1, x_8, x_{13}\}$, $A_1 \cap pos(C_7) = \{x_1, x_7, x_8\}$, $A_1 \cap pos(C_8) = \{x_1, x_6\}$.

Proposition 51 *Si F est une formule Horn élargie simple, alors tout agrégat de F admet une arborescence acceptable.*

Preuve : Si F est Horn élargie simple par rapport à T , alors T_A est une arborescence acceptable pour A (Prop. 49). □

Nous montrons maintenant comment on peut construire une arborescence reconnaissant F en utilisant les arborescences acceptables et la relation $PRED$.

Définition 23 ($R(t)$) *Pour tout agrégat A, soit $t(A)$ un arborescence acceptable pour A, et supposons $card(PRED(A)) \leq 1$. On définit l'arborescence $R(t)$ comme ceci :*

1. *L'ensemble des sommets de $R(t)$ est $V \cup \{r\}$, où r est la racine de $R(t)$ et $r \notin V$.*

2. *L'ensemble des arcs de $R(t)$ est l'union des ensembles suivants :*

   - *$\{(r, x)/$ il existe un agrégat A tel que $x \in A, PRED(A) = \emptyset$ et $foot(t(A)) = \overrightarrow{x}\}$*
   - *$\{(x, y)/$ il existe deux agrégats A et A' tels que $x \in A, y \in A', PRED(A') = \{A\}, foot(t(A')) = \overrightarrow{y}$ et $anchor(A', t(A)) = \overrightarrow{x}\}$*
   - *$\{(x, y)/$ il existe un agrégat A tel que $x, y \in A$, et $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ dans $t(A)\}$.*

3. *Chaque arc (x, y) est étiqueté par y.*

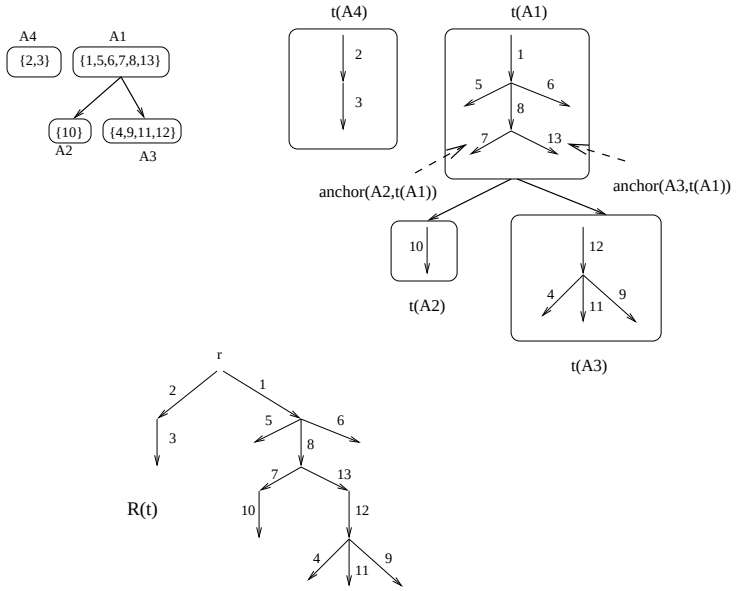

FIG. 3.6 – *Construction de $R(t)$*

Exemple : La figure 3.6 représente la construction d'une arborescence $R(t)$ pour F_0.

Proposition 52 *Supposons que tout agrégat A satisfasse les conditions suivantes :*

1. $\forall x, y \in A, CNeg(x) = CNeg(y)$;
2. $card(PRED(A)) \leq 1$;
3. *Il existe une arborescence $t(A)$ pour A ;*
4. *Pour tout agrégat A' tel que $PRED(A') = \{A\}$, $CNeg(A') \subseteq CNeg(A)$.*

Alors F est Horn élargie simple par rapport à $R(t)$.

Preuve : Soit $T = R(t)$. Dans un premier temps on prouve que pour toute clause $C \in F$, $T_{neg(C)}$ est une arborescence dont la racine est la racine de T. Il est suffisant de prouver que pour tous $x, y \in V$, si $\overrightarrow{x}$ est un ancêtre de $\overrightarrow{y}$ dans T, alors $CNeg(y) \subseteq CNeg(x)$. Par construction de T il existe une séquence d'agrégats $A_1, \ldots, A_k$ telle que $x \in A_1, y \in A_k$ et $PRED(A_i) = \{A_{i-1}\}$ (pour $1 < i \leq k$). La condition 4 nous donne $CNeg(A_k) \subseteq CNeg(A_1)$, et la condition 1 implique $CNeg(A_k) = CNeg(y)$ et $CNeg(A_1) = CNeg(x)$. Donc $CNeg(y) \subseteq CNeg(x)$. Maintenant on prouve que pour toute clause $C \in F$, $T_{pos(C)}$

est un chemin orienté de T. Soit $lAgregats(C) = (A_1, \ldots, A_k)$. Par hypothèse $T_{A_i \cap pos(C)}$ est un chemin orienté de $t(A_i)$ $(1 \leq i \leq k)$ et par construction de T, $T_{pos(C)}$ est la concaténation des chemins orientés $T_{A_1 \cap pos(C)}, \ldots, T_{A_k \cap pos(C)}$. □

L'algorithme Horn Élargie Simple détermine si une formule F est Horn élargie simple. Si c'est le cas, l'algorithme retourne une arborescence reconnaissant F, sinon l'algorithme retourne *faux*. Sa correction vient des propositions 43, 46, 47, 51 et 52.

Algorithme Horn Élargie Simple
Entrée : Une formule F.
Sortie : Un arborescence reconnaissant F si F est Horn élargie simple, *faux* sinon.

1. Construire les agrégats de F. Si un agrégat contient deux variables x et y telles que $CNeg(x) \neq CNeg(y)$ alors retourner *faux* (Prop. 43).
2. S'il existe A tel que $card(PRED(A)) > 1$, alors retourner *faux* (Prop. 46).
3. S'il existe un agrégat n'ayant pas d'arborescence acceptable, alors retourner *faux*. (Prop. 51) ; sinon, construire une arborescence $t(A)$ pour tout agrégat A.
4. Pour tout agrégat A, s'il existe A' tel que $PRED(A') = \{A\}$ et $CNeg(A') \not\subseteq CNeg(A)$, alors retourner *faux* (Prop. 47).
5. Construire $R(t)$; retourner $R(t)$ (Prop. 52).

Proposition 53 *l'algorithme Horn Élargie Simple est de complexité linéaire.*

Preuve : Rappelons que $F = \{C_1, \ldots, C_m\}$, et $N = card(C_1) + \ldots + card(C_m)$. La Prop. 48 implique que l'on peut calculer en temps linéaire les agrégats de F. Pour tout agrégat $A = \{x_1, \ldots, x_k\}$, il est possible de tester en temps $O(card(CNeg(x_1)) + \ldots + card(CNeg(x_k)))$ si $CNeg(x_1) = \ldots = CNeg(x_k)$, en utilisant un tableau indexé par les clauses. Donc, le premier pas de l'algorithme s'exécute en temps linéaire car $\sum_{x \in V} card(CNeg(x)) \leq N$. Les ensembles $PRED(A)$ peuvent être calculés en temps linéaire à partir des listes $ordPos(C)$ $(C \in F)$ calculées dans la preuve de la Prop. 48. Le second pas de l'algorithme est donc aussi de complexité linéaire. Pour tout agrégat A, soit $SUC(A) = \{A'/PRED(A') = \{A\}\}$. Soit A un agrégat, $SUC(A) = \{A_1, \ldots, A_k\}$, et $x_0 \in A$, $x_1 \in A_1$, ..., $x_k \in A_k$. On peut tester si $CNeg(x_1) \subseteq CNeg(x_0)$, ..., $CNeg(x_k) \subseteq CNeg(x_0)$ en temps $O(card(CNeg(x_0)) + \ldots + card(CNeg(x_k)))$. La complexité du quatrième pas de cet algorithme

est donc $O(N)$. Nous montrons à la section suivante que les arborescences acceptables peuvent être calculées en temps linéaires. Une fois ces arborescences calculées, il est facile de construire $R(t)$ en temps linéaire. La complexité totale de cet algorithme est donc $O(N)$. □

3.6 Calcul des arborescences acceptables

Soit A un agrégat et $\{B_1, \ldots, B_k\} = \{A \cap pos(C)/\exists C \in F, A \cap pos(C) \neq \emptyset\}$. Nous devons construire, si possible, une arborescence T dont les arcs sont étiquetés (de manière unique) par les éléments de A et tel que pour tout i $(1 \leq i \leq k)$, T_{B_i} est un chemin orienté. C'est exactement le problème que Swaminathan et Wagner ont appelé problème de réalisation arborescente [46].

Définition 24 (Réalisation arborescente) *Soient $\mathcal{S} = \{S_1, \ldots, S_k\}$ une collection d'ensembles finis et T une arborescence dont les arcs sont étiquetés (de manière unique) par les éléments de $S_1 \cup \ldots \cup S_k$. T est une réalisation arborescente de $\mathcal{S}$ si T_{S_i} est un chemin orienté de T $(1 \leq i \leq k)$.*

Exemple : Soit $\mathcal{S} = \{\{1,3,5\}, \{1,3,6\}, \{3,5\}, \{1,2\}, \{3,4\}\}$. L'arborescence T (Fig. 3.7) est une réalisation arborescente de $\mathcal{S}$.

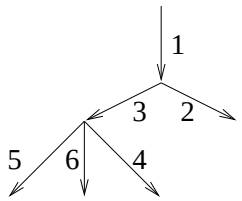

FIG. 3.7 – *T est une réalisation arborescente de $\mathcal{S}$*

Soit $\mathcal{S} = \{S_1, \ldots, S_k\}$ une collection d'ensembles finis. On appelle $G(\mathcal{S})$ le graphe dont l'ensemble des sommets est $\mathcal{S}$, et l'ensemble des arêtes est $\{\{S_i, S_j\}/S_i \cap S_j \neq \emptyset, i \neq j\}$. On remarque que si $G(\mathcal{S})$ est connexe alors tout réalisation arborescente de $\mathcal{S}$ a un pied.

Soit A un agrégat et $\mathcal{A} = \{B_1, \ldots, B_k\} = \{A \cap pos(C)/\exists C \in F, A \cap pos(C) \neq \emptyset\}$. Par définition d'un agrégat, $G(\mathcal{A})$ est connexe. La condition 1 de la définition d'une arborescence acceptable T pour A (Déf. 22), exige que des chemins orientés T_{B_i} aient comme origine la racine de T. La condition 2 demande que certains chemins orientés aient le même dernier arc.

Lemme 54 *Soit* $\mathcal{S} = \{S_1, \ldots, S_k\}$ *une collection d'ensembles finis telle que* $G(\mathcal{S})$ *est connexe. Soit* $X = \{b, e_1, \ldots, e_k, f_1, \ldots, f_k\}$ *tel que* $X \cap (S_1 \cup \ldots \cup S_k) = \emptyset$, *et* $I \subseteq \{1, \ldots, k\}$. *Il existe une réalisation arborescente* T *de* $\mathcal{S}$, *telle que pour chaque* $i \in I$, T_{S_i} *commence à la racine de* T, *si et seulement si il existe une réalisation arborescente de* $\mathcal{S} \cup \bigcup_{i \in I} \{S_i \cup \{b\}, S_i \cup \{b, e_i\}, S_i \cup \{b, f_i\}, S_i \cup \{e_i\}, S_i \cup \{f_i\}\}$.

Preuve :
($\Rightarrow$) Soit T une réalisation arborescente telle que pour tout $i \in I$, S_i commence à la racine de T. Soit T' l'arborescence obtenue à partir de T en ajoutant les arcs suivant : $\overrightarrow{b}$ est le pied de T', son extrémité est la racine de T. Les arcs $\overrightarrow{e_i}$ et $\overrightarrow{f_i}$ sont rajoutés, pour tout $i \in I$ comme les enfants du dernier arc de T_{S_i}. Il est trivial de vérifier que dans un tel arbre les contraintes rajoutées dans le lemme ci-dessus sont vérifiées.
($\Leftarrow$) Soit T' une réalisation arborescente des contraintes données ci-dessus. Soit T la forêt obtenue en retirant de T' les arcs $\overrightarrow{e_i}$ $\overrightarrow{f_i}$ (pour tout $i \in I$) et $\overrightarrow{b}$. Comme $G(\mathcal{S})$ est connexe, $T'_{(S_1 \cup S_2 \cup \ldots \cup S_k)}$ est aussi connexe, donc T est une arborescence. Soit j un élément quelconque de I, T'_{S_j} est un chemin car on a la contrainte S_j. Comme nous avons la contrainte $S_j \cup b$, l'arc $\overrightarrow{b}$ forme un chemin avec T'_{S_j}. Comme on a les contraintes, $S_j \cup \{e_j\}$ et $S_j \cup \{e_j, b\}$, on voit que $T'_{S_j \cup \{e_j, b\}}$ est un chemin dont $\overrightarrow{b}$ et $\overrightarrow{e_j}$ sont les deux extrémités. Il en est de même pour f_j. On voit que la seule solution est que l'arc $\overrightarrow{b}$ soit parent du premier arc de T'_{S_j} et $\overrightarrow{e_j}$ ainsi que $\overrightarrow{f_j}$ sont des enfants du dernier arc de T'_{S_j}. Comme $T'_{(S_1 \cup S_2 \cup \ldots \cup S_k)}$ est connexe, on a obligatoirement que $\overrightarrow{b}$ est le parent de la racine de $T'_{(S_1 \cup S_2 \cup \ldots \cup S_k)}$, ce qui implique que pour tout $i \in I$, T_{S_i} commence à la racine de T. □

On voit que si T' est une réalisation arborescente de $\mathcal{S} \cup \bigcup_{i \in I} \{S_i \cup \{b\}, S_i \cup \{b, e_i\}, S_i \cup \{b, f_i\}, S_i \cup \{e_i\}, S_i \cup \{f_i\}\}$, alors le graphe orienté T obtenu en retirant de T' les arcs $\overrightarrow{b}$, $\overrightarrow{e_i}$ et $\overrightarrow{f_i}$ ($i \in I$), est une réalisation arborescente de $\mathcal{S}$ telle que pour tout $i \in I$, T_{S_i} commence à la racine de T. Le pied de T est l'arc (unique) de T' dont le parent est $\overrightarrow{b}$.

Lemme 55 *Soit* $\mathcal{S} = \{S_1, \ldots, S_k\}$ *une collection d'ensembles finis telle que* $G(\mathcal{S})$ *est connexe. Soit* $X = \{g, h\}$ *tel que* $X \cap (S_1 \cup \ldots \cup S_k) = \emptyset$, *et* $I \subseteq \{1, \ldots, k\}$. *Il existe une réalisation arborescente* T *de* $\mathcal{S}$ *telle que les chemins orientés* T_{S_i} ($i \in I$) *ont tous le même dernier arc, si et seulement s'il existe une réalisation arborescente de* $\mathcal{S} \cup \bigcup_{i \in I} \{S_i \cup \{g\}, S_i \cup \{h\}\}$.

Preuve : Soit T' une réalisation arborescente de $\mathcal{S} \cup \bigcup_{i \in I} \{S_i \cup \{g\}, S_i \cup \{h\}\}$. Le graphe orienté T obtenu en retirant de T' les arcs $\overrightarrow{g}$ et $\overrightarrow{h}$ est une réalisation arborescente de $\mathcal{S}$ telle que tous les chemins orientés de T_{S_i} ($i \in I$) partagent le même dernier arc. On remarque en outre qu'au

moins un des arcs $\overrightarrow{g}$ ou $\overrightarrow{h}$ a son parent dans T', et que s'ils ont tous les deux un parent, alors c'est le même pour les deux. Supposons que $\overrightarrow{g}$ ait un parent $\overrightarrow{x}$ dans T'. Alors $\overrightarrow{x}$ est le dernier arc du chemin orienté T_{S_i} $(i \in I)$. □

On utilise les résultats des Lemmes 54 et 55 pour construire nos arborescences acceptables. On associe à chaque clause $C \in F$ deux nouveaux symboles e_c et f_c ($e_c \notin V$, $f_c \notin V$), et à chaque agrégat A les symboles nouveaux g_A et h_A ($g_A \notin V$, $h_A \notin V$). La définition suivante utilise en outre le symbole b ($b \notin V$).

Définition 25 (contraintes(A)) *Soient A un agrégat et $C \in F$ une clause tels que $A \cap pos(C) \neq \emptyset$. On définit les ensembles $\mathcal{D}(C)$, $\mathcal{E}(C)$, $\mathcal{F}(C)$ et $\mathcal{G}(C)$ comme ceci :*

- $\mathcal{D}(C) = \{A \cap pos(C)\}$;
- *Si A n'est pas le premier élément de $lAgregats(C)$, alors* $\mathcal{E}(C) = \{(A \cap pos(C)) \cup \{b\}, (A \cap pos(C)) \cup \{b, e_C\}, (A \cap pos(C)) \cup \{b, f_C\}, (A \cap pos(C)) \cup \{e_C\}, (A \cap pos(C)) \cup \{f_C\}\}$, *sinon* $\mathcal{E}(C) = \emptyset$;
- *Si A a un successeur A' dans $lAgregats(C)$, alors* $\mathcal{F}(C) = \{(A \cap pos(C)) \cup \{g_{A'}\}, (A \cap pos(C)) \cup \{h_{A'}\}\}$, *sinon* $\mathcal{F}(C) = \emptyset$;
- $\mathcal{G}(C) = \mathcal{D}(C) \cup \mathcal{E}(C) \cup \mathcal{F}(C)$.

Soit $contraintes(A) = \mathcal{G}(C_1) \cup \ldots \cup \mathcal{G}(C_k)$ *où* $\{C_1, \ldots, C_k\} = \{C \in F / A \cap pos(C) \neq \emptyset\}$.

On remarque que si $CNeg(A) = \emptyset$ alors $\mathcal{F}(C_i) = \emptyset$ $(1 \leq i \leq k)$ puisque, par définition de $lAgretgats(C_i)$, A n'a pas de successeur dans $lAgregats(C_i)$.

