De la validité des formules booléennes quantifiées : étude de complexité et exploitation de classes traitables au sein d'un prouveur QBF

THÈSE

présentée et soutenue publiquement le 13 décembre 2005

en vue de l'obtention du

Doctorat de l'Université d'Artois

(Spécialité Informatique)

par

Florian LETOMBE

Composition du jury

Rapporteurs :	Eric MONFROY	Professeur, Université de Nantes
	Pascal NICOLAS	Maître de conférences (HdR), Université d'Angers
Examinateurs :	Sylvie COSTE-MARQUIS	Maître de conférences, Université d'Artois
	Eric GREGOIRE	Professeur, Université d'Artois
	Daniel LE BERRE	Maître de conférences, Université d'Artois
	Pierre MARQUIS	Professeur, Université d'Artois (directeur de thèse)
Invités :	Salem BENFERHAT	Professeur, Université d'Artois
	Lakhdar SAÏS	Professeur, Université d'Artois

UNIVERSITÉ D'ARTOIS – CENTRE DE RECHERCHE EN INFORMATIQUE DE LENS – CNRS FRE2499
rue Jean Souvraz, SP 18 – F-62307 LENS CEDEX

Remerciements

Je vous remercie tous.

« Telle est la vie des hommes. Quelques joies, très vite effacées par d'inoubliables chagrins.
« Il n'est pas nécessaire de le dire aux enfants. »
Marcel Pagnol, *Le château de ma mère*

À mes grand-mères

Table des matières

Liste des définitions

Table des figures

Liste des tableaux

Liste des algorithmes

Introduction générale

« Le secret d'un bon discours, c'est d'avoir une bonne introduction et une bonne conclusion. Ensuite, il faut s'arranger pour que ces deux parties ne soient pas très éloignées l'une de l'autre ... »
George Burns [1]

Présentation du problème

Depuis plus de trente ans, le problème de satisfiabilité de formules propositionnelles (le célèbre problème SAT) fait l'objet d'études très poussées en informatique. Du point de vue théorique, SAT est le premier problème à avoir été prouvé **NP**-complet (Cook 1971). Depuis lors, de très nombreux problèmes, issus de domaines variés, ont rejoint SAT dans sa catégorie. Comme **NP** est une classe de complexité fermée par réduction polynomiale, tout problème de **NP** peut être réduit à SAT. Ainsi, du point de vue pratique, de nombreuses applications de SAT existent, par exemple la conception assistée par ordinateur de circuits intégrés (Larrabee 1992; Kim *et al.* 2000), la planification classique (Kautz & Selman 1992), la vérification de modèles de systèmes dynamiques (Biere *et al.* 1999a), l'ordonnancement (Crawford & Baker 1994), ou encore la cryptographie (Massacci & Marraro 2000) pour ne citer que celles-ci. Par ailleurs, les progrès réalisés ces dernières années dans la résolution pratique de SAT poussent les chercheurs à s'orienter vers des problèmes de complexité plus élevée.

Le problème QBF de la validité des formules booléennes quantifiées est une généralisation de SAT dans le sens où une formule propositionnelle est une formule booléenne quantifiée particulière dans laquelle toutes les variables sont quantifiées existentiellement. Formellement, QBF [2] est le problème de décision consistant à déterminer si une formule booléenne quantifiée donnée du type $\forall y_1 \exists x_1 \ldots \forall y_h \exists x_h . \phi$ est valide ou pas. Intuitivement, une telle formule, où ϕ est une formule propositionnelle construite sur l'ensemble des variables $\{x_1, \ldots, x_h, y_1, \ldots y_h\}$ est valide si et seulement si pour toute valeur de vérité donnée à y_1 dans ϕ, il existe une valeur de vérité pour x_1 dans $\phi \ldots$ pour toute valeur de vérité donnée à y_h dans ϕ, il existe une valeur de vérité pour x_h dans ϕ telle que remplacer dans ϕ chaque variable par la constante booléenne codant la valeur de vérité associée conduit à une formule propositionnelle valide.

QBF constitue le problème **PSPACE**-complet canonique. Comme la classe **PSPACE** est elle aussi fermée par réduction polynomiale, tout problème de **PSPACE** (i.e. tout problème de décision décidable en espace polynomial) est réductible à QBF.