Proposition 56 *Soit A un agrégat. Il existe une arborescence acceptable pour A si et seulement s'il existe une réalisation arborescente de $contraintes(A)$.*

Preuve : Conséquence des Lemmes 54 et 55. □

Soit T' une réalisation arborescente de $contraintes(A)$. Le graphe orienté T obtenu en enlevant les arcs $\overrightarrow{b}$, $\overrightarrow{e_C}$ et $\overrightarrow{f_C}$ $(C \in F)$, $\overrightarrow{g_{A'}}$ et $\overrightarrow{h_{A'}}$ (pour tout agrégat A'), de T' est une arborescence acceptable pour A. Le pied de T est le seul arc de T' dont le parent est $\overrightarrow{b}$. Soit A' un agrégat tel que $PRED(A') = \{A\}$. On peut supposer que $\overrightarrow{g_{A'}}$ a un parent $\overrightarrow{x}$ dans T'. Alors $anchor(A', T) = \overrightarrow{x}$.

Procédure Construit Contraintes
Entrée : Les agrégats de F et les listes $ordPosAg(C)$ $(C \in F)$;
Sortie : Les ensembles $contraintes(A)$;
début
  pour tout agrégat A **faire**
    $contraintes(A) \leftarrow \emptyset$;
  pour toute clause $C \in F$ telle que $pos(C) \neq \emptyset$ **faire**
  début
    Soit $(A_1 \cap pos(C), \ldots, A_k \cap pos(C)) = ordPosAg(C)$;
    pour $i = 1$ **jusqu'à** k **faire**
      $contraintes(A_i) \leftarrow contraintes(A_i) \cup (A_i \cap pos(C))$;
    pour $i = 2$ **jusqu'à** k **faire**
      $contraintes(A_i) \leftarrow contraintes(A_i) \cup \{(A_i \cap pos(C)) \cup \{b\}, (A_i \cap pos(C)) \cup \{b, e_C\},$
        $(A_i \cap pos(C)) \cup \{b, f_C\}, (A_i \cap pos(C)) \cup \{e_C\}, (A_i \cap pos(C)) \cup \{f_C\}\}$;
    pour $i = 1$ **jusqu'à** $k - 1$ **faire**
      $contraintes(A_i) \leftarrow contraintes(A_i) \cup \{(A_i \cap pos(C)) \cup \{g_{A_{i+1}}\},$
        $(A_i \cap pos(C)) \cup \{h_{A_{i+1}}\}\}$;
  fin
fin.

FIG. 3.8 – *Procédure Construit Contraintes*

Swaminathan et Wagner présentent [46] un algorithme qui détermine, si pour un ensemble $\mathcal{S} = \{S_1, \ldots, S_k\}$, il existe une réalisation arborescente de $\mathcal{S}$ et, s'il en existe une la construit. Leur algorithme est de complexité quasi-linéaire. Plus précisément sa complexité est $O(n.\alpha(n, r))$, où $n = card(S_1) + \ldots + card(S_k)$, $r = card(S_1 \cup \ldots \cup S_k)$ et $\alpha(n, r)$ est l'inverse fonctionnel de la fonction d'Ackermann. Swaminathan et Wagner ont aussi remarqué que le problème de réalisation arborescente peut se résoudre en temps linéaire en utilisant les résultats de Dietz et al. [21].

On rappelle que $F = \{C_1, \ldots, C_m\}$ et $N = card(C_1) + \ldots + card(C_m)$.

Proposition 57 *Les arborescences acceptables de F peuvent se calculer en temps $O(N)$.*

Preuve : Soit A un agrégat, $contraintes(A) = \{E_1, \ldots, E_h\}$ et $N_A = card(E_1) + \ldots + card(E_h)$. Par définition de $contraintes(A)$, $\sum N_A$ est $O(N)$. Soit $C \in F$ et $(A_1, \ldots, A_k) = lAgregats(C)$. On note $ordPosAg(C)$ la liste $(A_1 \cap pos(C), \ldots, A_k \cap pos(C))$. Les listes $ordPosAg(C)$ $(C \in F)$ peuvent être obtenues en temps linéaire à partir des listes $ordPos(C)$ définies dans la preuve de la Prop. 48. Les listes $ordPosAg(C)$ $(C \in F)$ sont données en entrées de la procédure Construit-Contraintes (Fig. 3.8) qui calcule les ensembles $contraintes(A)$. Il est aisé de vérifier que la complexité de la procédure Construit Contraintes est $O(N)$. Une arborescence acceptable pour un agrégat A peut être calculée en temps $O(N_A)$

en utilisant l'algorithme de Dietz et al. [21]. Le résultat découle de ce que $\sum N_A$ soit $O(N)$. □

3.7 Reconnaissance des formules Horn étendues simples

Nous présentons ici un algorithme linéaire pour la reconnaissance des formules Horn étendues simples. Nous verrons que si toute variable apparaît négativement dans F, alors il n'est pas nécessaire de savoir résoudre le problème réalisation arborescente, pour décider si une formule est Horn étendue simple. La classe des formules Horn étendues simples est une sous-classe des formules Horn élargies simples, donc toutes les propriétés décrites dans les sections précédentes sont toujours vraies. Soit une clause $C \in F$ et T une arborescence telles que C soit Horn élargie simple par rapport à T. Par définition $T_{neg(C)}$ est une arborescence dont la racine est la racine de T, ce qui correspond à la définition d'une clause Horn élargie simple par rapport à T. La clause C est Horn étendue simple, si en plus $T_{neg(C)}$ est une union disjointe par arcs de chemins orientés de T. Examinons les conséquences de cette exigence supplémentaire.

Proposition 58 *Supposons que F est Horn étendue simple. Soient T une arborescence reconnaissant F et $x, y, z \in V$ $(y \neq z)$. Si $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ et $\overrightarrow{z}$, alors $CNeg(y) \cap CNeg(z) = \emptyset$.*

Preuve : Supposons que $CNeg(y) \cap CNeg(z) \neq \emptyset$ et soit $C \in CNeg(y) \cap CNeg(z)$. Par définition $neg(C) = N_1 \cup \ldots \cup N_k$, avec $N_i \cap N_j = \emptyset$ $(1 \leq i < j \leq k)$, et T_{N_i} est un chemin orienté qui commence à la racine de T $(1 \leq i \leq k)$. Nous avons $y, z \in neg(C)$. Il existe i $(1 \leq i \leq k)$ tel que $y \in N_i$. Par hypothèse $\overrightarrow{z}$ ne peut pas être un arc de T_{N_i}. Donc il existe un j $(i \neq j)$ $(1 \leq j \leq k)$ tel que $z \in N_j$. Par hypothèse $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ et $\overrightarrow{z}$, d'où $\overrightarrow{x}$ est un arc de T_{N_i} et T_{N_j}, et donc $x \in N_i \cap N_j$. Contradiction. □

Proposition 59 *Supposons que F est Horn étendue simple. Soient T une arborescence reconnaissant F, et A un agrégat. Si $CNeg(A) \neq \emptyset$ alors T_A est un chemin orienté de T.*

Preuve : Supposons qu'il existe $x, y, z \in A$ $(y \neq z)$, tels que $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ et $\overrightarrow{z}$ dans T_A. Nous avons $CNeg(y) = CNeg(z) = CNeg(A)$ (Prop. 43), et $CNeg(y) \cap CNeg(z) = \emptyset$ (Prop. 58), d'où $CNeg(A) = \emptyset$. Contradiction. □

Proposition 60 *Supposons que F est Horn étendue simple. Soit T une arborescence reconnaissant F et A un agrégat tel que $CNeg(A) \neq \emptyset$. Pour tout agrégat A' tel que $PRED(A') = \{A\}$ et $CNeg(A') \neq \emptyset$, le parent de $foot(T_{A'})$ est le dernier arc de T_A.*

Preuve : Le lemme 45 implique que le parent de $foot(T_{A'})$ est dans T_A. Soient $\overrightarrow{z} = foot(T_{A'})$ et $\overrightarrow{x}$ le parent de $\overrightarrow{z}$. Supposons que $\overrightarrow{x}$ n'est pas le dernier arc de T_A. Soit $y \in A$ tel que $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$. On a $CNeg(y) \cap CNeg(z) = \emptyset$ (Prop. 58), d'où $CNeg(A) \cap CNeg(A') = \emptyset$. Mais $CNeg(A') \neq \emptyset$ et $CNeg(A') \subseteq CNeg(A)$ (Prop. 47). Contradiction. □

Nous définissons maintenant les arborescences viables qui jouent pour les formules Horn étendues simples le même rôle que les arborescences acceptables pour les formules Horn élargies simples.

Définition 26 (Arborescence viable) *Soient A un agrégat, et T une arborescence acceptable pour A. T est* viable *si T satisfait les conditions suivantes lorsque $CNeg(A) \neq \emptyset$:*

1. *T est un chemin orienté.*
2. *Pour tout agrégat A' tel que $PRED(A') = \{A\}$ et $CNeg(A') \neq \emptyset$, $anchor(A', T)$ est le dernier arc de T.*

Proposition 61 *Si F est Horn étendue simple, alors tout agrégat de F admet une arborescence viable.*

Preuve : Conséquence directe des propositions : Prop. 51, Prop. 59 et Prop. 60. □

Proposition 62 *Supposons que F est Horn étendue simple. Soient A, A' et A'' trois agrégats ($A' \neq A''$) tels que $PRED(A') = PRED(A'') = \{A\}$. Alors $CNeg(A') \cap CNeg(A'') = \emptyset$*

Preuve : Supposons que $CNeg(A') \neq \emptyset$ et $CNeg(A'') \neq \emptyset$. Nous avons $CNeg(A) \neq \emptyset$, car $PRED(A') = \{A\}$. Soit $\overrightarrow{x}$ le dernier arc du chemin orienté T_A. Soit $\overrightarrow{y} = foot(T_{A'})$. La Prop. 60 implique que $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ et $\overrightarrow{z}$, donc $CNeg(y) \cap CNeg(z) = \emptyset$ (Prop. 58) et $CNeg(A') \cap CNeg(A'') = \emptyset$. □

Proposition 63 *Supposons que tout agrégat A satisfait les conditions suivantes :*

1. *$\forall x, y \in A, CNeg(x) = CNeg(y)$;*

2. $card(PRED(A)) \leq 1$;
3. *Il existe une arborescence viable* $t(A)$ *pour* A ;
4. *Pour tout agrégat* A' *tel que* $PRED(A') = \{A\}$, $CNeg(A') \subseteq CNeg(A)$.
5. *Pour tous agrégats* A' *et* A'' *tels que* $PRED(A') = PRED(A'') = \{A\}$, $CNeg(A') \cap CNeg(A'') = \emptyset$.

Alors F *est Horn étendue simple par rapport à* $R(t)$.

Preuve : Soit $T = R(t)$. La Prop. 52 nous donne que F est Horn élargie simple par rapport à T. Soient $x, y, z \in V$ $(y \neq z)$, tels que $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$ et $\overrightarrow{z}$ dans T. Il suffit de montrer que $CNeg(y) \cap CNeg(z) = \emptyset$. Soient A, A' et A'' trois agrégats tels que $x \in A$, $y \in A'$ et $z \in A''$. Nous avons deux cas à considérer :

1. $A' = A''$. L'arborescence $t(A')$ a un pied, donc $x \in A'$ et $A = A' = A''$. L'arborescence $T_A = t(A)$ n'est pas un chemin orienté et donc $CNeg(A) = \emptyset$, puisque par hypothèse $t(A)$ est viable. D'où $CNeg(y) = CNeg(z) = CNeg(A) = \emptyset$.
2. $A' \neq A''$. Supposons que $CNeg(A') \neq \emptyset$ et $CNeg(A'') \neq \emptyset$. Les arborescences $t(A')$ et $t(A'')$ sont des chemins orientés. Si $x \in A'$, alors par construction de T $PRED(A'') = \{A'\}$ et $\overrightarrow{x} = anchor(A'', t(A'))$; de plus $\overrightarrow{x}$ est le dernier arc de $t(A')$ donc $t(A')$ est viable. Ce qui contredit le fait que $\overrightarrow{x}$ est le parent de $\overrightarrow{y}$, donc $x \notin A'$. De même on peut voir que $x \notin A''$. Par conséquent $PRED(A') = PRED(A'') = \{A\}$ par construction de T. Le résultat est une conséquence de la condition 5.

□

L'algorithme Horn Étendue Simple détermine si un formule F est Horn étendue simple. Si F est Horn étendue simple, l'algorithme retourne une arborescence reconnaissant F, sinon il retourne *faux*. La correction de cet algorithme vient des propositions 43, 46, 47, 61, 62 et 63.

Algorithme Horn Étendue Simple
Entrée : Une formule F.
Sortie : Une arborescence T telle que F est Horn étendue simple par rapport à T si F est Horn étendue simple.
faux sinon.

1. Construire les agrégats de F. Si un agrégat contient deux variables x et y telles que $CNeg(x) \neq CNeg(y)$ alors retourner *faux* (Prop. 43).

2. S'il existe un agrégat A tel que $card(PRED(A)) > 1$, alors retourner *faux* (Prop. 46).

3. S'il existe un agrégat n'ayant pas d'arborescence viable, alors retourner *faux* (Prop. 61) ; sinon construire une arborescence viable $t(A)$ pour tout agrégat A.

4. Pour tout agrégat A, s'il existe un agrégat A' tel que $PRED(A') = \{A\}$ et $CNeg(A') \not\subseteq CNeg(A)$, alors retourner *faux* (Prop. 47).

5. Pour tout agrégat A, s'il existe A' et A'' tels que $PRED(A') = PRED(A'') = \{A\}$ et $CNeg(A') \cap CNeg(A'') \neq \emptyset$, alors retourner *faux* (Prop. 62).

6. Construire $R(t)$; retourner $R(t)$ (Prop. 63).

Proposition 64 *L'algorithme Horn Étendue Simple a une complexité linéaire.*

Preuve : Nous avons prouvé dans la Prop. 53 que les étapes 1,2 et 4 se font en temps $O(N)$. Pour tout agrégat A, soit $SUC(A) = \{A'/PRED(A') = \{A\}\}$. Soit A un agrégat, $SUC(A) = \{A_1, \ldots, A_k\}$, et $x_1 \in A_1$, ..., $x_k \in A_k$. Nous pouvons tester si $CNeg(A_i) \cap CNeg(A_j) = \emptyset$ $(1 \leq i < j \leq k)$ en temps $O(card(CNeg(x_1)) + \ldots + card(CNeg(x_k)))$, en utilisant un tableau indexé par les clauses. Donc la complexité de la cinquième étape est $O(N)$. Nous prouvons dans la Sec. 3.9 que les arborescences viables peuvent être calculées en temps linéaire. Donc, la complexité totale de l'algorithme est $O(N)$. □

Dans la section suivante (Sec. 3.8) nous présentons un algorithme simple pour calculer les arborescences viables lorsque toute variable a au moins une occurrence négative dans F. Dans celle d'après (Sec. 3.9) nous étudions le calcul des arborescences viables dans le cas général.

3.8 Un cas facile

Dans cette section nous supposons que toute variable apparaît négativement dans F, c'est à dire que pour tout $x \in V$, $CNeg(x) \neq \emptyset$. Pour tout agrégat A, $CNeg(A) \neq \emptyset$, donc, par définition, les arborescences viables sont des chemins orientés. Soient A un agrégat, T une arborescence viable pour A (T est un chemin orienté), et π une permutation des éléments de A associée à T (x est le ième élément de π si $\overrightarrow{x}$ est le ième arc de T). Soit $C \in F$ tel que $A \cap pos(C) \neq \emptyset$ et soit $B = A \cap pos(C)$. Par définition, les éléments de B apparaissent consécutivement dans π. De plus s'il existe A_0 (resp. A') tel que $PRED(A) = \{A_0\}$ (resp. $PRED(A') = \{A\}$) et

$A_0 \cap pos(C) \neq \emptyset$ (resp. $A' \cap pos(C) \neq \emptyset$) alors le premier (resp. dernier) élément de π appartient à B. On peut voir que par hypothèse $CNeg(A') \neq \emptyset$ et donc la condition 2 de la Déf. 26 doit être satisfaite. Trouver une arborescence viable se réduit donc à trouver une permutation des éléments de A qui satisfasse les contraintes énoncées à la section précédente.

Définition 27 (Permutation permise) *Soit $\mathcal{E} = \{E_1, \ldots, E_k\}$ une collection d'ensembles finis et $U = E_1 \cup \ldots \cup E_k$. Une permutation π des éléments de U est* permise *respectivement à $\mathcal{E}$, si pour tout i $(1 \leq i \leq k)$ les éléments de E_i apparaissent consécutivement dans π.*

Exemple : Soit $U = \{A, B, C, D, E, F\}$ et $\mathcal{E} = \{\{A, C, F\}, \{B, C, D, F\}, \{B, E\}\}$. Les permutations permises de U respectivement à $\mathcal{E}$ sont : (A, C, F, D, B, E), (A, F, C, D, B, E), (E, B, D, F, C, A) et (E, B, D, C, F, A).