Puisque SAT constitue une restriction de QBF, QBF est au moins aussi difficile que SAT du point de vue calculatoire. Ainsi, on sait que **NP** $\subseteq$ **PSPACE**. On conjecture en théorie de la complexité que l'inclusion est stricte donc que QBF est en fait plus difficile que SAT. Dans tous les cas, on ne connaît pas d'algorithme en temps polynomial pour résoudre SAT ou QBF et on conjecture qu'il n'en existe pas.

Néanmoins, QBF a une importance croissante en intelligence artificielle. Cela peut s'expliquer par le fait que, en tant que problème **PSPACE**-complet canonique, de nombreux problèmes d'IA peuvent

1. Burns (George), acteur, scénariste, producteur américain né à New York (1896–1996).
2. En fait, le problème que nous noterons hQBF dans la suite.

être réduits à QBF de manière polynomiale (cf. (Egly *et al.* 2000; Fargier *et al.* 2000; Besnard *et al.* 2002; Rintanen 1999a; Pan *et al.* 2002; Pan & Vardi 2003)) ; de plus, pour différents domaines de l'intelligence artificielle (parmi lesquels la planification, le raisonnement non monotone, l'inférence paraconsistante), une approche basée sur une traduction vers l'ensemble $\mathcal{QBF}$ des formules booléennes quantifiées peut se révéler plus efficace en pratique que des algorithmes dédiés dépendants du domaine. Par conséquent, de nombreux prouveurs QBF ont été conçus (cf. principalement (Cadoli *et al.* 1998; Feldmann *et al.* 2000; Rintanen 1999b; Rintanen 2001; Giunchiglia *et al.* 2001a; Letz 2002; Pan & Vardi 2004; Benedetti 2005; GhasemZadeh *et al.* 2004; Zhang & Malik 2002; Audemard & Saïs 2004)) et évalués (Le Berre *et al.* 2003; Le Berre *et al.* 2005) ces dernières années.

De fait, le champ d'application de QBF est extrêmement vaste. Parmi les problèmes pouvant être modélisés à l'aide de formules booléennes quantifiées, citons la vérification de programmes (la vérification de modèles bornés (Biere *et al.* 1999b) ou la construction de modèles bornés (Ayari & Basin 2000)), des applications logicielles telles que la vérification d'équivalence (Scholl & Becker 2001), des tâches en intelligence artificielle telles que la planification dans l'incertain mais aussi divers problèmes d'inférence ou de satisfiabilité dans les logiques non standard ou modales (Rintanen 1999a; Castellini *et al.* 2003) ou les jeux séquentiels à deux joueurs et à information incomplète (Coste-Marquis *et al.* 2002). Ainsi, la résolution de problèmes de formules booléennes quantifiées est, depuis quelques années, un sujet d'importance en pratique. L'intérêt grandissant des industriels pour la vérification formelle en est l'une des explications.

Objectifs

Puisque QBF est un problème intéressant mais calculatoirement difficile, il importe de définir des méthodes et algorithmes permettant d'augmenter le nombre d'instances de formules booléennes quantifiées décidables en pratique, i.e. en un temps « raisonnable ». Pour ce faire, plusieurs voies sont à explorer parmi lesquelles développer des algorithmes stochastiques incomplets qui ne fourniront pas à coup sûr une réponse exploitable, ou restreindre le problème à des sous-problèmes plus simples (en complexité). C'est dans cette seconde voie que nous nous sommes engagés.

Nous nous sommes focalisés sur des restrictions de QBF obtenues en imposant aux formules d'entrée $\forall y_1 \exists x_1 \ldots \forall y_n \exists x_n . \phi$ des conditions sur ϕ, voire sur la suite de quantifications $\forall y_1 \exists x_1 \ldots \forall y_n \exists x_n$. Globalement, notre objectif fut double : (1) déterminer la complexité de certaines restrictions de QBF pour lesquelles elle restait à être identifiée et (2) explorer l'utilisation de restrictions de QBF décidables en temps polynomial au sein de méthodes pour résoudre le problème QBF entier.

Contributions

Les contributions apportées dans cette thèse à l'étude des formules booléennes quantifiées peuvent être réparties sur deux plans : un plan théorique et un plan pratique.