Remarque 2 *Si $(x_1, \ldots, x_n)$ est une permutation permise de U respectivement à $\mathcal{E}$, alors $(x_n, \ldots, x_1)$ est aussi une permutation permise de U respectivement à $\mathcal{E}$.*

Lemme 65 *Soit $\mathcal{E} = \{E_1, \ldots, E_k\}$ une collection d'ensembles finis, $U = E_1 \cup \ldots \cup E_k$, $X = \{b, e\}$ telle que $X \cap U = \emptyset$, $I \subseteq \{1, \ldots, k\}$ et $J \subseteq \{1, \ldots, k\}$. Il existe une permutation permise π des éléments de U respectivement à $\mathcal{E}$, telle que pour tout $i \in I$ le premier élément de π appartient à E_i, et pour tout $j \in J$ le dernier élément de π appartient à E_j, si et seulement s'il existe une permutation permise des éléments de $U \cup \{b, e\}$ respectivement à $\mathcal{E} \cup \{U \cup \{b\}, U \cup \{e\}\} \cup (\bigcup_{i \in I}\{E_i \cup \{b\}\}) \cup (\bigcup_{j \in J}\{E_j \cup \{e\}\})$.*

Preuve : $\Rightarrow$ Soit π une permutation permise des éléments de U respectivement à $\mathcal{E}$, telle que pour tout $i \in I$, le premier élément de π appartient à E_i, et pour tout $j \in J$, le dernier élément de π appartient à E_j, dans ce cas, la permutation $\pi' = (b, \pi_1, \ldots, \pi_k, e)$ (où π_i désigne le i^{ime} élément de π), vérifie bien les contraintes de $\mathcal{E}$. Elle vérifie aussi la contrainte $U \cup \{b\}$ car e est à une extrémité de π', il en est de même pour la contrainte $U \cup \{e\}$. Comme pour tout $i \in I$, $\pi_1 \in E_i$, il est évident que la contrainte $E_i \cup \{b\}$ est satisfaite, pour tout $i \in I$, dans π'. Il en est de même pour les contraintes $E_j \cup \{e\}$, pour $j \in J$.
$\Leftarrow$ Soit π' une permutation permise de $U \cup \{b, e\}$ respectivement à $\mathcal{E} \cup \{U \cup \{b\}, U \cup \{e\}\} \cup (\bigcup_{i \in I}\{E_i \cup \{b\}\}) \cup (\bigcup_{j \in J}\{E_j \cup \{e\}\})$. La contrainte $U \cup b$ oblige e à être à une des extrémités de π', la contrainte $U \cup \{e\}$ oblige b à être à l'autre extrémité de π'. On peut donc supposer que b est le premier élément de π' et que e en est le dernier (si ce n'est pas le cas, on peut inverser l'ordre des éléments de π'). Comme on a les contraintes $E_i \cup \{b\}$ pour tout $i \in I$, on sait que dans π', les E_i $(i \in I)$ contiennent tous le second élément de π' (puisqu'ils sont consécutifs à b). De même

les E_j ($j \in J$) contiennent tous l'avant-dernier élément de π'. On obtient donc facilement une permutation permise de U respectivement à $\mathcal{E}$ en retirant à π' les éléments b et e. □

Nous utilisons maintenant le Lemme 65 pour exprimer les conditions qu'une arborescence viable doit vérifier. La définition suivante utilise les symboles b et e ($b \notin V$ et $e \notin V$).

Définition 28 (contraintes1(A)) *Soit A un agrégat. L'ensemble contraintes1(A) est définie comme l'union des ensembles suivants :*

- $\{A \cup \{b\}, A \cup \{e\}\}$;
- $\{A \cap pos(C) / C \in F\}$;
- $\{(A \cap pos(C)) \cup \{b\} \,/\, C \in F$ *et* A *n'est pas le premier élément de* $lAgregats(C)\}$;
- $\{(A \cap pos(C)) \cup \{e\} \,/\, C \in F$ *et* A *n'est pas le dernier élément de* $lAgregats(C)\}$.

Proposition 66 *Supposons que pour tout $x \in V$, $CNeg(x) \neq \emptyset$. Soit A un agrégat. Il existe une arborescence viable pour A si et seulement s'il existe une permutation permise de $A \cup \{b, e\}$ respectivement à contraintes1(A).*

Preuve : Le résultat découle du Lemme 65. □

Soit A un agrégat et π une permutation permise de $A \cup \{b, e\}$ respectivement à $contraintes1(A)$. Supposons que π est de la forme $(b, x_1, \ldots, x_k, e)$. On associe à π un chemin orienté de T étiqueté par les éléments de A et tel que $\overrightarrow{x_i}$ est le parent de $\overrightarrow{x_{i+1}}$ ($1 \leq i < k$). T est une arborescence viable pour A, et pour tout agrégat A' tel que $PRED(A') = \{A\}$, $anchor(A', T) = \overrightarrow{x_k}$.

Les ensembles $contraintes1(A)$ sont calculés par la procédure Construit Contraintes 1 (Fig. 3.9). Les listes $ordPosAg(C)$ ($C \in F$) données en entrée à Construit Contraintes 1 sont définies dans la preuve de la Prop. 57. Une permutation permise de $A \cup \{b, e\}$ respectivement à $contraintes1(A)$ peut être trouvée en temps linéaire en utilisant l'algorithme de Booth et Lueker [5] ou l'algorithme de Habib et al. [28, 29]. On peut donc construire les arborescences viables de F en temps $O(N)$ (la preuve est la même que celle de la Prop. 57).

3.9 Calcul des arborescences viables

On peut adapter la méthode présentée à la section 3.6 pour le calcul des arborescences acceptables. Nous n'avons qu'à changer la définition

Procédure Construit Contraintes 1
Entrée : Les agrégats de F et les listes $ordPosAg(C)$ $(C \in F)$;
Sortie : Les ensembles $contraintes1(A)$;
début
  pour tout agrégat A **faire**
    $contraintes1(A) \leftarrow \{A \cup \{b\}, A \cup \{e\}\}$;
  pour toute clause $C \in F$ telle que $pos(C) \neq \emptyset$ **faire**
  début
    Soit $(A_1 \cap pos(C), \ldots, A_k \cap pos(C)) = ordPosAg(C)$;
    pour $i = 1$ **jusqu'à** k **faire**
      $contraintes1(A_i) \leftarrow contraintes1(A_i) \cup (A_i \cap pos(C))$;
    pour $i = 2$ **jusqu'à** k **faire**
      $contraintes1(A_i) \leftarrow contraintes1(A_i) \cup \{(A_i \cap pos(C)) \cup \{b\}\}$;
    pour $i = 1$ **jusqu'à** $k - 1$ **faire**
      $contraintes1(A_i) \leftarrow contraintes1(A_i) \cup \{(A_i \cap pos(C)) \cup \{e\}\}$;
  fin
fin.

FIG. 3.9 – *Procédure Construit Contraintes 1*

des ensembles $contraintes(A)$ pour tenir compte des conditions supplémentaires induites par la définition 26. Nous devons séparer deux cas : $CNeg(A) = \emptyset$ et $CNeg(A) \neq \emptyset$. Si $CNeg(A) \neq \emptyset$ alors tout arborescence viable pour A est un chemin orienté donc nous ajoutons à $contraintes(A)$ les ensembles $A, A \cup \{b\}, A \cup \{b, e\}, A \cup \{b, f\}, A \cup \{e\}$ et $A \cup \{f\}$. On rappelle que $\{e, b\} \cap V = \emptyset$. La définition de $contraintes2(A)$ utilise le nouveau symbole f $(f \notin V)$.

Définition 29 (contraintes2(A)) *Soient un agrégat A et une clause $C \in F$ tels que $A \cap pos(C) \neq \emptyset$. On définit les ensembles $\mathcal{D}(C)$, $\mathcal{E}(C)$, $\mathcal{F}(C)$ et $\mathcal{G}(C)$:*

- $\mathcal{D}(C) = \{A \cap pos(C)\}$.
- *Si A n'est pas le premier élément de $lAgregats(C)$, alors $\mathcal{E}(C) = \{(A \cap pos(C)) \cup \{b\}, (A \cap pos(C)) \cup \{b, e_C\}, (A \cap pos(C)) \cup \{b, f_C\}, (A \cap pos(C)) \cup \{e_C\}, (A \cap pos(C)) \cup \{f_C\}\}$, sinon $\mathcal{E}(C) = \emptyset$.*
- *Si A est le dernier élément de $lAgregats(C)$ alors $\mathcal{F}(C) = \emptyset$, sinon soit A' le successeur de A dans $lAgregats(C)$;*
  Si $CNeg(A') = \emptyset$, alors $\mathcal{F}(C) = \{(A \cap pos(C)) \cup \{g_{A'}\}, (A \cap pos(C)) \cup \{h_{A'}\}\}$;
  Si $CNeg(A') \neq \emptyset$, alors $\mathcal{F}(C) = \{(A \cap pos(C)) \cup \{e\}\}$.
- $\mathcal{G}(C) = \mathcal{D}(C) \cup \mathcal{E}(C) \cup \mathcal{F}(C)$.

Soit $\{C_1, \ldots, C_k\} = \{C \in F / A \cap pos(C) \neq \emptyset\}$ *et* $\mathcal{B} = \{A, A \cup \{b\}, A \cup \{b, e\}, A \cup \{b, f\}, A \cup \{e\}, A \cup \{f\}\}$.
Si $CNeg(A) \neq \emptyset$, *alors* $contraintes2(A) = \mathcal{B} \cup \mathcal{G}(C_1) \cup \ldots \cup \mathcal{G}(C_k)$;
Si $CNeg(A) = \emptyset$, *alors* $contraintes2(A) = \mathcal{G}(C_1) \cup \ldots \cup \mathcal{G}(C_k)$.

Proposition 67 *Soit A un agrégat. Il existe une arborescence viable pour A si et seulement si il existe une réalisation arborescente de* $contraintes2(A)$.

Preuve : Se déduit des Lemme 54 et Lemme 55. □

Soit T' une réalisation arborescente de $contraintes2(A)$. Le graphe orienté T obtenu à partir de T' en retirant les arcs $\overrightarrow{b}$, $\overrightarrow{e}$, $\overrightarrow{f}$, $\overrightarrow{e_C}$ et $\overrightarrow{f_C}$ ($C \in F$), $\overrightarrow{g_{A'}}$ et $\overrightarrow{h_{A'}}$ (pour tout agrégat A'), est une arborescence viable pour A. Si $CNeg(A) \neq \emptyset$ alors T est un chemin orienté, et le parent de $\overrightarrow{e}$ dans T' est le dernier arc de T. Donc, pour tout agrégat A' tel que $PRED(A') = \{A\}$ et $CNeg(A') \neq \emptyset$, $anchor(A', T)$ est le parent de $\overrightarrow{e}$ dans T'.

Proposition 68 *Les arborescences viables de F peuvent être calculées en temps* $O(N)$.

Preuve : Cette preuve est similaire à la preuve de la Prop. 57. □

Chapitre 4

Conclusion

Dans cette partie, nous avons étudié la classe des formules Horn étendues, ainsi que les classes proches telles que Horn étendues simples, Horn élargies et Horn élargies simples. Nous avons quatre classes pour lesquelles le test de satisfaisabilité se fait à l'aide uniquement de la résolution unitaire, ce qui nous permet de proposer la génération à délai $O(nN)$ pour toutes les formules de ces classes.

Nous avons étudié la structure d'agrégats, qui nous a permis de construire des algorithmes de reconnaissance linéaires pour les formules Horn étendues simples et Horn élargies simples.

Cette structure d'agrégats et l'étude que nous avons faite peut aider à mieux connaître les formules Horn étendues. Des idées de ce types pourront sans doute être utilisées pour trouver un algorithme polynomial de reconnaissance des formules Horn étendues.

Étudier la classe Horn étendue, ainsi que les classes apparentées, nous a donné l'idée de généraliser les notions d'arbre et de chemins qui sont à la base des définitions de ces classes. Cela nous a permis de mettre à jour la classe des formules ordonnées, que nous présentons dans la partie III.

Troisième partie

Formules ordonnées et presque ordonnées

Chapitre 1

Formules ordonnées

Sommaire

1.1 Présentation

Nous présentons la classe des formules ordonnées. Il s'agit d'une extension de la classe des formules de Horn étendant les formules Horn élargies simples pour laquelle la résolution unitaire suffit, de la même façon que pour les formules de Horn, à déterminer la satisfaisabilité. On peut générer les solutions de telles formules avec un délai $O(nN)$ (où N est la longueur totale de la formule et n son nombre de variables). Nous présentons un algorithme de complexité $O(nN)$ permettant de tester si une formule appartient ou non à cette classe. Renommer (i.e. inverser le signe de toutes les occurrences) certaines variables ne change pas la satisfaisabilité d'une formule. Nous allons ici étudier la classe des formules ordonnées-renommables, c'est à dire telles qu'en renommant certaines variables on obtient une formule ordonnée. Les résultats obtenus sur les formules ordonnées s'appliquent aux formules ordonnées-renommables, donc on peut résoudre le problème de satisfaisabilité en temps $O(N)$ et générer toutes les solutions avec un délai $O(nN)$. Nous présentons dans ce chapitre un algorithme de complexité $O(nN)$ pour la reconnaissance des formules ordonnées-renommables.

1.2 Définitions

La définition des formules ordonnées étend naturellement celle des formules de Horn. On rappelle qu'une formule de Horn est une formule dont chaque clause contient au plus un littéral positif. Une formule $\mathcal{F}$ sera appelée ordonnée si chacune de ses clauses contient au plus un littéral positif libre. La définition suivante introduit la notion de littéral libre.

Définition 30 (littéral libre/lié) *Soit $C \in \mathcal{F}$ et $l \in C$. On dit que l est* lié *dans C (par rapport à $\mathcal{F}$), si on a $Occ(\overline{l}) = \emptyset$ ou s'il existe $t \in C$ $(t \neq l)$ tel que $Occ(\overline{l}) \subseteq Occ(\overline{t})$. Si l n'est pas lié dans C, alors on dit que l est* libre *dans C.*

Exemple : Soit $\mathcal{F}_1 = \{C_1, C_2, C_3, C_4\}$ avec $C_1 = \{x_1, x_2, x_3, x_4\}$, $C_2 = \{\neg x_1, x_2, \neg x_3\}$, $C_3 = \{\neg x_1, \neg x_2, \neg x_4, x_5\}$, $C_4 = \{\neg x_1, x_3, x_4, \neg x_5\}$. x_2 est lié dans C_1 par rapport à $\mathcal{F}_1$, puisque $Occ(\neg x_2) \subseteq Occ(\neg x_1)$, mais le littéral x_1 est libre dans C_1 par rapport à $\mathcal{F}_1$, puisque pour tout littéral l appartenant à C_1, $Occ_{\mathcal{F}_1}(\neg x_1) \not\subseteq Occ_{\mathcal{F}_1}(\overline{l})$.

La proposition suivante généralise le fait que si toute clause de $\mathcal{F}$ contient un littéral négatif, alors $\mathcal{F}$ est satisfaisable. On rappelle que pour toute clause C sur V, on appelle $neg(C)$ l'ensemble $\{x \in V \mid \neg x \in C\}$.

Proposition 69 *Si toute clause $C \in \mathcal{F}$ contient un littéral négatif ou un littéral lié dans C, alors $\mathcal{F}$ est satisfaisable.*

Preuve : Soit $V = \{x_1, \ldots, x_n\}$, on définit un ordre partiel sur V par : $x_i \leq x_j$ ssi $Occ(\neg x_i) \subsetneq Occ(\neg x_j)$ ou $(Occ(\neg x_i) = Occ(\neg x_j)$ et $i \leq j)$. Soit $W = \{y \in V \mid$ il existe $C \in \mathcal{F}$ tel que $neg(C) = \emptyset$, $y \in C$, y est lié dans C et y est minimal dans C pour l'ordre $\leq\}$ et $M = W \cup \{\neg y \mid y \in V \setminus W\}$. Prouvons que M est un modèle de $\mathcal{F}$. Soit $C \in \mathcal{F}$. Si $neg(C) = \emptyset$, alors par hypothèse, C contient un littéral lié dans C et par définition, $C \cap W \neq \emptyset$, donc $C \cap M \neq \emptyset$. Supposons que $neg(C) \neq \emptyset$ et soit y un élément maximal de $neg(C)$. Nous allons montrer que $y \notin W$. Supposons que $y \in W$ et soit $C' \in \mathcal{F}$ une clause telle que $y \in C'$, $neg(C') = \emptyset$, y est lié et minimal dans C'. On a $Occ(\neg y) \neq \emptyset$. Donc il existe $z \in C'$ $(z \neq y)$ tel que $Occ(\neg y) \subseteq Occ(\neg z)$ et $y \leq z$. Nous avons $\neg z \in C$ et donc y n'est pas maximal dans $neg(C)$, contradiction. Finalement, $y \notin W$, $\neg y \in M$ et $C \cap M \neq \emptyset$. □

Exemple : Toute clause C de la formule $\mathcal{F}_1$ définie ci-dessus contient un littéral négatif ou un littéral positif lié. On a $x_2 \leq x_4 \leq x_1$, $x_3 \leq x_1$, $W = \{x_2, x_3\}$ et $M = \{\neg x_1, x_2, x_3, \neg x_4, \neg x_5\}$.

Nous introduisons maintenant la notion de formule ordonnée. Cette définition généralise celle des formules de Horn. On rappelle qu'une formule est dite de Horn, si chacune de ses clauses contient au plus un littéral positif.

Définition 31 (formule ordonnée) *Une formule $\mathcal{F}$ est* ordonnée *si chaque clause $C \in \mathcal{F}$ contient au plus un littéral positif libre dans C.*

Exemple : $\mathcal{F}_1$ est ordonnée car x_1 est le seul littéral positif libre dans C_1 ($Occ(\neg x_2) \subseteq Occ(\neg x_1)$, $Occ(\neg x_3) \subseteq Occ(\neg x_1)$ et $Occ(\neg x_4) \subseteq Occ(\neg x_1)$) et x_3 est le seul littéral positif libre dans C_4 ($Occ(\neg x_4) \subseteq Occ(x_5)$).

Remarque 3 *Toute formule de Horn est ordonnée.*

Remarque 4 *Si les clauses de $\mathcal{F}$ ne contiennent que des littéraux positifs, alors $\mathcal{F}$ est ordonnée.*

Remarque 5 *Toute formule Horn élargie simple est ordonnée et par conséquent toute formule Horn étendue simple est aussi ordonnée.*

1.3 Satisfaisabilité et génération à délai polynômial

On sait que si $\mathcal{F}$ est une formule de Horn et qu'elle ne contient pas de clause unitaire positive, alors $\mathcal{F}$ est satisfaisable. Cette propriété reste vraie si $\mathcal{F}$ est ordonnée.

Une clause unitaire C est dite positive si $C = \{x\}$ avec $x \in V$. On sait que si $\mathcal{F}$ est une formule de Horn, et ne contient pas de clause unitaire positive, alors $\mathcal{F}$ est satisfaisable. Cette propriété reste vraie si $\mathcal{F}$ est ordonnée.

Proposition 70 *Si $\mathcal{F}$ est ordonnée et ne contient pas de clause unitaire positive $C = \{x\}$, telle que x est libre dans C, alors $\mathcal{F}$ est satisfaisable.*

Preuve : Conséquence immédiate de la Prop. 69. □

Il est facile de voir que si $\mathcal{F}$ est une formule de Horn, alors pour tout ensemble fini $\mathcal{U}$ de clauses unitaires, $Noyau(\mathcal{F} \cup \mathcal{U})$ est une formule de Horn. Nous montrons ici que cette propriété est conservée pour les formules ordonnées.

Proposition 71 *Soit $\mathcal{U}$ un ensemble fini de clauses unitaires sur V, si $\mathcal{F}$ est ordonnée, alors $Noyau(\mathcal{F} \cup \mathcal{U})$ est ordonnée.*

Preuve : Soit $\mathcal{F}' = Noyau(\mathcal{F} \cup \mathcal{U})$ et $C' \in \mathcal{F}'$. Il existe $C \in \mathcal{F}$ tel que $C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$, et $C' = C \setminus \overline{Unit(\mathcal{F} \cup \mathcal{U})}$. Soit $l \in C'$. Supposons que l est lié dans C par rapport à $\mathcal{F}$. On prouve que l est lié dans C' par rapport à $\mathcal{F}'$. Supposons que $Occ_{\mathcal{F}'}(\bar{l}) \neq \emptyset$. Alors $Occ_{\mathcal{F}}(\bar{l}) \neq \emptyset$ et il existe $t \in C$ $(t \neq l)$ tel que $Occ_{\mathcal{F}}(\bar{l}) \subseteq Occ_{\mathcal{F}}(\bar{t})$. On a $C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$, donc $t \notin Unit(\mathcal{F} \cup \mathcal{U})$ et $\bar{t} \notin Unit(\mathcal{F} \cup \mathcal{U})$. De plus, $t \notin \overline{Unit(\mathcal{F} \cup \mathcal{U})}$, car sinon $\bar{t} \in Unit(\mathcal{F} \cup \mathcal{U})$, $Occ_{\mathcal{F}'}(\bar{t}) = \emptyset$ et $Occ_{\mathcal{F}'}(\bar{l}) = \emptyset$. D'où $t \in C'$, $Occ_{\mathcal{F}'}(\bar{l}) \subseteq Occ_{\mathcal{F}'}(\bar{t})$, et l est lié dans C' par rapport à $\mathcal{F}'$. Par hypothèse, C contient au plus un littéral positif libre dans C, donc C' contient au plus un littéral positif libre dans C'. □

Exemple : Soit $\mathcal{U} = \{\{\neg x_4\}, \{x_5\}\}$, étudions la formule $\mathcal{F}_1 \cup \mathcal{U}$. $Noyau(\mathcal{F}_1 \cup \mathcal{U}) = \{\{x_1, x_2, x_3\}, \{\neg x_1, x_2, \neg x_3\}, \{\neg x_1, x_3\}\}$. On peut vérifier que $Noyau(\mathcal{F}_1 \cup \mathcal{U})$ est une formule ordonnée : x_1 est le seul littéral positif libre dans la première clause et les deux autres clauses vérifient trivialement la définition car elles ne contiennent qu'un seul littéral positif. Puisque cette formule est ordonnée et ne contient pas de clause unitaire, elle est satisfaisable. Donc, $\mathcal{F}_1 \cup \mathcal{U}$ est satisfaisable ssi $Unit(\mathcal{F}_1 \cup \mathcal{U})$ est cohérent. $Unit(\mathcal{F}_1 \cup \mathcal{U}) = \{\neg x_4, x_5\}$, est un ensemble cohérent,donc $\mathcal{F}_1 \cup \mathcal{U}$ est satisfaisable.

Proposition 72 *On peut tester en temps $O(N)$ si une formule ordonnée est satisfaisable.*

Preuve : Conséquence des Prop. 70 et 71. □

Les propositions Prop. 70 et Prop. 71 impliquent que les formules ordonnées vérifient les propriétés P1 et P2. Leurs modèles peuvent donc être générés avec un délai polynômial (Partie I Chap. 2 Prop. 3 et Corollaire 4).

1.4 Algorithme de reconnaissance

Nous allons montrer qu'il est possible de tester si un formule est ordonnée avec un algorithme polynômial. Ce résultat n'est pas trivial, puisqu'il existe des classes de formules pour lesquelles on connaît un algorithme polynômial qui résout SAT et que l'on ne sait pas reconnaître, c'est le cas par exemple des formules Horn étendues (Partie II de cette thèse)

Proposition 73 *On peut décider en temps $O(nN)$ si une formule $\mathcal{F}$ est ordonnée.*

Preuve : Soit $Lit(V) = \{l_1, \ldots, l_{2n}\}$. On construit un tableau à deux dimensions $M[i,j]$ $(i,j \in \{1,\ldots,2n\})$ tel que $M[i,j] = 1$ si $Occ(l_i) \subseteq Occ(l_j)$ et $M[i,j] = 0$ sinon. Cette construction peut être réalisée par la procédure Inclusion (Fig. 1.1). On va montrer que sa complexité est $O(nN)$. Les étapes 1 et 2 peuvent être exécutées en un temps $O(\mid Occ(l_i) \mid + \mid Occ(l_j) \mid)$ en utilisant un tableau auxiliaire de booléens qui est indexé sur l'ensemble $\{1,\ldots,2n\}$. Le coût de la procédure Inclusion est proportionnel à $S = \Sigma_{i=1}^{2n}\Sigma_{j=1}^{i}(\mid Occ(l_i) \mid + \mid Occ(l_j) \mid)$. On a $S = S_1 + S_2$ avec $S_1 = \Sigma_{i=1}^{2n}(\mid Occ(l_1) + \ldots + \mid Occ(l_i) \mid)$ et $S_2 = \Sigma_{i=1}^{2n}(i. \mid Occ(l_i) \mid)$. Maintenant, $S_1 \leq 2nN$ et $S_2 \leq \Sigma_{i=1}^{2n}(i. \mid Occ(l_i) \mid) = \Sigma_{i=1}^{2n}\Sigma_{k=i}^{2n} \mid Occ(l_k) \mid \leq \Sigma_{i=1}^{2n} N = 2nN$. On en déduit que la complexité de Inclusion est $O(nN)$. Les ensembles $Occ(l_i)$ $(1 \leq i \leq 2n)$ donnés en entrée à la procédure Inclusion peuvent être calculés en temps $O(N)$. La complexité totale de la construction de M est donc $O(nN)$. Maintenant, soit $C \in \mathcal{F}$ et $l \in C$. Une fois les ensembles $Occ(l_i)$ $(1 \leq i \leq 2n)$ calculés, on peut tester que $Occ(\overline{l}) \neq \emptyset$ en un temps constant pour tout littéral l. En utilisant M, on peut déterminer si l est libre dans C en temps $O(\mid C \mid)$. D'où on déduit que l'on peut déterminer si une clause $C \in \mathcal{F}$ contient au plus un littéral positif libre en temps $O(n \mid C \mid)$. Finalement, on peut décider en temps $O(nN)$ si $\mathcal{F}$ est ordonnée. □

Procédure *Inclusion*
Entrée : Les ensembles $Occ(l_i)$ $(1 \leq i \leq 2n)$;
Sortie : Un tableau à deux dimensions $M[i,j]$ $(i,j \in \{1,\ldots,2n\})$ tel que $M[i,j] = 1$ si $Occ(l_i) \subseteq Occ(l_j)$ et $M[i,j] = 0$ sinon.
début
  pour $i = 2$ **jusqu'à** $2n$ **faire**
    pour $j = 1$ **jusqu'à** i **faire**
      début
(1) **si** $Occ(l_i) \subseteq Occ(l_j)$ **alors** $M[i,j] \leftarrow 1$ **sinon** $M[i,j] \leftarrow 0$;
(2) **si** $Occ(l_j) \subseteq Occ(l_i)$ **alors** $M[j,i] \leftarrow 1$ **sinon** $M[j,i] \leftarrow 0$;
      fin ;
fin

FIG. 1.1 – *Procédure Inclusion*

1.5 Formules ordonnées-renommables

Soit x une variable apparaissant dans $\mathcal{F}$. On rappelle que renommer une variable consiste à remplacer toutes les occurrences de x par $\neg x$ et toutes les occurrences de $\neg x$ par x. Nous allons présenter un ensemble de résultats portant sur la possibilité de renommer certaines variables

d'une formule quelconque pour la transformer en une formule ordonnée. Nous proposons un algorithme polynômial $O(nN)$ (où n est le nombre de variables et N la longueur totale de la formule) permettant de tester si une formule est ordonnée-renommable et de donner l'ensemble des variables qu'il faut renommer pour obtenir une formule ordonnée. On prouve que le renommage conserve les propriétés P1 et P2. Donc, si $\mathcal{F}$ est obtenue en renommant certaines variables d'une formule ordonnée, alors on peut générer les modèles de $\mathcal{F}$ avec un délai polynômial.

Définition 32 (formule ordonnée-renommable) *$\mathcal{F}$ est* ordonnée-renommable *si on peut la transformer en une formule ordonnée en renommant certaines de ses variables.*

Exemple : Soit $\mathcal{F}_2 = \{C'_1, C'_2, C'_3, C'_4\}$, avec $C'_1 = \{\neg x_1, x_2, \neg x_3, x_4\}$, $C'_2 = \{x_1, x_2, x_3\}$, $C'_3 = \{x_1, \neg x_2, \neg x_4, x_5\}$, $C'_4 = \{x_1, \neg x_3, x_4, \neg x_5\}$. $\mathcal{F}_2$ est une formule ordonnée-renommable, puisque $\mathcal{F}_2$ est obtenue en renommant les variables x_1 et x_3 dans la formule $\mathcal{F}_1$.

On vérifie dans un premier temps que la classe des formules ordonnées-renommables vérifie la propriété P1.

Proposition 74 *Si $\mathcal{F}$ est ordonnée-renommable, alors pour tout ensemble de littéraux $\mathcal{U}$, la formule $Noyau(\mathcal{F} \cup \mathcal{U})$ est ordonnée-renommable.*

Preuve : On rappelle qu'un renommage est un ensemble de littéraux qui est cohérent et complet (cf. définitions Partie I, Sec. 1.2).

Comme $\mathcal{F}$ est ordonnée-renommable, il existe un renommage R tel que $R(\mathcal{F})$ est ordonnée. Soit $\mathcal{U}' = R(\mathcal{U})$, on a donc $Noyau(R(\mathcal{F}) \cup \mathcal{U}') = Noyau(R(\mathcal{F} \cup \mathcal{U}))$ est ordonnée (Prop. 71). D'où $Noyau(\mathcal{F} \cup \mathcal{U})$ est ordonnée-renommable. □

Nous vérifions maintenant que la classe des formules ordonnées-renommables vérifie aussi P2.

Proposition 75 *Si $\mathcal{F}$ est ordonnée-renommable et toute clause $C \in \mathcal{F}$ est de longueur supérieure ou égale à deux, alors $\mathcal{F}$ est satisfaisable.*

Preuve : Évident car le renommage des variables ne change pas la satisfaisabilité d'une formule. □

Comme la classe des formules ordonnées-renommables vérifie les propriétés P1 et P2, les résultats établis dans la Partie I impliquent que l'on peut résoudre efficacement le problème SAT pour de telles formules, et aussi générer efficacement tous leurs modèles.

Corollaire 76 *Si $\mathcal{F}$ est ordonnées-renommable, alors le problème de satisfaisabilité de la formule $\mathcal{F}$ peut être résolu en temps $O(N)$ et les modèles de $\mathcal{F}$ peuvent être générés avec un délai $O(nN)$.*

On montre maintenant que l'on peut reconnaître les formules ordonnées-renommables en temps polynômial de la même manière que l'on peut reconnaître les formules Horn-renommables. Notre algorithme est basé sur les propriétés du graphe orienté qui représente les contraintes que $\mathcal{F}$ doit satisfaire pour être ordonnée-renommable. Ce graphe est similaire au graphe d'implication introduit par Aspvall et al. [2] (Partie I, Chap. 2) pour déterminer en temps linéaire si une formule binaire est satisfaisable.

On rappelle que l'on peut représenter tout renommage comme un ensemble de littéraux R, cohérent et complet pour V, en respectant la convention suivante : la variable x est renommée ssi $\neg x \in R$; on écrit $R(x) = x$ et $R(\neg x) = \neg x$ si $x \in R$ (x n'est pas renommée), et on écrit $R(x) = \neg x$ et $R(\neg x) = x$ si $\neg x \in R$ (x est renommée). On remarque que pour $l \in Lit(V)$, $R(l)$ est positif ssi $l \in R$.

Supposons que R est un ensemble de littéraux qui représente un renommage qui transforme $\mathcal{F}$ en une formule ordonnée. Soient $C \in \mathcal{F}$ et $l, t \in C$. Supposons que l et t sont libres dans C. Alors R ne peut pas contenir à la fois l et t ; plus précisément, si $l \in R$, alors $\overline{t} \in R$. Ceci nous emmène à définir la relation $\overset{\mathcal{F}}{\Longrightarrow}$ sur l'ensemble $Lit(V)$.

Définition 33 ($\overset{\mathcal{F}}{\Longrightarrow}$) *Pour tous $l, t \in Lit(V)$, $l \overset{\mathcal{F}}{\Longrightarrow} t$ ssi il existe $C \in \mathcal{F}$ telle que $l \in C$, $\overline{t} \in C$, $l \neq \overline{t}$ et l et $\overline{t}$ sont tous les deux libres dans C.*

On notera $G(\mathcal{F})$ le graphe associé à la relation $\overset{\mathcal{F}}{\Longrightarrow}$.

Exemple : Construisons le graphe $G(\mathcal{F}_2)$ correspondant à la formule $\mathcal{F}_2$. Dans la première clause, un seul littéral est libre : $\neg x_1$; on n'ajoute donc pas d'arc dans le graphe. Dans la seconde clause, les littéraux x_2 et x_3 sont libres ; on ajoute donc les arcs $x_2 \rightarrow \neg x_3$ et $x_3 \rightarrow \neg x_2$ dans $G(\mathcal{F}_2)$. Dans la troisième clause, les littéraux $\neg x_2$ et $\neg x_4$ sont libres ; on ajoute $\neg x_2 \rightarrow x_4$ et $\neg x_4 \rightarrow x_2$ dans $G(\mathcal{F}_2)$. Dans la dernière clause, x_1 et $\neg x_3$ sont libres, $G(\mathcal{F}_2)$ reçoit donc deux arcs supplémentaires $x_1 \rightarrow x_3$ et $\neg x_3 \rightarrow \neg x_1$. On peut voir le graphe $G(\mathcal{F}_2)$ en Fig. 1.2.

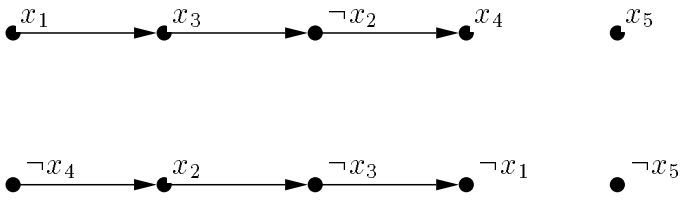

FIG. 1.2 – *Graphe $G(\mathcal{F}_2)$*

On appelle $\overset{\mathcal{F}}{\Longrightarrow}^*$ la clôture transitive et réflexive de $\overset{\mathcal{F}}{\Longrightarrow}$, et pour tout littéral l, $Clos_{\mathcal{F}}(l)$ représente l'ensemble $\{t \mid l \overset{\mathcal{F}}{\Longrightarrow}^* t\}$ (on écrira parfois

plus simplement $Clos(l)$ lorsqu'aucune confusion ne sera possible). Un ensemble de littéraux L est dit $\mathcal{F}$-*clos* si ($l \in L$ et $l \stackrel{\mathcal{F}}{\Longrightarrow} t$) implique que $t \in L$. On remarque que L est $\mathcal{F}$-clos ssi $Clos(l) \subseteq L$ pour tout $l \in L$. On peut observer que $l \stackrel{\mathcal{F}}{\Longrightarrow} t$ ssi $\overline{t} \stackrel{\mathcal{F}}{\Longrightarrow} \overline{l}$, et $l \stackrel{\mathcal{F}}{\Longrightarrow}^* t$ ssi $\overline{t} \stackrel{\mathcal{F}}{\Longrightarrow}^* \overline{l}$ (dualité).

Proposition 77 *$\mathcal{F}$ est ordonnée-renommable ssi il existe $R \subseteq Lit(V)$ tel que R est cohérent, $\mathcal{F}$-clos et complet pour V.*

Preuve : ($\Rightarrow$) Soit $X \subseteq V$ tel que le renommage des variables de X transforme $\mathcal{F}$ en une formule ordonnée $\mathcal{F}'$. Soit $R = \{\neg p \mid p \in V \cap X\} \cup \{p \mid p \in V \setminus X\}$. Par définition R est cohérent et complet. On montre maintenant que R est $\mathcal{F}$-clos. Soient l et t deux littéraux tels que $l \in R$ et $l \stackrel{\mathcal{F}}{\Longrightarrow} t$. Nous devons prouver que $t \in R$. Il existe une clause $C \in \mathcal{F}$ telle que $l, \overline{t} \in C$, $l \neq \overline{t}$ et les deux littéraux l et $\overline{t}$ sont libres dans C par rapport à $\mathcal{F}$. Soit $C' \in \mathcal{F}'$ la clause obtenue en renommant les variables de X dans C. C' contient $R(l)$ et $R(\overline{t})$. $R(l)$ est positif puisque $l \in R$, $R(\overline{t})$ est négatif car $\mathcal{F}'$ est ordonnée et C' contient au plus un littéral positif libre. Par conséquent $\overline{t} \notin R$ et $t \in R$.
($\Leftarrow$) Soit $R \subseteq Lit(V)$ un ensemble cohérent, $\mathcal{F}$-clos et complet pour V. Et soit $\mathcal{F}'$ la formule obtenue en renommant les variables x telles que $\neg x \in R$ dans la formule $\mathcal{F}$. On montre maintenant que $\mathcal{F}'$ est ordonnée. Soit $C' \in \mathcal{F}'$ et C la clause correspondante de $\mathcal{F}$. Supposons que C' contienne un littéral positif libre dans C' par rapport à $\mathcal{F}'$, appelons le l'. Soit $l \in C$ tel que $R(l) = l'$. On a $l \in R$ puisque $R(l)$ est positif. Soit $t' \in C'$ ($t' \neq l'$) un littéral libre dans C' par rapport à $\mathcal{F}'$, et $t \in C$ tel que $R(t) = t'$. On a $t \neq l$ et l et t sont libres dans C par rapport à $\mathcal{F}$. Donc $l \stackrel{\mathcal{F}}{\Longrightarrow} \overline{t}$ et $\overline{t} \in R$ puisque R est $\mathcal{F}$-clos. Finalement, $t \notin R$ et $R(t)$ est négatif. Ceci prouve que $\mathcal{F}'$ est ordonnée. □

Exemple : Pour la formule $\mathcal{F}_2$, le graphe $G(\mathcal{F}_2)$ de la relation $\stackrel{\mathcal{F}_2}{\Longrightarrow}$ est donné en Fig. 1.2. On peut voir que l'ensemble complet et cohérent de littéraux $\{\neg x_1, x_2, \neg x_3, x_4, x_5\}$ est $\mathcal{F}_2$-clos. Il correspond au renommage des variables x_1 et x_3 ; il transforme $\mathcal{F}_2$ en $\mathcal{F}_1$ qui est une formule ordonnée.