Le plan théorique

Notre étude théorique a consisté principalement à mettre en évidence le caractère traitable ou non de certaines restrictions de formules booléennes quantifiées. En particulier, nous avons étudié la complexité de fragments propositionnels utilisés pour la compilation de « connaissances ».

Nous nous sommes focalisés sur des fragments qui constituent des classes polynomiales pour SAT, i.e. des ensembles de formules pour lesquelles l'appartenance peut être décidée en temps polynomial, ainsi que la satisfiabilité de leurs éléments. Nous avons principalement mis l'accent sur des fragments complets pour la logique propositionnelle, comme les diagrammes de décision binaires ordonnés, les impliqués premiers ou les formules sous forme normale négative décomposable et avons déterminé s'ils constituent aussi des classes polynomiales pour QBF.

Les résultats obtenus montrent que les restrictions de QBF obtenues en considérant des formules d'entrée du type $\forall y_1 \exists x_1 \ldots \forall y_n \exists x_n.\phi$ où ϕ appartient à de tels fragments sont aussi difficiles que le problème QBF en toute généralité (**PSPACE**-complet). Nous avons néanmoins mis en évidence deux restrictions pour lesquelles un algorithme en temps polynomial pour la validité existe.

Le plan pratique

Nous avons développé un prouveur QBF basé sur l'exploitation de certaines classes polynomiales pour les formules booléennes quantifiées. Ce prouveur intègre une nouvelle heuristique de branchement destinée à promouvoir la génération de sous-formules Horn renommables quantifiées au sein d'une formules booléenne quantifiée du type $\forall y_1 \ \exists x_1 \ \ldots \forall y_n \ \exists x_n.\phi$ où ϕ est une formule propositionnelle sous forme normale conjonctive.

De plus, afin d'exploiter au maximum cette heuristique, notre prouveur QBF reconnaît en temps polynomial les formules Horn renommables quantifiées et les résout également en temps polynomial. Notre étude montre que les prouveurs QBF actuels ne résolvent pas facilement les formules Horn renommables quantifiées (et les formules de Horn quantifiées non plus par extension) et donc que la détection de sous-classes polynomiales peut se révéler être un apport intéressant à un prouveur QBF.

Nous montrons que cet apport dû à l'action combinée de notre heuristique et de la détection de formules Horn renommables quantifiées est évidemment effectif sur les instances de formules de Horn quantifiées et les formules Horn renommables quantifiées mais également sur d'autres instances de formules telles que celles proposées par G. Pan et issues d'une traduction d'instances du problème de la satisfiabilité pour la logique modale K (Balsiger *et al.* 2000).

Plan du mémoire

Ce document s'articule en trois parties distinctes, complétées par cette introduction, une conclusion, une annexe et une bibliographie.

En premier lieu, nous introduisons les bases formelles sur lesquelles repose la suite du manuscrit.

Le premier chapitre décrit syntaxiquement et sémantiquement, dans un premier temps la logique propositionnelle, puis, dans un second temps, les formules booléennes quantifiées.

Nous évoquons, lors du second chapitre de la première partie, quelques notions de complexité. Ce chapitre nous permet d'introduire le modèle des machines de Turing, utile pour définir la classe de complexité **PSPACE** qui est au cœur du sujet qui nous intéresse : le problème QBF. Nous décrivons dans ce chapitre quelques classes élémentaires de complexité et détaillons la hiérarchie polynomiale de manière à situer les restrictions de QBF obtenues en limitant le nombre d'alternances des quantificateurs.

La deuxième partie de ce mémoire constitue un état de l'art du travail qui a été accompli ces dernières années sur QBF puis plus particulièrement sur les classes polynomiales pour QBF.

Dans le troisième chapitre de ce document, nous décrivons très largement – tout en ayant conscience de ne pas être exhaustifs – différentes techniques mises au point à ce jour pour la résolution des formules booléennes quantifiées. Nous présentons une méthode de preuve pour QBF (la Q-résolution) et un algorithme (Q-DPLL) qui systématise la recherche de preuve par Q-résolution. Ensuite, nous passons en revue un certain nombre de règles de simplification conduisant à l'élagage de l'espace de recherche exploré par un tel algorithme. Puis, les heuristiques de branchement les plus classiques sont rapidement évoquées. Enfin, nous présentons le cadre expérimental utilisé depuis quelques années par la communauté des chercheurs intéressés par QBF pour évaluer les prouveurs QBF.