Le Lemme 78 sera utilisé dans les preuves des Prop. 79 et 83.

Lemme 78 *Si L est un ensemble de littéraux $\mathcal{F}$-clos et l est un littéral tel que $\overline{l} \notin L$ alors $Clos(l) \cap \overline{L} = \emptyset$.*

Preuve : Soit $t \in Clos(l)$. On a $l \stackrel{\mathcal{F}}{\Longrightarrow}^* t$ et donc $\overline{t} \stackrel{\mathcal{F}}{\Longrightarrow}^* \overline{l}$ par dualité. Si $t \in \overline{L}$ alors $\overline{t} \in L$, et $\overline{l} \in L$ puisque L est $\mathcal{F}$-clos. Contradiction. □

La Prop. 79 est une caractérisation utile des formules ordonnées-renommables, elle va servir à écrire un algorithme de reconnaissance de ces formules.

Proposition 79 *$\mathcal{F}$ est ordonnée-renommable ssi pour tout $x \in V$, $Clos(x)$ est cohérent ou $Clos(\neg x)$ est cohérent.*

Preuve : ($\Rightarrow$) Soit $R \subseteq Lit(V)$ un ensemble cohérent, $\mathcal{F}$-clos et complet pour V (Prop. 77). Soit $x \in V$. Si $x \in R$, alors $Clos(x) \subseteq R$ (R est $\mathcal{F}$-clos) et $Clos(x)$ est cohérent (R est cohérent). Sinon, $\neg x \in R$, (R est complet) et $Clos(\neg x)$ est cohérent.
($\Leftarrow$) Soit $V = \{x_1, \ldots, x_n\}$. On définit les ensembles de littéraux $L_1, \ldots, L_n$ de la façon suivante. Si $Clos(x_1)$ est cohérent, alors $L_1 = Clos(x_1)$, dans le cas contraire, $L_1 = Clos(\neg x_1)$. Pour tout $i \in \{2, \ldots, n\}$, si $x_i \in L_{i-1}$ ou $\neg x_i \in L_{i-1}$, alors $L_i = L_{i-1}$, sinon $L_i = L_{i-1} \cup Clos(x_i)$ si $Clos(x_i)$ est cohérent, et $L_i = L_{i-1} \cup Clos(\neg x_i)$ si $Clos(x_i)$ est incohérent. Par définition L_i ($1 \leq i \leq n$) est $\mathcal{F}$-clos. Prouvons maintenant par récurrence que L_i ($1 \leq i \leq n$) est cohérent. Par définition L_1 est cohérent. Soit $i \in \{2, \ldots, n\}$. Si $x_i \in L_{i-1}$, ou $\neg x_i \in L_{i-1}$, alors $L_i = L_{i-1}$, et L_i est cohérent par hypothèse de récurrence. Supposons que $x_i \notin L_{i-1}$, $\neg x_i \notin L_{i-1}$ et $L_i = L_{i-1} \cup Clos(x_i)$ (la preuve est similaire si $L_i = L_{i-1} \cup Clos(\neg x_i)$). Par hypothèse, $Clos(x_i)$ est cohérent et L_{i-1} est cohérent par hypothèse de récurrence. Le Lemme 78 implique que L_i est cohérent. Finalement L_n est complet pour V, cohérent et $\mathcal{F}$-clos, donc $\mathcal{F}$ est ordonnée-renommable (Prop. 77). □

En utilisant la caractérisation mise en évidence ci-dessus, il est maintenant facile de reconnaître si une formule est ordonnée-renommable.

Proposition 80 *On peut tester en temps $O(nN)$ si une formule $\mathcal{F}$ est ordonnée-renommable.*

Preuve : Pour tout littéral l, $Clos(l)$ est incohérent ssi $\overline{l} \in Clos(l)$. Donc pour tout $x \in V$, les deux ensembles $Clos(x)$ et $Clos(\neg x)$ sont incohérents ssi x et $\neg x$ sont dans la même composante fortement connexe de $G(\mathcal{F})$. La Prop. 79 implique que $\mathcal{F}$ est ordonnée ssi pour tout $x \in V$, x et $\neg x$ ne sont pas dans la même composante fortement connexe de $G(\mathcal{F})$. On peut remarquer que $G(\mathcal{F})$ contient $O(nN)$ arcs. Les composantes fortement connexes de $G(\mathcal{F})$ peuvent être calculées en temps $O(nN)$ en utilisant l'algorithme de Tarjan [47]. On peut tester si une composante fortement connexe de $G(\mathcal{F})$ contient des littéraux complémentaires en temps $O(n)$ en utilisant un tableau de booléens indexé par les éléments de V. Il reste à prouver que $G(\mathcal{F})$ peut être construit en temps $O(nN)$. On utilise la procédure Construit-$G(\mathcal{F})$ (Fig. 1.3). Le pas 2 peut être réalisé en temps $O(nN)$ par la procédure Inclusion (Fig. 1.1 et preuve de

Prop. 73). Les pas 3 et 4 prennent un temps $O(\Sigma(\mid C_i \mid^2)$ (pour le pas 3, voir la preuve de la Prop. 73) et $O(\Sigma(\mid C_i \mid^2) \leq O(nN)$. Ceci implique le résultat désiré. □

Procédure *Construit-G*($\mathcal{F}$)
début
1. $A \leftarrow \emptyset$; {A représente l'ensemble des arcs de $G(\mathcal{F})$.}
2. Construire un tableau à deux dimensions $M[i,j]$ tel que
$M[i,j] = 1$ si $Occ(l_i) \subseteq Occ(l_j)$ et $M[i,j] = 0$ sinon.
3. Pour tout $C \in \mathcal{F}$ calculer, à l'aide de M, l'ensemble $Libre(C)$
des littéraux libres dans C.
4. Pour tout $C \in \mathcal{F}$ et tout $\{l,t\} \subseteq Libre(C)$, $A \leftarrow A \cup \{(l,\overline{t}),(t,\overline{l})\}$;
fin

FIG. 1.3 – *Procédure Construit-G*($\mathcal{F}$)

Les classes ordonnées et ordonnées-renommables sont donc deux nouvelles classes étendant Horn, dont la reconnaissance est polynômiale et pour lesquelles il existe un algorithme de génération à délai polynômial.

Chapitre 2

Formules presque ordonnées

Sommaire

2.1 Présentation

Dans ce chapitre, nous étendons la classe des formules ordonnées de telle façon que les modèles des nouvelles formules puissent toujours être générés avec un délai polynômial. Nous adaptons les résultats obtenus sur la notion de base de Horn présentée par Hébrard et Luquet [32] et sur les formules presque Horn (Partie I Chap. 3) aux formules ordonnées. Nous présentons la classe des formules presque ordonnées, qui malheureusement, tout comme la classe des formules presque Horn, ne vérifie pas la propriété P1. On rappelle que la classe $\mathcal{C}$ vérifie P1 quand pour tout ensemble de clauses unitaires $\mathcal{U}$, si $\mathcal{F} \in \mathcal{C}$ alors $Noyau(\mathcal{F} \cup \mathcal{U}) \in \mathcal{C}$.

De la même manière que pour les formules presque Horn, nous présentons un ordre sur les variables qui permet d'être sûr que pour tout ensemble de clauses unitaires $\mathcal{U}$ utilisé dans l'algorithme de génération des solutions (Partie I Fig. 1.1) on a $Noyau(\mathcal{F} \cup \mathcal{U})$ est presque ordonnée si $\mathcal{F}$ est presque ordonnée. On peut donc générer avec un délai $O(nN)$ tous les modèles de telles formules

Nous donnons de plus un algorithme de complexité $O(n^2N)$ pour la reconnaissance des formules presque ordonnées.

Soit $X \subseteq V$ et $C \in \mathcal{F}$. On rappelle qu'une clause C est dite clause sur X si $C \subseteq Lit(X)$.

Définition 34 (formule X-ordonnée, formule X-ordonnée-renommable) *Soit $X \subseteq V$. $\mathcal{F}$ est* X-ordonnée *si pour toute clause $C \in \mathcal{F}$ et tout littéral*

positif $x \in X$, on a : si x est libre dans C, alors $C \subseteq Lit(X)$ et x est le seul littéral positif libre dans C.

$\mathcal{F}$ est X-ordonnée-renommable *si on peut transformer $\mathcal{F}$ en une formule X-ordonnée en renommant certaines de ses variables.*

Exemple : Soit $\mathcal{G}_1$ la formule $\{\{x_1, x_2\}, \{x_3, \neg x_4, x_5\}, \{\neg x_3, \neg x_4, x_5\}, \{\neg x_1, \neg x_2, \neg x_3, x_6, x_7\}, \{\neg x_3, x_4, \neg x_5, x_6, \neg x_7\}, \{\neg x_1, \neg x_3, \neg x_5, \neg x_6, x_7\}, \{\neg x_1, \neg x_2, x_6, \neg x_7\}, \{x_6, x_7\}, \{\neg x_6, \neg x_7\}\}$. $\mathcal{G}_1$ est $\{x_3, x_4, x_5\}$-ordonnée, on peut vérifier que les clauses contenant un littéral positif libre sur $\{x_3, x_4, x_5\}$ sont $\{x_3, \neg x_4, x_5\}$ et $\{\neg x_3, \neg x_4, x_5\}$; elles ne contiennent que des littéraux appartenant à $Lit(\{x_3, x_4, x_5\})$ (la clause $\{\neg x_3, x_4, \neg x_5, x_6, \neg x_7\}$ contient le littéral positif x_4, mais x_4 n'est pas libre car $Occ(\neg x_4) \subseteq Occ(x_5)$ et $\neg x_5$ appartient à cette clause).

On peut remarquer que les clauses d'une formule X-ordonnée peuvent être de trois types :
• clauses sur X contenant au plus un littéral positif libre.
• clauses sur V contenant un ou des littéraux négatifs ou liés de $Lit(X)$ ainsi que un ou des littéraux de $Lit(V \setminus X)$, mais ne contenant aucun littéral positif libre de $Lit(X)$.
• clauses sur l'ensemble $V \setminus X$.

On remarque que $\mathcal{F}$ est ordonnée ssi $\mathcal{F}$ est V-ordonnée, et que $\mathcal{F}$ est ordonnée-renommable ssi elle est V-ordonnée-renommable. On montre que si $\mathcal{F}$ est X-ordonnée-renommable avec $X \neq \emptyset$, et $\mathcal{F}$ ne contient pas de clause unitaire, alors décider si $\mathcal{F}$ est satisfaisable se ramène à décider si un sous-ensemble strict de $\mathcal{F}$ est satisfaisable.

Notation 3 *Soit $X \subseteq V$. On note $Reste(\mathcal{F}, X)$ l'ensemble $\{C \in \mathcal{F} \mid C \cap Lit(X) = \emptyset\}$.*

Proposition 81 *Si $\mathcal{F}$ est X-ordonnée-renommable et ne contient pas de clause unitaire positive, alors $\mathcal{F}$ est satisfaisable ssi $Reste(\mathcal{F}, X)$ est satisfaisable.*

Preuve :
($\Rightarrow$) Immédiat.
($\Leftarrow$) *Cas particulier :* $\mathcal{F}$ est X-ordonnée. Soit $M \subseteq Lit(V \setminus X)$ un modèle de $Reste(\mathcal{F}, X)$. Soit $\mathcal{F}' = \{C' \subseteq Lit(X) \mid$ il existe $C \in \mathcal{F} \setminus Reste(\mathcal{F}, X)$ telle que $C \cap M = \emptyset$ et $C' = C \setminus \overline{M}\}$. Soit $C' \in \mathcal{F}'$ et $C \in \mathcal{F} \setminus Reste(\mathcal{F}, X)$ tel que $C \cap M = \emptyset$ et $C' = C \setminus \overline{M}$. Par hypothèse, C contient un littéral $l \in Lit(X)$ tel que l est négatif ou l est lié dans C par rapport à $\mathcal{F}$. On a $l \in C'$ car $\overline{M} \subseteq Lit(V \setminus X)$. Supposons que l est lié dans C par rapport à $\mathcal{F}$. On va prouver que l est lié dans C' par rapport à $\mathcal{F}'$. Supposons que $Occ_{\mathcal{F}'}(\overline{l}) \neq \emptyset$. Alors $Occ_{\mathcal{F}}(\overline{l}) \neq \emptyset$ et il existe $t \in C$ $(t \neq l)$ tel que $Occ_{\mathcal{F}}(\overline{l}) \subseteq Occ_{\mathcal{F}}(\overline{t})$. On a $C \cap M = \emptyset$, donc $t \notin M$ et $\overline{t} \notin \overline{M}$. De plus,

$t \notin \overline{M}$, puisque dans le cas contraire $\overline{t} \in M$, $Occ_{\mathcal{F}'}(\overline{t}) = \emptyset$ et $Occ_{\mathcal{F}'}(\overline{l}) = \emptyset$ D'où $t \in C'$, $Occ_{\mathcal{F}'}(\overline{l}) \subseteq Occ_{\mathcal{F}'}(\overline{t})$, et l est lié dans C' par rapport à $\mathcal{F}'$. Ceci montre que toute clause $C' \in \mathcal{F}'$ contient un littéral négatif ou un littéral lié dans C'.

Donc $\mathcal{F}'$ est satisfaisable (Prop. 69). Soit M' un modèle de $\mathcal{F}'$. $M \cup M'$ est cohérent, car $var(Reste(\mathcal{F}, X)) \cap var(\mathcal{F}') = \emptyset$. Soit $C \in \mathcal{F}$. Ou bien $C \cap M \neq \emptyset$, ou bien il existe $C' \in \mathcal{F}'$ telle que $C' \subseteq C$, $C' \cap M' \neq \emptyset$ et $C \cap M' \neq \emptyset$. Donc $\mathcal{F}$ est satisfaisable.

Cas Général: $\mathcal{F}$ est X-ordonnée-renommable. Se déduit directement du cas précédent, car le renommage préserve la satisfaisabilité. □

La relation $\stackrel{\mathcal{F}}{\Longrightarrow}$ peut être utilisée pour caractériser le fait que $\mathcal{F}$ est X-ordonnée-renommable.

Proposition 82 *$\mathcal{F}$ est X-ordonnée-renommable ssi il existe un ensemble de littéraux $R \subseteq Lit(X)$ tel que R est cohérent, $\mathcal{F}$-clos et complet pour X.*

Preuve : La preuve de cette proposition est pratiquement la même que celle de la Prop. 77. Il suffit de remplacer V par X. □

Exemple : On peut voir (Fig. 2.1) que l'ensemble des noeuds $\{x_3, \neg x_4, x_5\}$ est $\mathcal{G}_1$-clos. On voit aussi que l'ensemble $\{x_1, \neg x_2\}$ est cohérent et $\mathcal{G}_1$-clos, donc $\mathcal{G}_1$ est aussi $\{x_1, x_2\}$-ordonnée-renommable.

Proposition 83 *Soit $X_1 \subseteq V$ et $X_2 \subseteq V$. Si $\mathcal{F}$ est X_1-ordonnée-renommable et X_2-ordonnée-renommable alors $\mathcal{F}$ est $(X_1 \cup X_2)$-ordonnée-renommable.*

Preuve : Grâce à la Prop. 82, on obtient qu'il existe $R_1 \subseteq Lit(X_1)$ et $R_2 \subseteq Lit(X_2)$ tels que R_1 et R_2 sont cohérents, $\mathcal{F}$-clos et complets respectivement pour les ensembles X_1 et X_2. L'ensemble $R = R_1 \cup (R_2 \setminus \overline{R_1})$ est cohérent et complet pour $X_1 \cup X_2$. On prouve maintenant que R est $\mathcal{F}$-clos. Soit $l \in R$. Si $l \in R_1$ alors $Clos(l) \subseteq R_1 \subseteq R$ car R_1 est $\mathcal{F}$-clos. Si $l \in R_2 \setminus \overline{R_1}$ alors $Clos(l) \subseteq R_2$. Le lemme 78 avec $L = R_1$ montre que $Clos(l) \subseteq (R_2 \setminus \overline{R_1}) \subseteq R$. La conclusion est une conséquence de la Prop. 82. □

On cherche maintenant à caractériser le plus grand ensemble X tel que $\mathcal{F}$ soit X-ordonnée-renommable. Celui-ci est unique, grâce à la proposition ci-dessus.