Après avoir introduit la notion de classe polynomiale pour QBF, nous nous penchons sur le travail accompli sur ces dernières dans le quatrième chapitre. Dans un premier temps, nous décrivons les techniques existantes de résolution des formules $2\mathcal{CNF}$ et des formules de Horn. Nous décrivons ensuite deux techniques de reconnaissance des formules Horn renommables quantifiées, ainsi qu'une adaptation à QBF d'un algorithme de résolution de cette classe de formules.

Enfin, les contributions apportées lors de cette thèse sont décrites dans la dernière partie.

Le chapitre 5 présente l'étude réalisée sur la complexité de QBF pour des fragments propositionnels pour la compilation de « connaissances » (Coste-Marquis *et al.* 2005b). Nous présentons les résultats que nous avons obtenus concernant la complexité des restrictions de QBF pour les principaux fragments propositionnels complets. Nous mettons ensuite en évidence un lien avec la notion de compilabilité.

Le sixième chapitre du mémoire est consacré à la résolution pratique de QBF. Dans celui-ci, nous présentons le prouveur QBF que nous avons développé lors de cette thèse. Une nouvelle heuristique de branchement (Coste-Marquis *et al.* 2005a; Letombe 2005) orientée vers la promotion des formules d'une classe polynomiale particulière – les formules Horn renommables quantifiées – est intégrée à ce prouveur. Quelques résultats expérimentaux sont finalement présentés.

Première partie

Préliminaires formels

1

Logique propositionnelle et formules booléennes quantifiées

Sommaire

« *Les préliminaires, c'est mettre le corps à l'ouvrage.* »
Jean-Loup Chiflet [3]

Dans ce chapitre, nous décrivons formellement dans un premier temps la logique propositionnelle et, dans un second temps, les formules booléennes quantifiées. Les aspects syntaxiques et sémantiques sont successivement présentés.

1.1 Logique propositionnelle

Nous décrivons dans ce paragraphe les aspects syntaxiques puis les aspects sémantiques propres à la logique propositionnelle.

3. Chiflet (Jean-Loup), écrivain et éditeur français, né en 1942. Passionné par les expressions idiomatiques, les nuances, les difficultés grammaticales et les aberrations de la langue française, Jean-Loup Chiflet (« John-Wolf Whistle ») est écrivain et éditeur. Son objectif, qui consiste à s'instruire en s'amusant, lui vaut un immense succès depuis des années. Il a écrit le fameux « Sky my husband ! », regard humoristique sur la traduction du français à l'anglais, « J'ai un mot à vous dire », qui met en scène un mot qui raconte sa vie, de sa naissance dans une clinique à l'occasion d'une opération du larynx jusqu'à sa mort, « Le Cafard laqué » ou « Le Mokimanké, le dictionnaire des mots qui existent enfin », après « Le Dictionnaire des mots qui n'existent pas » et « Mais que fait l'Académie ? Le Dictionnaire des mots qui devraient exister ». Ses livres, pleins d'humour et de finesse, étonnent et enchantent les amoureux de la langue, française ou étrangère. Jean-Loup Chiflet a créé sa propre maison d'édition : « Mots et Cie ».

1.1.1 Aspects syntaxiques

Soit PS un ensemble fini de *symboles propositionnels* appelés également *variables* ou *atomes*. Pour chaque sous-ensemble V de PS, $PROP_V$ désigne le *langage propositionnel* construit à partir des symboles de V, les constantes booléennes Vrai et Faux et les connecteurs logiques $\neg$ (la négation), $\wedge$ (la conjonction), $\vee$ (la disjonction), $\Rightarrow$ (l'implication), $\Leftrightarrow$ (l'équivalence), $\oplus$ (le ou exclusif, ou XOR). Les éléments de $PROP_{PS}$ sont appelés des **formules**.

Définition 1.1 ($PROP_V$)
Soit $V \subseteq PS$. $PROP_V$ est le plus petit ensemble de mots sur $V \cup \{\mathsf{Vrai}, \mathsf{Faux}, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \oplus\} \cup \{(,)\}$ défini inductivement par :
- $\mathsf{Vrai}, \mathsf{Faux} \in PROP_V$,
- *si $x \in V$ alors $x \in PROP_V$,*
- *si $\Sigma \in PROP_V$ alors $(\neg\Sigma) \in PROP_V$,*
- *si $\Sigma, \Psi \in PROP_V$ alors $(\Sigma \odot \Psi) \in PROP_V$ pour tout connecteur binaire $\odot \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow, \oplus\}$.*

La taille $|\Sigma|$ d'une formule Σ est le nombre de symboles (incluant les connecteurs et les constantes booléennes) utilisés pour l'écrire.

Pour simplifier les notations, on omet souvent les parenthèses qui sont inutiles compte tenu de l'associativité de certains connecteurs. Ainsi, $a \vee b \vee c$ est un raccourci pour $((a \vee b) \vee c)$; on considère également que $\neg$ est le connecteur le plus prioritaire ; ainsi, $\neg a \vee b$ est un raccourci pour $((\neg a) \vee b)$.

Définition 1.2 (Sous-formule)
Soit $\Sigma \in PROP_{PS}$. L'ensemble $SF(\Sigma)$ des **sous-formules** *de Σ est défini inductivement comme suit :*
- *si $\Sigma = \mathsf{Vrai}, \mathsf{Faux}$ ou $\Sigma \in PS$, alors $SF(\Sigma) = \{\Sigma\}$,*
- *si $\Sigma = \neg\Psi$, alors $SF(\Sigma) = \{\Sigma\} \cup SF(\Psi)$,*
- *si $\Sigma = \Psi \odot \Theta$ où $\odot \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow, \oplus\}$, alors $SF(\Sigma) = \{\Sigma\} \cup SF(\Psi) \cup SF(\Theta)$.*

Définition 1.3 (Littéral)
Un ***littéral*** *sur PS est une formule de la forme x (littéral positif) ou $\neg x$ (littéral négatif) avec $x \in PS$. Pour tout $V \subseteq PS$, $L_V = \{x, \neg x \mid x \in V\}$ est l'ensemble des littéraux sur V.*

Définition 1.4 (Littéral complémentaire)
Soit $l \in L_{PS}$. Le **littéral complémentaire** *l^c de l est le littéral de L_{PS} défini par :*
- *si $l = x \in PS$, alors $l^c = \neg x$,*
- *sinon, $l = \neg x$ et $l^c = x$.*

Définition 1.5 (Clause)
Une ***clause*** *est une disjonction finie de littéraux ou la constante* Faux. *Une clause ne contenant pas de littéraux complémentaires est dite fondamentale, sinon, elle est dite tautologique.*

Définition 1.6 (Terme)
Un ***terme*** *est une conjonction finie de littéraux ou la constante* Vrai. *Un terme ne contenant pas de littéraux complémentaires est dit fondamental.*

Exemple 1.1 (Littéral, clause, terme)
$a \vee b \vee \neg c$ (resp. $a \wedge b \wedge \neg c$) est une clause composée (resp. un terme composé) des littéraux positifs a et b et du littéral négatif $\neg c$. ■

Une formule **conjonctive** (resp. **disjonctive**) est la conjonction (resp. la disjonction) de plusieurs formules. Une conjonction de clauses (resp. disjonction de termes) est appelée formule sous **forme normale conjonctive** ou $\mathcal{CNF}$ (resp. formule sous **forme normale disjonctive** ou $\mathcal{DNF}$). $\mathcal{CNF}$ et $\mathcal{DNF}$ sont des cas spécifiques de **formes normales négatives** ou $\mathcal{NNF}$: une formule est dite sous $\mathcal{NNF}$ lorsque les seuls connecteurs binaires qu'elle contient sont $\wedge$ et $\vee$ et que la portée de chaque occurrence du connecteur de négation $\neg$ est réduite à un symbole propositionnel.