Définition 35 (base ordonnée, OReste($\mathcal{F}$)) *Soient $X_1, \ldots, X_k$ tous les sous-ensembles de V tels que $\mathcal{F}$ est X_i-ordonnée-renommable et $B = X_1 \cup \ldots \cup X_k$. La Prop. 83 implique que $\mathcal{F}$ est B-ordonnée-renommable.*

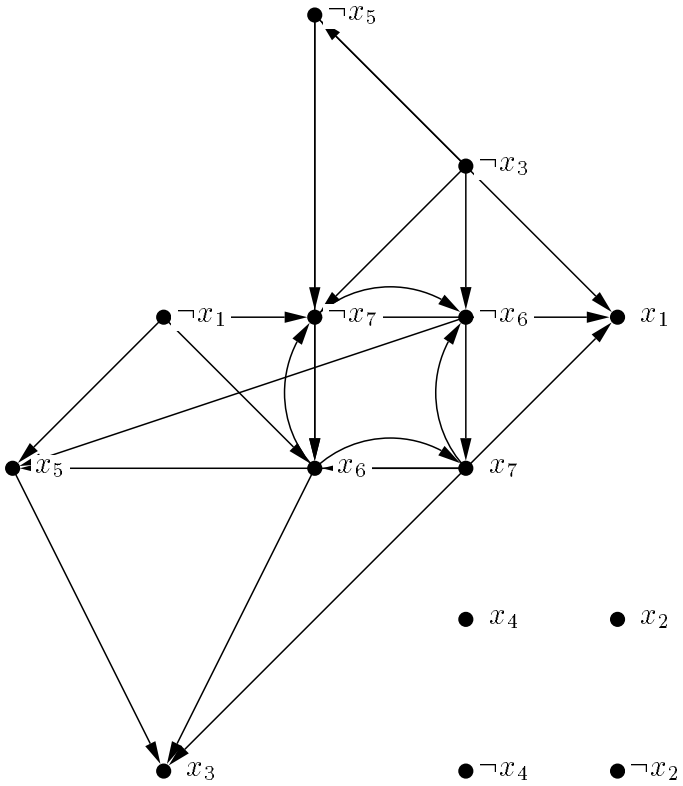

FIG. 2.1 – *Graphe* $G(\mathcal{G}_1)$

L'ensemble B sera appelé la base ordonnée de $\mathcal{F}$ *et notée* $OBase(\mathcal{F})$. *On utilisera la notation* $OReste(\mathcal{F})$ *pour représenter la formule* $Reste(\mathcal{F}, OBase(\mathcal{F}))$.

Exemple : L'ensemble $\{x_1, x_2, x_3, x_4, x_5\}$ est cohérent et $\mathcal{G}_1$-clos ; par contre les littéraux $x_6, \neg x_6, x_7, \neg x_7$ appartiennent à une seule composante fortement connexe de $G(\mathcal{G}_1)$ (voir Fig. 2.1) il n'existe donc pas d'ensemble cohérent et $\mathcal{G}_1$-clos contenant un de ces littéraux, ils n'appartiennent donc pas à la base ordonnée de $G(\mathcal{G}_1)$. Donc $OBase(\mathcal{G}_1) = \{x_1, x_2, x_3, x_4, x_5\}$ et la formule $\mathcal{G}_1$ est $\{x_1, x_2, x_3, x_4, x_5\}$-ordonnée-renommable (et même $\{x_1, x_2, x_3, x_4, x_5\}$-ordonnée).

Corollaire 84 *Si $\mathcal{F}$ ne contient pas de clause unitaire, alors $\mathcal{F}$ est satisfaisable ssi $OReste(\mathcal{F})$ est satisfaisable.*

On remarque donc que si $\mathcal{F}$ ne contient pas de clause unitaire et que $OReste(\mathcal{F}) = \emptyset$, alors $\mathcal{F}$ est satisfaisable.

Nous présentons une définition alternative de la base ordonnée qui utilise la relation $\overset{\mathcal{F}}{\Rightarrow}$.

Proposition 85 $OBase(\mathcal{F}) = \{x \in V \mid Clos(x)$ *est cohérent ou* $Clos(\neg x)$ *est cohérent*$\}$.

Preuve : Soit $x \in V$ et $R \subseteq Lit(OBase(\mathcal{F}))$ tel que R est cohérent, $\mathcal{F}$-clos et complet pour $OBase(\mathcal{F})$ (Prop. 82). Si $x \in OBase(\mathcal{F})$, alors x ou $\neg x$ appartient à R, donc $Clos(x)$ ou $Clos(\neg x)$ est cohérent. Supposons maintenant que $Clos(x)$ soit cohérent (l'autre cas est similaire). Soit $X = var(Clos(x))$. La Prop. 82 implique que $\mathcal{F}$ est X-ordonnée-renommable. Donc, $X \subseteq OBase(\mathcal{F})$ et $x \in OBase(\mathcal{F})$. □

Cette caractérisation de la base ordonnée d'une formule nous permet de déduire de nouvelles propriétés pour les bases ordonnées et les restes ordonnés.

Proposition 86 *Si $\mathcal{F}' \subseteq \mathcal{F}$ alors $OReste(\mathcal{F}') \subseteq OReste(\mathcal{F})$.*

Preuve : On observe que pour tous $l, t \in Lit(var(\mathcal{F}'))$, si $Occ_{\mathcal{F}}(l) \subseteq Occ_{\mathcal{F}}(t)$, alors $Occ_{\mathcal{F}'}(l) \subseteq Occ_{\mathcal{F}'}(t)$. Donc tout arc de $G(\mathcal{F}')$ est un arc de $G(\mathcal{F})$. Donc pour tout littéral $l \in Lit(var(\mathcal{F}'))$, si $Clos_{\mathcal{F}}(l)$ est cohérent, alors $Clos_{\mathcal{F}'}(l)$ est cohérent. D'où $OBase(\mathcal{F}) \cap var(\mathcal{F}') \subseteq OBase(\mathcal{F}')$. Soit $C \in OReste(\mathcal{F}')$. On a $C \cap Lit(OBase(\mathcal{F})) \subseteq C \cap Lit(OBase(\mathcal{F}'))$. Par définition du reste ordonné $C \cap Lit(OBase(\mathcal{F}')) = \emptyset$, donc $C \cap Lit(OBase(\mathcal{F})) = \emptyset$ et $C \in OReste(\mathcal{F})$. □

On peut remarquer que la base ordonnée de $OReste(\mathcal{F})$ peut être non vide, dans ce cas, $OReste(OReste(\mathcal{F}))$ est strictement inclus dans $OReste(\mathcal{F})$. Il est donc possible de répéter ce processus tant que l'on n'a pas obtenu une formule avec une base ordonnée vide.

Définition 36 (formule presque ordonnée) *Soit* OReste-itéré($\mathcal{F}$) *le sous-ensemble de $\mathcal{F}$ défini récursivement : si $OBase(\mathcal{F}) = \emptyset$ alors* OReste-itéré($\mathcal{F}$) $= \mathcal{F}$ *sinon* OReste-itéré($\mathcal{F}$) $=$ OReste-itéré($OReste(\mathcal{F})$). *Une formule $\mathcal{F}$ appartient à la classe* presque ordonnée, *si OReste-itéré($\mathcal{F}$) $= \emptyset$.*

Exemple : $OReste(\mathcal{G}_1) = \{\{x_6, x_7\}, \{\neg x_6, \neg x_7\}\}$ (puisque $OBase(\mathcal{G}_1) = \{x_1, x_2, x_3, x_4, x_5\}$). $OReste(\mathcal{G}_1)$ est Horn-renommable (on renomme x_7 par exemple) donc ordonnée-renommable. Ceci implique que, $OReste(OReste(\mathcal{G}_1)) = \emptyset$. Donc, $\mathcal{G}_1$ est une formule presque ordonnée.

Corollaire 87 *Si $\mathcal{F}$ est presque ordonnée et ne contient pas de clause unitaire, alors $\mathcal{F}$ est satisfaisable.*

2.2 Génération à délai polynômial

On va maintenant prouver que, même si la classe presque ordonnée ne vérifie pas P1, on peut tout de même générer tous les modèles de telles formules avec un délai $O(nN)$.

Proposition 88 *Si $\mathcal{F}' \subseteq \mathcal{F}$ et $\mathcal{F}$ est presque ordonnée, alors $\mathcal{F}'$ est presque ordonnée.*

Preuve : Si $OReste\text{-}itéré(\mathcal{F}) = \emptyset$, alors la Prop. 86 implique que $OReste\text{-}itéré(\mathcal{F}') = \emptyset$. □

La proposition suivante donne une caractérisation des formules presque ordonnées qui va nous être utile par la suite.

Proposition 89 *$\mathcal{F}$ est presque ordonnée ssi il existe k ($k \geq 1$), $X_1, \ldots, X_k$, $X_i \subseteq V$ ($1 \leq i \leq k$), et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ ($1 \leq i \leq k$), tels que $\mathcal{F}_1 = \mathcal{F}$, $\mathcal{F}_i$ est X_i-ordonnée-renommable ($1 \leq i \leq k$), $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ ($1 \leq i \leq k-1$) et $\mathcal{F}_k = \emptyset$.*

Preuve : ($\Rightarrow$) Soit $\mathcal{F}_1 = \mathcal{F}$, $\mathcal{F}_{i+1} = OReste(\mathcal{F}_i)$ ($i \geq 1$) and $X_i = OBase(\mathcal{F}_i)$ ($i \geq 1$). Par définition, $\mathcal{F}_i$ ($i \geq 1$) est X_i-ordonnée-renommable, et, par hypothèse, il existe k ($k \geq 1$) tel que $\mathcal{F}_k = \emptyset$.

($\Leftarrow$) La preuve se fait par récurrence sur k.

$\underline{k = 1}$ On a $\mathcal{F} = \emptyset$, $OReste(\mathcal{F}) = \emptyset$ et $\mathcal{F}$ est presque ordonnée.

$\underline{k > 1}$ Par hypothèse de récurrence, $\mathcal{F}_2$ est presque ordonnée. Par définition de $OBase(\mathcal{F}_1)$, $OReste(\mathcal{F}_1) \subseteq \mathcal{F}_2$. Donc $OReste(\mathcal{F}_1)$ est presque ordonnée (Prop. 88), $OReste\text{-}itéré(\mathcal{F}_1) = \emptyset$ et $\mathcal{F}_1$ est presque ordonnée. □

Remarque 6 *Si $\mathcal{F}$ est presque ordonnée, alors il existe une formule presque ordonnée $\mathcal{F}'$ telle que $\mathcal{F}$ est obtenue à partir de $\mathcal{F}'$ en renommant certaines variables et de plus il existe k ($k \geq 1$), $X_1, \ldots, X_k$ ($X_i \subseteq V$) ($1 \leq i \leq k$), et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, ($\mathcal{F}_i \subseteq \mathcal{F}'$) ($1 \leq i \leq k$) tels que $\mathcal{F}_1 = \mathcal{F}'$, $\mathcal{F}_i$ est X_i-ordonnée ($1 \leq i \leq k$), $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ ($1 \leq i \leq k-1$) et $\mathcal{F}_k = \emptyset$.*

Malheureusement, la propriété P1 n'est pas vérifiée pour la classe presque ordonnée.

Exemple : Soit $\mathcal{U} = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}, \{x_5\}\}$, $Unit(\mathcal{G}_1 \cup \mathcal{U}) = \{x_1, x_2, x_3, x_4, x_5\}$ et $Noyau(\mathcal{G}_1 \cup \mathcal{U}) = \{ \{x_6, \neg x_7\}, \{\neg x_6, x_7\}, \{\neg x_6, \neg x_7\}, \{x_6, x_7\} \}$. Quel que soit l'ensemble $X \subseteq \{x_6, x_7\}$ la formule $Noyau(\mathcal{G}_1 \cup \mathcal{U})$ n'est pas X-ordonnée-renommable, donc $Base(Noyau(\mathcal{G}_1 \cup \mathcal{U})) = \emptyset$, et donc $Noyau(\mathcal{G}_1 \cup \mathcal{U})$ n'est pas une formule presque ordonnée.

Nous proposons d'ordonner les variables de $\mathcal{F}$ de telle sorte que si l'ensemble $\mathcal{U}$ de clauses unitaire est construit à partir des i premières variables de notre ordre, alors la satisfaisabilité de $\mathcal{F} \cup \mathcal{U}$ peut être testée en temps polynômial. Ceci correspond à la façon dont l'ensemble $\mathcal{U}$ est construit dans l'algorithme Génération (Partie I, Chap. 1, Fig. 1.1)

Définition 37 (permutation convenable, ensemble convenable) *Supposons que $\mathcal{F}$ est presque ordonnée. Il existe $X_1, \ldots X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ $(1 \leq i \leq k)$, tels que $\mathcal{F}_1 = \mathcal{F}$, $X_i = OBase(\mathcal{F}_i)$ $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = OReste(\mathcal{F}_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Soit $W = V \setminus (X_i \cup \ldots \cup X_k)$. Une permutation $(x_1, \ldots, x_n)$ des variables de $\mathcal{F}$ est dite* convenable *si pour tout j $(1 \leq j \leq n)$, on a $\{x_1, \ldots, x_j\} \subseteq W$ ou bien il existe i tel que $\{x_1, \ldots, x_j\} = W \cup X_k \cup X_{k-1} \cup \ldots \cup X_{i+1} \cup X$ avec $X \subseteq X_i$. Un ensemble $\mathcal{U}$ de clauses unitaires est dit* convenable *s'il existe une permutation convenable $(x_1, \ldots, x_n)$ et $i \in \{1, \ldots, n\}$ tels que $var(\mathcal{U}) = \{x_j \mid 1 \leq j \leq i\}$.*

Exemple : On calcule une permutation convenable pour $\mathcal{G}_1$ (nous allons utiliser les notations présentées dans la définition ci-dessus). $\mathcal{F}_1 = \mathcal{G}_1$, $X_1 = OBase(\mathcal{G}_1) = \{x_1, x_2, x_3, x_4, x_5\}$, $\mathcal{F}_2 = OReste(\mathcal{G}_1)$, $X_2 = OBase(OReste(\mathcal{G}_1)) = \{x_6, x_7\}$, et $k = 2$ puisque $Reste(\mathcal{F}_2, X_2) = \emptyset$. L'ensemble W est vide. Soit p_1 la permutation $(x_7, x_6, x_1, x_2, x_3, x_4, x_5)$, p_1 est une permutation convenable, il en est de même pour la permutation $p_2 = (x_6, x_7, x_2, x_3, x_1, x_4, x_5)$. L'ensemble $\mathcal{U} = \{\{\neg x_6\}, \{x_7\}\}$ est convenable car p_1 est une permutation convenable, $\mathcal{U} = \{\{\neg x_1\}, \{x_2\}, \{x_6\}, \{\neg x_7\}\}$ est convenable car p_2 l'est.

La proposition suivante permet de dire qu'à chaque pas de l'algorithme Génération (Partie I, Fig. 1.1), on peut, dans le cas où $\mathcal{U}$ est un ensemble convenable, tester la satisfaisabilité de $\mathcal{F} \cup \mathcal{U}$ en n'utilisant que la résolution unitaire.

Proposition 90 *Supposons que $\mathcal{F}$ est presque ordonnée et ne contient pas de clause unitaire. Soit $\mathcal{U}$ un ensemble convenable de clauses unitaires. Alors $\mathcal{F} \cup \mathcal{U}$ est satisfaisable ssi $Unit(\mathcal{F} \cup \mathcal{U})$ est cohérent.*

Preuve :
($\Rightarrow$) Immédiat.
($\Leftarrow$) Il est suffisant de montrer que $Noyau(\mathcal{F} \cup \mathcal{U})$ est presque ordonnée (Prop. 1 et Corollaire 87). La proposition 89 nous dit qu'il existe $X_1, \ldots, X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ $(1 \leq i \leq k)$, tels que $\mathcal{F}_1 = \mathcal{F}$, $X_i = OBase(\mathcal{F}_i)$ $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = OReste(\mathcal{F}_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Pour tout i $(1 \leq i \leq k)$, $\mathcal{F}_i$ est X_i-ordonnée-renommable. Sans perte de généralité, on peut supposer que $\mathcal{F}_i$ est X_i-ordonnée $(1 \leq i \leq k)$ (voir Remarque 6). Soit $\mathcal{G}_i = \{C' \subseteq Lit(V) \mid$ il existe $C \in \mathcal{F}_i, C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$ et $C' = C \setminus Unit(\mathcal{F} \cup \mathcal{U})\}$ $(1 \leq i \leq k)$. Soit $Y_i = X_i \cap var(\mathcal{G}_i)$ $(1 \leq i \leq k)$. On a $\mathcal{G}_1 = Noyau(\mathcal{F} \cup \mathcal{U})$, $\mathcal{G}_i \subseteq \mathcal{G}_1$ $(1 \leq i \leq k)$ et $\mathcal{G}_k = \emptyset$. Il suffit de prouver que $\mathcal{G}_i$ est Y_i-ordonnée $(1 \leq i \leq k)$ et $\mathcal{G}_{i+1} = Reste(\mathcal{G}_i, Y_i)$ $(1 \leq i \leq k-1)$ (Prop. 89).

Dans un premier temps on va prouver que $\mathcal{G}_i$ est Y_i-ordonnée $(1 \leq i \leq k)$. Soit $C' \in \mathcal{G}_i$. La définition des formules $\mathcal{G}_i$ nous dit qu'il existe

$C \in \mathcal{F}_i$ telle que $C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$, et $C' = C \setminus \overline{Unit(\mathcal{F} \cup \mathcal{U})}$. Soit $l \in C'$. Supposons que l est lié dans C par rapport à $\mathcal{F}_i$. On va prouver que l est lié dans C' par rapport à $\mathcal{G}_i$. Supposons que $Occ_{\mathcal{G}_i}(\overline{l}) \neq \emptyset$. Alors $Occ_{\mathcal{F}_i}(\overline{l}) \neq \emptyset$ et il existe $t \in C$ $(t \neq l)$ tel que $Occ_{\mathcal{F}_i}(\overline{l}) \subseteq \overline{Occ_{\mathcal{F}_i}(\overline{t})}$. On a $C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$, donc $t \notin Unit(\mathcal{F} \cup \mathcal{U})$ et $\overline{t} \notin \overline{Unit(\mathcal{F} \cup \mathcal{U})}$. De plus, $t \notin \overline{Unit(\mathcal{F} \cup \mathcal{U})}$, $Occ_{\mathcal{G}_i}(\overline{t}) = \emptyset$ et $Occ_{\mathcal{G}_i}(\overline{l}) = \emptyset$. Donc $t \in C'$, $Occ_{\mathcal{G}_i}(\overline{l}) \subseteq Occ_{\mathcal{G}_i}(\overline{t})$ et l est lié dans C' par rapport à $\mathcal{G}_i$. Maintenant, on suppose que $l \in Lit(Y_i)$, et l est un littéral positif libre dans C' par rapport à $\mathcal{G}_i$. Alors $l \in Lit(X_i)$ et l est libre dans C par rapport à $\mathcal{F}_i$. Par hypothèse, $\mathcal{F}_i$ est X_i-ordonnée, donc $C \subseteq Lit(X_i)$ et l est le seul littéral positif de C qui est libre dans C. Donc $C' \subseteq Lit(Y_i)$ et l est le seul littéral positif de C' qui est libre dans C'. Ceci prouve que $\mathcal{G}_i$ est Y_i-ordonnée.

On prouve maintenant que $\mathcal{G}_{i+1} = Reste(\mathcal{G}_i, Y_i)$ $(1 \leq i \leq k-1)$. $(\subseteq)$ On a $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$. Donc $\mathcal{F}_{i+1} \subseteq \mathcal{F}_i$ et $\mathcal{G}_{i+1} \subseteq \mathcal{G}_i$. Soit $C' \in \mathcal{G}_{i+1}$. Il existe $C \in \mathcal{F}_{i+1}$ telle que $C' \subseteq C$. On a $C \in Reste(\mathcal{F}_i, X_i)$, c'est pourquoi $C \cap Lit(X_i) = \emptyset$, $C' \cap Lit(Y_i) = \emptyset$ et $C' \in Reste(\mathcal{G}_i, Y_i)$. $(\supseteq)$ Soit $C' \in Reste(\mathcal{G}_i, Y_i)$. On a $C' \subseteq \mathcal{G}_i$ et $C' \cap Lit(Y_i) = \emptyset$. Il existe $C \in \mathcal{F}_i$ telle que $C \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$ et $C' = C \setminus Unit(\mathcal{F} \cup \mathcal{U})$. On prouve d'abord que $var(\mathcal{U}) \cap X_j = \emptyset$ $(1 \leq j \leq i)$ (ceci sera nécessaire pour pouvoir appliquer le Lemme 91). On a $var(C') \cap X_j = \emptyset$ $(1 \leq j < i)$ pour $C \in \mathcal{F}_i$ et $var(C') \cap X_i = \emptyset$ puisque $var(C') \cap Y_i = \emptyset$. Donc $var(C') \subseteq V \setminus (X_1 \cup \ldots \cup X_i)$. On a $var(C') \cap var(\mathcal{U}) = \emptyset$ car $C' \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$ et $C' \cap Unit(\mathcal{F} \cup \mathcal{U}) = \emptyset$. Par hypothèse, $\mathcal{U}$ est convenable, donc $var(\mathcal{U}) \cap X_j = \emptyset$ $(1 \leq j \leq i)$. Maintenant on va prouver que $C \in \mathcal{F}_{i+1}$. On suppose que $C \notin Reste(\mathcal{F}_i, X_i)$. Alors il existe $t \in C \cap Lit(X_i)$. On a $t \notin C'$ puisque $C' \cap Lit(Y_i) = \emptyset$, donc $\overline{t} \in Unit(\mathcal{F} \cup \mathcal{U})$. Si $\overline{t} = x$ $(x \in X_i)$, alors le Lemme 91(i) implique qu'il existe $r \in Lit(V)$ $(r \neq \neg x)$ tel que $r \in C$ et $r \in Unit(\mathcal{F} \cup \mathcal{U})$, contradiction. Supposons que $\overline{t} = \neg x$ $(x \in X_i)$. On a $C' \neq \emptyset$ et $C' \cap Lit(Y_i) = \emptyset$, d'où $C \not\subseteq Lit(X_i)$. Donc x est lié dans C par rapport à $\mathcal{F}_i$, et le Lemme 91(ii) implique qu'il existe $s \in Lit(V)$ $(s \neq x)$ tel que $s \in C$ et $s \in Unit(\mathcal{F} \cup \mathcal{U})$, contradiction. On obtient donc que $C \in Reste(\mathcal{F}_i, X_i)$, $C \in \mathcal{F}_{i+1}$ et $C' \in \mathcal{G}_{i+1}$. □

Lemme 91 *Supposons que $\mathcal{F}$ ne contient pas de clause unitaire et qu'il existe $X_1, \ldots, X_k$, $X_i \subseteq V$ $(1 \leq i \leq k)$, et $\mathcal{F}_1, \ldots, \mathcal{F}_k$, $\mathcal{F}_i \subseteq \mathcal{F}$ $(1 \leq i \leq k)$, tels que $\mathcal{F}_1 = \mathcal{F}$, $\mathcal{F}_i$ est X_i-ordonnée $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = Reste(\mathcal{F}_i, X_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Soit $\mathcal{U}$ un ensemble de clauses unitaires tel que $Unit(\mathcal{F} \cup \mathcal{U})$ est cohérent. Soit $i_0 \in \{1, \ldots, k\}$ tel que pour tout j $(1 \leq j \leq i_0)$ $var(\mathcal{U}) \cap X_j = \emptyset$. Soit $l \in Lit(X_j)$ $(1 \leq j \leq i_0)$ tel que $Occ_{\mathcal{F}_j}(\overline{l}) \neq \emptyset$, et soit $\sigma = (C_1, \ldots, C_p)$ une dérivation unitaire de $\mathcal{F} \cup \mathcal{U}$ telle que $C_p = \{l\}$ et $C_i \neq \{l\}$ $(1 \leq i < p)$.*

i. Si $l = x$ $(x \in X_j)$, alors il existe $r \in Lit(V)$ $(r \neq \neg x)$ tel que $Occ_{\mathcal{F}_j}(\neg x) \subseteq Occ_{\mathcal{F}_j}(r)$ et $\{r\} \in \{C_1, \ldots, C_{p-1}\}$.

ii. Si $l = \neg x$ $(x \in X_j)$, alors pour tout $C \in \mathcal{F}_j$, si $x \in C$ et x est lié dans C par rapport à $\mathcal{F}_j$, alors il existe $s \in Lit(V)$ $(s \neq x)$ tel que $s \in C$ et $\{s\} \in \{C_1, \ldots, C_{p-1}\}$.

Preuve : Par hypothèse $\{l\} \notin \mathcal{F} \cup \mathcal{U}$. Donc, il existe $C' \in \{C_1, \ldots, C_{p-1}\}$ telle que $C' = \{l, l_1, \ldots, l_h\}$ $(h \geq 1)$, $l_i \neq l$ et $\{\overline{l_i}\} \in \{C_1, \ldots C_{p-1}\}$ $(1 \leq i \leq h)$. Nous faisons une preuve par récurrence sur le nombre q de clauses unitaires sur X_g $(1 \leq g \leq j)$ qui apparaissent dans σ.

- $\underline{q = 1}$ $C_p = \{l\}$ est une clause unitaire sur X_j. Donc pour tout i $(1 \leq i \leq h)$, $var(l_i) \in V \setminus (X_1 \cup \ldots \cup X_j)$. Donc $C' \in \mathcal{F}_j$.

  i. On a $l = x$ $(x \in X_j)$. $\mathcal{F}_j$ est X_j-ordonnée, donc x est lié dans C' par rapport à $\mathcal{F}_j$. Par hypothèse, $Occ_{\mathcal{F}_j}(\neg x) \neq \emptyset$, c'est pourquoi il existe i $(1 \leq i \leq h)$ tel que $Occ_{\mathcal{F}_j}(\neg x) \subseteq Occ_{\mathcal{F}_j}(\overline{l_i})$, c'est ce que l'on cherchait à prouver.

  ii. On a $l = \neg x$ $(x \in X_j)$. Soit $C \in \mathcal{F}_j$ tel que $x \in C$. Si x est lié dans C par rapport à $\mathcal{F}_j$, alors il existe i $(1 \leq i \leq h)$ tel que $\overline{l_i} \in C$ car $C' \in Occ_{\mathcal{F}_j}(\neg x)$.

- $\underline{q > 1}$ Soit j' $(1 \leq j' \leq i_0)$ tel que $C' \in \mathcal{F}_{j'}$ et j' est maximal. On a $j' \leq j$ pour $l \in Lit(X_j)$, et $C' \cap Lit(X_{j'}) \neq \emptyset$ puisque j' est maximal.

  On prouve d'abord que pour tout i $(1 \leq i \leq h)$ l_i n'est pas un littéral négatif sur $X_{j'}$. Supposons qu'il existe i $(1 \leq i \leq h)$ tel que $l_i = \neg y$ et $y \in X_{j'}$. On a $\{y\} \in \{C_1, \ldots, C_{p-1}\}$. Il existe p' $(1 \leq p' < p)$ tel que $C_{p'} = \{y\}$ et $C_u \neq \{y\}$ $(1 \leq u < p')$. Soit $\sigma' = (C_1, \ldots, C_{p'})$. On a $Occ_{\mathcal{F}_{j'}}(\neg y) \neq \emptyset$ $(C' \in Occ_{\mathcal{F}_{j'}}(\neg y))$. Donc les hypothèses du Lemme 91(i) sont satisfaites par y et σ'. Par hypothèse de récurrence, il existe $r \in Lit(V)$ $(r \neq \neg y)$ tel que $Occ_{\mathcal{F}_{j'}}(\neg y) \subseteq Occ_{\mathcal{F}_{j'}}(r)$ et $\{r\} \in \{C_1, \ldots, C_{p'-1}\}$. Donc $r \in C'$. Si $r = l$, alors $\{l\} \in \{C_1, \ldots, C_{p'-1}\}$, ce qui entre en contradiction avec l'hypothèse. Supposons que $r \neq l$. Alors il existe a $(1 \leq a \leq h)$ tel que $r = l_a$, donc $\{\overline{r}\} \in \{C_1, \ldots, C_{p-1}\}$. Ce qui implique que $Unit(\mathcal{F} \cup \mathcal{U})$ est incohérent , contradiction. Ceci prouve que l_i $(1 \leq i \leq h)$ n'est pas un littéral négatif de $X_{j'}$.

  On considére maintenant deux cas :

  1. Supposons qu'il existe i $(i \leq i \leq h)$ tel que $l_i = y$ et $y \in X_{j'}$. On a $\{\neg y\} \in \{C_1, \ldots, C_{p-1}\}$. Il existe p' $(1 \leq p' < p)$ tel que $C_{p'} = \{\neg y\}$ et $C_u \neq \{\neg y\}$ $(1 \leq u < p')$. Soit $\sigma' = (C_1, \ldots, C_{p'})$. On a $Occ_{\mathcal{F}_{j'}}(y) \neq \emptyset$ $(C' \in Occ_{\mathcal{F}_{j'}}(y))$. Donc les hypothèses du Lemme 91(ii) sont satisfaites par $\neg y$ et σ'. Supposons que y est lié dans C' par rapport à $\mathcal{F}_{j'}$. Par hypothèse de récurrence, il existe $s \in Lit(V)$ $(s \neq y)$ tel que $s \in C'$ et

$\{s\} \in \{C_1, \ldots, C_{p'-1}\}$. Si $s = l$, alors $\{l\} \in \{C_1, \ldots, C_{p'-1}\}$, ce qui entre en contradiction avec les hypothèses. Supposons que $s \neq l$. Alors il existe a $(1 \leq a \leq h)$ tel que $s = l_a$, donc $\{\overline{s}\} \in \{C_1, \ldots, C_{p-1}\}$. Par conséquent, $Unit(\mathcal{F} \cup \mathcal{U})$ est incohérent, contradiction. Ceci prouve que y est libre dans C' par rapport à $\mathcal{F}_{j'}$. Par hypothèses $\mathcal{F}_{j'}$ est $X_{j'}$-ordonnée, donc $C' \subseteq Lit(X_{j'})$. Mais $l \in C' \cap Lit(X_j)$, donc $j = j'$ et $C' \in \mathcal{F}_j$.

i. On a $l = x$ $(x \in X_j)$ et x est lié dans C' par rapport à $\mathcal{F}_j$. Par hypothèse, $Occ_{\mathcal{F}_j}(\neg x) \neq \emptyset$, c'est pourquoi il existe i $(1 \leq i \leq h)$ tel que $Occ_{\mathcal{F}_j}(\neg x) \subseteq Occ_{\mathcal{F}_j}(\overline{l_i})$, qui est le résultat recherché.

ii. On a $l = \neg x$ $(x \in X_j)$. Soit $C \in \mathcal{F}_j$ tel que $x \in C$. Si x est lié dans C par rapport à $\mathcal{F}_j$, alors il existe i $(1 \leq i \leq h)$ tel que $\overline{l_i} \in C$ car $C' \in Occ_{\mathcal{F}_j}(\neg x)$.

2. Il reste à examiner le cas où $C' \cap Lit(X_{j'}) = \{l\}$. Alors on a $j = j'$, $C' \in \mathcal{F}_j$ et $var(l_i) \in V \setminus (X_1 \cup \ldots \cup X_j)$ $(1 \leq i \leq h)$.

i. On a $l = x$ $(x \in X_j)$, et x est lié dans C' par rapport à $\mathcal{F}_j$. On conclue de la même manière que pour le cas 1i.

ii. On a $l = \neg x$ $(x \in X_j)$. On conclue de la même manière que pour le cas 1ii.

□

Ces résultats nous permettent donc de proposer une méthode efficace pour générer tous les modèles des formules presque ordonnées.

Proposition 92 *Si $\mathcal{F}$ est presque ordonnée et ne contient pas de clause unitaire, alors les modèles de $\mathcal{F}$ peuvent être générés avec un délai $O(nN)$, si une permutation convenable est donnée.*

Preuve : On rappelle que la résolution unitaire peut être implémentée de manière linéaire. Donc pour tout ensemble convenable $\mathcal{U}$ de clauses unitaires, on peut décider en temps $O(N)$ si $\mathcal{F} \cup \mathcal{U}$ est satisfaisable (Prop. 90). Donc, si la permutation qui est donnée en entrée à l'algorithme Génération (Partie I, Chap. 1, Fig. 1.1) est convenable, alors les modèles de $\mathcal{F}$ peuvent être générés avec un délai $O(nN)$. □

2.3 Reconnaissance des formules presque ordonnées

Pour tester si une formule est presque ordonnée, la première chose à faire est de calculer la base ordonnée de cette formule.

Proposition 93 *La base ordonnée de $\mathcal{F}$ peut être calculée en temps $O(nN)$.*

Preuve : Soit $x \in V$. On déduit de la Prop. 85 que $x \notin OBase(\mathcal{F})$ ssi $x \stackrel{\mathcal{F}}{\Longrightarrow}^* \neg x$ et $\neg x \stackrel{\mathcal{F}}{\Longrightarrow}^* x$. Donc $x \in OBase(\mathcal{F})$ ssi x et $\neg x$ n'appartiennent pas à la même composante fortement connexe de $G(\mathcal{F})$. On rappelle que la construction de $G(\mathcal{F})$, et le calcul de ses composantes fortement connexes peut être fait en temps $O(nN)$ (preuve de la Prop. 80). □

Connaissant la base ordonnée d'une formule, il est facile de calculer son reste ordonné, si on itère ce calcul, on obtient le reste itéré de la formule. Déterminer si celui-ci est vide permet de dire si $\mathcal{F}$ est presque ordonnée.

Proposition 94 *On peut tester si une formule $\mathcal{F}$ est presque ordonnée, et (si c'est le cas) construire une permutation convenable, en temps $O(n^2N)$.*

Preuve : L'ensemble $OBase(\mathcal{F})$ peut être calculé en temps $O(nN)$ (Prop. 93). On remarque que si $OBase(\mathcal{F})$ est connu, alors il est facile de calculer $OReste(\mathcal{F})$ en temps $O(N)$. Donc on peut obtenir $OReste(\mathcal{F})$ en partant de $\mathcal{F}$ en temps $O(nN)$. Si $OBase(\mathcal{F})$ n'est pas vide, alors l'ensemble des variables sur lequel $OReste(\mathcal{F})$ est défini est strictement inclus dans l'ensemble V ; par conséquent, le calcul de *OReste-itéré*$(\mathcal{F})$ demande au plus n étapes, et peut donc être effectué en temps $O(n^2N)$. On obtient en plus comme sous-produit de ce calcul les ensembles $X_1, \ldots, X_k$ et les formules $\mathcal{F}_1, \ldots, \mathcal{F}_k$, tels que $\mathcal{F}_1 = \mathcal{F}$, $X_i = OBase(\mathcal{F}_i)$ $(1 \leq i \leq k)$, $\mathcal{F}_{i+1} = OReste(\mathcal{F}_i)$ $(1 \leq i \leq k-1)$ et $\mathcal{F}_k = \emptyset$. Il est facile de construire en temps $O(n)$ une permutation convenable lorsque l'on connaît $X_1, \ldots, X_k$ et $V \setminus (X_1 \cup \ldots \cup X_k)$. □

Conclusion et Perspectives

Conclusion

Dans cette thèse, le fil conducteur que nous avons suivi est la génération à délai polynômial de tous les modèles d'une formule CNF propositionnelle, et plus précisément, la génération à délai polynômial en utilisant uniquement la résolution unitaire. Cela nous a conduit à établir des résultats intéressants :

Nous avons donné un algorithme permettant de générer les modèles de toute formule. Nous avons montré que cet algorithme est efficace (c'est à dire que l'écart entre deux solutions consécutives est polynômial) pour presque toutes les classes pour lesquelles un algorithme polynômial est connu pour résoudre le problème SAT. Les seules classes pour lesquelles notre algorithme n'est pas efficace, sont celles pour lesquelles l'existence d'un tel algorithme impliquerait que P=NP.

Nous nous sommes aussi attachés à la classe des formules Horn étendues introduite par Chandru et Hooker. Nous avons étudié les origines de cette classe, et avons prouvé que l'on peut générer tous les modèles de telles formules avec un délai $O(nN)$ en n'utilisant que la résolution unitaire. Malheureusement, il n'existe pas, actuellement, d'algorithme de reconnaissance des formules Horn étendues. Nous avons donc du travailler sur deux classes proches, les formules Horn étendues simples et Horn élargies simples pour lesquelles nous avons proposé une analyse pointue qui nous a permis de dégager une structure importante : les agrégats. Cette structure nous a amenés à la rédaction d'un algorithme linéaire de reconnaissance des formules Horn élargies simples et Horn étendues simples.