Définition 1.7 ($\mathcal{NNF}_V$)
$\mathcal{NNF}_V$ est le sous-ensemble de $PROP_V$ défini inductivement par :
- $\mathsf{Vrai}, \mathsf{Faux} \in \mathcal{NNF}_V$,
- *si $x \in V$, alors $x, \neg x \in \mathcal{NNF}_V$,*
- *si $\Sigma, \Psi \in \mathcal{NNF}_V$, alors $(\Sigma \wedge \Psi), (\Sigma \vee \Psi) \in \mathcal{NNF}_V$.*

Définition 1.8 (Var)
*Étant donnée une formule $\Sigma \in PROP_V$, on note **Var(**Σ**)** l'ensemble des variables propositionnelles apparaissant dans Σ. De plus, si l est un littéral, alors $|l|$ désigne la variable correspondante : $v = |l|$ si et seulement si $l = v$ ou $l = \neg v$.*

Définition 1.9 (Lit)
*Étant donnée une formule $\Sigma \in \mathcal{NNF}_{PS}$, on note **Lit(**$\Sigma$**)** l'ensemble des littéraux apparaissant comme sous-formule de Σ.*

Définition 1.10 (Clause positive, négative, mixte)
On appelle **clause positive** *(resp.* **clause négative***) une clause qui ne contient pas de littéraux négatifs (resp. positifs).*
Une clause constituée à la fois de littéraux positifs et négatifs est appelée **clause mixte***.*

Définition 1.11 (Littéral unitaire, monotone)
Soit Φ une formule de $PROP_{PS}$ sous forme normale conjonctive. Un littéral l de L_{PS} est :
- **unitaire** *dans Φ si l est une clause de Φ.*
- **monotone** *dans Φ si $l \in Lit(\Phi)$, et $l^c \notin Lit(\Phi)$.*

Définition 1.12 (Clause de Horn, clause reverse-Horn)
Une **clause de Horn** *(resp.* **clause reverse-Horn***) est une clause qui contient au plus un littéral positif (resp. négatif).*

Exemple 1.2 (Clauses de Horn)
$(\neg a \vee \neg b \vee d \vee \neg c)$ est une clause de Horn.
$(a \vee b \vee \neg d \vee c)$ est un clause reverse-Horn.
Les clauses négatives sont des clauses de Horn.
Les clauses positives sont des clauses reverse-Horn. ■

Pour la suite, une formule de Horn (resp. une formule reverse-Horn) $\mathcal{CNF}$ désigne une formule sous forme $\mathcal{CNF}$ composée uniquement de clauses de Horn (resp. de clauses reverse-Horn).

Définition 1.13 (Clause de Krom)
Une **clause de Krom** *est une clause qui contient au plus deux littéraux.*

Exemple 1.3 (Clause de Krom)
$(a \vee \neg b)$ est une clause de Krom. ■

On assimilera tout ensemble fini de formules avec la conjonction de ces dernières.

1.1.2 Aspects sémantiques

Définition 1.14 (Interprétation)
*Une **interprétation** I sur V est une application ayant pour domaine V et pour co-domaine $\mathbb{B} = \{0, 1\}$.*

Lorsque $V \subset PS$, on dit que l'interprétation est *partielle*, et l'on parle d'interprétation *totale* lorsque $V = PS$. Étant donnée une interprétation sur un ensemble de variables, on peut définir la sémantique de toute formule construite sur cet ensemble de variables.

Définition 1.15 (Sémantique d'une formule propositionnelle)
Étant donnée une interprétation I sur V et une formule propositionnelle $\Sigma \in PROP_{PS}$ telle que $Var(\Sigma) \subseteq V$, la **sémantique** *de Σ dans I (notée $[\![\Sigma]\!](I)$) est définie inductivement par :*
- *si $\Sigma = \mathsf{Vrai}$ (resp. Faux), alors $[\![\Sigma]\!](I) = 1$ (resp. 0).*
- *si $\Sigma \in PS$, alors $[\![\Sigma]\!](I) = I(\Sigma)$.*
- *si $\Sigma = \neg\phi$, alors $[\![\Sigma]\!](I) = 1 - [\![\phi]\!](I)$.*
- *si $\Sigma = \phi \wedge \psi$, alors $[\![\Sigma]\!](I) = [\![\phi]\!](I) \times [\![\psi]\!](I)$.*
- *si $\Sigma = \phi \vee \psi$, alors $[\![\Sigma]\!](I) = max(\{[\![\phi]\!](I), [\![\psi]\!](I)\})$.*
- *si $\Sigma = \phi \Rightarrow \psi$, alors $[\![\Sigma]\!](I) = [\![\neg\phi \vee \psi]\!](I)$.*
- *si $\Sigma = \phi \Leftrightarrow \psi$, alors $[\![\Sigma]\!](I) = [\![(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)]\!](I)$.*
- *si $\Sigma = \phi \oplus \psi$, alors $[\![\Sigma]\!](I) = [\![\neg(\phi \Leftrightarrow \psi)]\!](I)$.*

La sémantique d'une formule dans une interprétation I ne dépend que de la sémantique de ses sous-formules dans I (tous les connecteurs sont vérifonctionnels). En particulier, $[\![\Sigma]\!](I)$ ne dépend que de la restriction de I à $Var(\Sigma)$.