Le résultat le plus intéressant de cette thèse est sans doute la présentation et l'étude d'une nouvelle classe des formules, que nous avons appelée classe des formules ordonnées. Cette extension de Horn peut être reconnue en temps $O(nN)$; on peut tester si une formule ordonnée est satisfaisable en temps $O(N)$, en utilisant uniquement la résolution unitaire, exactement comme pour une formule de Horn ; on peut générer les modèles de ces formules avec un délai $O(nN)$. On peut de plus tester si une formule peut être renommée en une formule ordonnée en temps $O(nN)$ en utilisant une technique très proche de celle utilisée pour tester si une formule est Horn-renommable. Ce résultat est non trivial, en effet,

pour les formules Horn généralisées, Eiter et al. [23] ont prouvé que ce problème est NP-complet. On généralise encore cette classe en définissant un ensemble de formules, appelées presque ordonnées qui ne sont pas ordonnées-renommables, mais qui le sont presque. Nous proposons en outre une méthode pour générer tous les modèles des formules presque ordonnées avec un délai polynômial (en n'utilisant que la résolution unitaire).

On peut résumer les résultats obtenus dans cette thèse en examinant le tableau de la génération à délai polynômial (Fig. 2) et le graphe (Fig. 3) représentant les inclusions des classes que nous avons plus particulièrement étudiées ici.

Classe	Reconnaissance	Satisfaisabilité	Génération
Horn	$O(N)$	$O(N)$	$O(nN)$
Horn renommable	$O(N)$	$O(N)$	$O(nN)$
Binaire	$O(N)$	$O(N)$	$O(nN)$
Équilibrée	polynômial	$O(N)$	$O(nN)$
Γ_k	$O(n^k N)$	$O(n^k N)$	$O(n^{k+1} N)$
Γ_k-renommable	NP-complet	$O(n^k N)$	$O(n^{k+1} N)$
Quad	$O(N^2)$	$O(N^2)$	non si P$\neq$NP
Ω_k et Δ_k	$O(n^{k+1})$	$O(n^{k+1})$	non si P$\neq$NP
Presque Horn(*)	$O(nN)$	$O(1)$	$O(nN)$
q-Horn	$O(N)$	$O(N)$	$O(nN)$
Horn étendue (resp. élargie)	Pb ouvert	$O(N)$	$O(nN)$
Horn étendue (resp. élargie) simple	$O(N)$	$O(N)$	$O(nN)$
Ordonnée	$O(nN)$	$O(N)$	$O(nN)$
Ordonnée-renommable	$O(nN)$	$O(N)$	$O(nN)$
Presque ordonnée(*)	$O(n^2 N)$	$O(1)$	$O(nN)$

(*) sans clause unitaire

FIG. 2 – *Génération à délai polynômial*

Perspectives

Ces résultats nous ouvrent les portes de recherches dans d'autres directions, nous présentons ici quelques idées à creuser.

Par exemple il nous semble possible d'utiliser les résultats établis lors de l'étude des formules Horn étendues simples pour rechercher un algorithme polynômial (sans doute ne sera t'il pas linéaire) permettant de déterminer si une formule est Horn étendue.

L'algorithme que nous avons présenté pour générer tous les modèles d'une formule peut très facilement être adapté aux problèmes de satisfaction de contraintes (CSP). On peut étudier l'ensemble des classes de

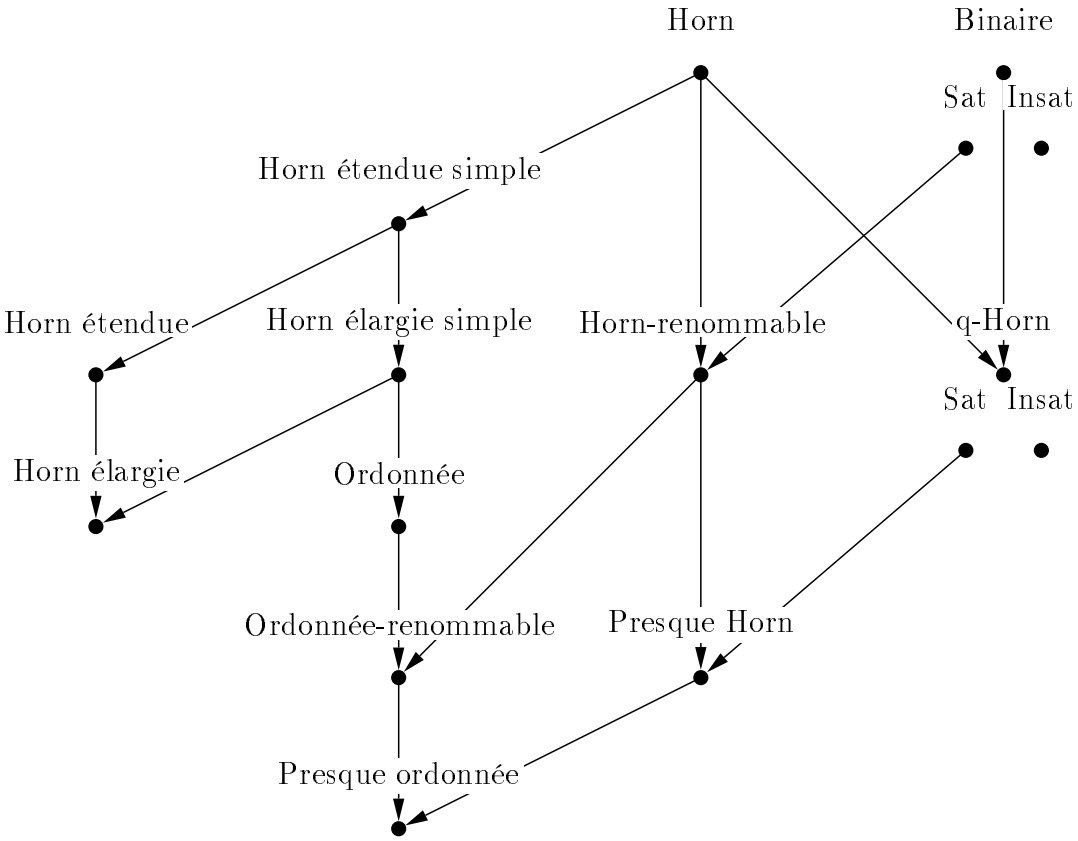

FIG. 3 – *Graphe d'inclusion des principales classes étudiées dans cette thèse*

CSP pour lesquelles on connaît un algorithme polynômial et regarder celles pour lesquelles il est possible de générer toutes les solutions à délai polynômial. On peut essayer de trouver un analogue à la résolution unitaire et voir quelles sont les classes pour lesquelles on peut trouver toutes les solutions avec cette seule méthode.

De même que l'étude des formules de Horn a conduit à la définition des fonctions booléennes de Horn, il est peut-être intéressant de regarder les modèles des formules ordonnées et de voir s'il est possible de définir un concept similaire que l'on pourrait appeler « fonctions booléennes ordonnées ».

On peut en outre chercher quels problèmes peuvent se représenter avec des formules ordonnées ou presque ordonnées. Peut-être des problèmes

concrets se modélisent-ils facilement avec des formules appartenant à l'une de ces classes ?

Index

Index alphabétique

Bibliographie

[1] B. Aspvall. Recognizing disguised NR(1) instances of the satisfiability problem. *Journal of Alogrithms*, 1 :97–103, 1980.

[2] B. Aspvall, M. F. Plass, and R. E. Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Information Processing Letters*, 8(3) :121–123, Mar. 1979. See also erratum [3].

[3] B. Aspvall, M. F. Plass, and R. E. Tarjan. Erratum : « A linear-time algorithm for testing the truth of certain quantified Boolean formulas » [Inform. Process. Lett. 8 (1979), no. 3, 121–123. *Information Processing Letters*, 14(4) :195–195, June 1982. See [2].

[4] E. Benoist and J.-J. Hébrard. Recognition of simple enlarged Horn formulas and simple extended Horn formulas. Technical Report 5, Université de Caen, Cahier du GREYC, Université de Caen (France), 1998.

[5] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using P-Q tree algorithms. *J. Comput. System Sci.*, 13 :335–379, 1976.

[6] E. Boros, Y. Crama, and P. L. Hammer. Polynomial-time inference of all valid implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1 :21–32, 1990.

[7] E. Boros, P. L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik. An implementation of logical analysis of data. Technical Report 22-96, Rutcor Research Report, RUTCOR, Rutgers Center for Operations Research, Bush Campus, P.O. Box 5062, New Brunswick, New Jersey 08903-5062, July 1996.

[8] E. Boros, P. L. Hammer, and X. Sun. Recognition of q-Horn formulae in linear time. *DAMATH : Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 55, 1994.

[9] E. Canfield and S. Williamson. A loop-free algorithm for generating the linear extensions of a poset. *Order*, 12 :1–18, 1995.

[10] R. Chandrasekaran. Integer programming problems for which a simple rounding type of algorithm works. In e. W.R. Pulleyblank, editor, *Progress in Combinatorial Optimization*, pages 101–106. Academic press Canada, Toronto, Ontario, Canada, 1984.

[11] V. Chandru, C. Coullard, P. Hammer, M. Montanez, and X. Sun. On renamable Horn and generalized Horn functions. *Annals of Mathematics*, 1(1) :33–47, 1990.

[12] V. Chandru and J. N. Hooker. Extended Horn sets in propositional logic. *Journal of the ACM*, 38(1) :205–221, Jan. 1991.

[13] C.-L. Chang and R. C.-T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973.

[14] M. Conforti and G. Cornuéjols. A class of logic problems solvable by linear programming. In IEEE, editor, *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, pages 670–675, Pittsburgh, PN, Oct. 1992. IEEE Computer Society Press.

[15] M. Conforti, G. Cornuejols, A. Kapoor, and K. Vušković. Recognizing balanced $0, \pm 1$ matrices. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 103–111, Arlington, Virginia, 23–25 Jan. 1994.

[16] S. A. Cook. The complexity of theorem-proving procedures. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, Shaker Heights, Ohio, 3–5 1971 1971.

[17] Crama, Ekin, and Hammer. Variable and term removal from boolean formulae. *DAMATH : Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 75, 1997.

[18] N. Creignou and J.-J. Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique Théorique et Applications / Theoretical Informatics and Applications*, 31(6) :499–511, 1997.

[19] M. Dalal. An almost quadratic class of satisfiability problems. In W. Wahlster, editor, *Proceedings of the 12th European Conference on Artificial Intelligence*, pages 355–359. John Wiley and Sons, 1996.

[20] M. Dalal and D. W. Etherington. A hierarchy of tractable satisfiability problems. *Information Processing Letters*, 44(4) :173–180, Dec. 1992.

[21] P. Dietz, M. Furst, and J. Hopcroft. A linear time algorithm for the generalized consecutive retrieval problem. Technical Report TR-79-386, Department of Computer Science, Cornell University, Ithaca, NY, 1979.

[22] T. Eiter, T. Ibaraki, and K. Makino. On disguised double Horn functions and extensions. In *15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373 of *lncs*, pages 50–60, Paris France, 25–27 Feb. 1998. Springer.

[23] T. Eiter, P. Kilpelainen, and H. Mannila. Recognizing renamable genralized propositional Horn formulas is NP-complete. *Discrete Appl. Math.*, 59 :23–31, 1995.

[24] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicmmodity flow problems. *SIAM Journal on Computing*, 5 :691–700, 1976.

[25] H. Farreny and M. Ghallab. *Éléments d'intelligence artificielle*. Éditions Hermès, 1987.

[26] H. N. Gabow and E. W. Myers. Finding all spanning trees of directed and undirected graphs. *SIAM Journal on Computing*, 7(3) :280–287, Aug. 1978.

[27] G. Gallo and M. G. Scutellà. Polynomially solvable satisfiability problems. *Information Processing Letters*, 29(5) :221–227, Nov. 1988.

[28] M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing . Technical Report 96021, LIRMM, Montpellier, 1996. To appear in "Theoretical Computer Science".

[29] M. Habib, C. Paul, and L. Viennot. A synthesis on partition refinement : A useful routine for strings, graphs, boolean matrices and automata. In *Proc. STACS'98*, pages 25–38, 1998.

[30] R. Hariharan, S. Kapoor, and V. Kumar. Faster enumeration of all spanning trees of a directed graph. In *Proc. 4th Worksh. Algorithms & Data Structures*, number 955 in Lecture Notes in Computer Science, pages 428–439. Springer Verlag, 1995.

[31] J.-J. Hébrard. A linear algorithm for renaming a set of clauses as a Horn set. *Theoretical Computer Science*, 124 :343–350, 1994.

[32] J.-J. Hébrard and P. Luquet. The Horn basis of a set of clauses. *Journal of Logic Programming*, 34(1) :59–66, Jan. 1998.

[33] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27(3) :119–123, Mar. 1988.

[34] A. D. Kalvin and Y. L. Varol. On the generation of all topological sortings. *Journal of Algorithms*, 4(2) :150–162, June 1983.

[35] H. Kleine-Büning. On generalized Horn formulas and k-resolution. *Theoretical Computer Science*, 116(2) :405–413, Aug. 1993.

[36] H. Kleine-Büning and T. Lettmann. *Aussagenlogik : Deduktion und Algorithmen.* B. G. Teubner, Stuttgart, 1994.

[37] H. Kleine-Büning and T. Lettmann. *Propositional Logic : Deduction and Algorithms.* Cambridge University Press, The Edinburgh Building, Shaftesbury Road, Cambridge CB2 2RU, UK, 1999.

[38] H. R. Lewis. Renaming a set of clauses as a Horn set. *Journal of the Association for Computing Machinery*, 25(1) :134–135, january 1978.

[39] D. Loveland. *Automatic Theorem Proving.* Elsevier Science Publishers North-Holland, 1978.

[40] D. Pretolani. Hierarchies of polynomially solvable satifiability problems. *Annals of Mathematics and Artificial Intelligence*, 17 :339–357, 1996.

[41] G. Pruesse and F. Ruskey. Generating linear extensions fast. *SIAM. J. Comp*, page to appear, 1992.

[42] R. Read and R. Tarjan. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5 :237–252, 1975.

[43] E. Rich and K. Knight. *Artificial Intelligence.* McGraw-Hill, New York, second edition, 1991.

[44] J. S. Schlipf, F. S. Annexstein, J. V. Franco, and R. P. Swaminathan. On finding solutions for extended Horn formulas. *Information Processing Letters*, 54(3) :133–137, May 1995.

[45] A. Shioura, A. Tamura, and T. UnoA. An optimal algorithm for scanning all spannig trees of an undirected graph. *SIAM Journal on Computing*, 26(3) : :678–692, 1997.

[46] R. P. Swaminathan and D. K. Wagner. The arborescence-realization problem. *Discrete Applied Mathematics*, 59 :267–283, 1995.

[47] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1 :146–160, 1972.

[48] S. Yamasaki and S. Doshita. The satisfiability problem for a class consisting of Horn sentences and some non-Horn sentences in proportional logic. *Information and Control*, 59(1–3) :1–12, Oct./Nov./Dec. 1983.

Résumé

Nous nous sommes intéressés à l'étude des classes de formules CNF propositionnelles pour lesquelles il est possible de générer tous les modèles de façon efficace (i.e. avec un délai polynômial entre chaque modèle généré).

Nous proposons un algorithme générique permettant de générer tous les modèles d'une formule quelconque. Nous prouvons que pour les principales classes de formules pour lesquelles on sait résoudre le problème SAT efficacement, notre algorithme génére les solutions avec un délai polynômial. Cette étude nous pousse à étudier ensuite plus en détail les classes de formules pour lesquelles la résolution unitaire est le seul outil utilisé pour la génération.

C'est pourquoi nous nous intéressons aux formules Horn étendues introduites par Chandru et Hooker. Malheureusement, il n'existe pas encore d'algorithme polynômial permettant de tester si une formule appartient à cette classe. Nous étudions donc la classe des formules Horn étendues simples qui est une restriction de la classe pécédente pour laquelle Swaminathan et Wagner ont proposé un algorithme de reconnaissance quadratique. Une étude de la structure de ces formules nous permet de proposer un algorithme de reconnaissance linéaire.

Le résultat de plus original de ce travail est la présentation de la classe des formules ordonnées. Cette classe étend de façon naturelle celle des formules de Horn, en préservant les propriétés relatives à la résolution unitaire (SAT, génération de modèles). De plus, nous proposons un algorithme quadratique permettant de déterminer si une formule est ordonnée-renommable. En outre nous présentons la classe des formules presque ordonnées qui englobe les formules ordonnées-renommables. Ces formules peuvent être reconnues en temps polynômial et on peut aussi générer leurs modèles en n'utilisant que la résolution unitaire, à condition de disposer d'un ordre convenable sur les variables.

Mots-clés: Logique propositionnelle, problème SAT, satisfaisabilité, génération, délai polynomial, formules ordonnées, algorithmique, Horn, résolution unitaire.

Abstract

This work deals with classes of propositional CNF formulas for which it is possible to generate solutions efficiently (i.e. with a polynomial delay between any two consecutive solutions).

We present a generic algorithm for the generation of the models of any formula. Then we show that our algorithm generates with polynomial delay the models of any formula belonging to the main classes for which a polynomial algorithm for the satisfiability problem is known. This study leads us to focus on classes of formulas for which unit resolution is the only tool required for the generation.

We study the class of extended Horn formulas introduced by Chandru and Hooker. Unfortunately, no polynomial time algorithm is known for determining whether a formula belongs to this class. That is why we study the class of simple extended Horn formulas which is a subclass of the extended Horn formulas presented by Swaminathan and Wagner. They give a quadratic time recognition algorithm. Our study permits to present a linear time recognition algorithm.

But the most original result of this work is the presentation of the class of ordered formulas. This class extends Horn on a natural way, preserving properties concerning unit resolution. (satisfiability, polynomial time generation). Moreover, we propose a quadratic time algorithm for the recognition of ordered-renamable formulas. Finally, we present the class of almost ordered formulas which includes the ordered-renamables formulas. These formulas can be recognized in polynomial time, and, provided that the variable are suitably ordered, one can generate with polynomial delay the models of almost ordered formulas.

Keywords: Propositionnal logic, problem SAT, satisfiability, generation, polynomial delai, ordered formulas, algorithmic, Horn, unit resolution.