Définition 1.16 (Modèle)
*Une interprétation I sur V **satisfait** une formule $\Sigma \in PROP_V$ si $[\![\Sigma]\!](I) = 1$. On dit alors que I est un **modèle** de Σ, ce que l'on note $I \models \Sigma$.*
*Inversement, si $[\![\Sigma]\!](I) = 0$, on dit que I **falsifie** Σ et que I est un **contre-modèle** de Σ, ce qui se note $I \not\models \Sigma$.*

Définition 1.17 (Satisfiabilité)
*Une formule $\Sigma \in PROP_{PS}$ est dite **satisfiable** (ou* **cohérente***) si elle admet au moins un modèle.*

Définition 1.18 (Insatisfiabilité/Contradiction)
*Une formule $\Sigma \in PROP_{PS}$ est dite **insatisfiable** si elle n'admet aucun modèle. Dans ce cas, on dit aussi que Σ est **contradictoire**, ce qui est noté $\models \neg\Sigma$ ou $\Sigma \models \mathsf{Faux}$.*

Définition 1.19 (Formule valide)
Si toute interprétation I sur PS est un modèle de $\Sigma \in PROP_{PS}$, Σ est **valide***.*

Définition 1.20 (Conséquence logique)
Si tout modèle de $\Sigma \in PROP_{PS}$ est un modèle d'une formule $\Pi \in PROP_{PS}$, alors Π est une **conséquence logique** *de Σ, notée $\Sigma \models \Pi$.*

Définition 1.21 (Formules équivalentes)
Soient $\Sigma \in PROP_{PS}$ *et* $\Pi \in PROP_{PS}$*. Lorsque* $\Sigma \models \Pi$ *et* $\Pi \models \Sigma$*,* Σ *et* Π *sont* **équivalentes***, noté* $\Sigma \equiv \Pi$.

On peut montrer que pour tout $\Sigma \in PROP_{PS}$, il existe une formule sous forme $\mathcal{NNF}$ (resp. $\mathcal{CNF}$, $\mathcal{DNF}$) qui est logiquement équivalente à Σ.

1.2 Formules booléennes quantifiées

Dans cette partie, nous présentons la syntaxe et la sémantique des formules booléennes quantifiées ; nous donnons également quelques méta-théorèmes qui se montreront utiles dans les chapitres suivants.

1.2.1 Aspects syntaxiques

La morphologie du langage considéré comprend l'ensemble des connecteurs suivants : $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\oplus$, les deux constantes booléennes et les quantificateurs $\forall$ et $\exists$; sur cette base, un langage pour les formules booléennes quantifiées peut être défini comme suit :

Définition 1.22 ($QPROP_{PS}$**)**
L'ensemble $QPROP_{PS}$ *de* **formules booléennes quantifiées** *(aussi noté* $\mathcal{QBF}$*) sur l'ensemble fini de symboles propositionnels* PS *est le plus petit ensemble de mots défini inductivement comme suit :*

1. *les constantes booléennes* $\mathsf{Vrai} \in QPROP_{PS}$ *et* $\mathsf{Faux} \in QPROP_{PS}$.
2. *chaque variable de* $PS \in QPROP_{PS}$.
3. *si* $\phi \in QPROP_{PS}$ *et* $\psi \in QPROP_{PS}$*, alors* $(\neg\phi)$, $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \Rightarrow \psi)$, $(\phi \Leftrightarrow \psi)$, $(\phi \oplus \psi) \in QPROP_{PS}$.
4. *si* $\phi \in QPROP_{PS}$ *et* $x \in PS$*, alors* $(\forall x.\phi) \in QPROP_{PS}$ *et* $(\exists x.\phi) \in QPROP_{PS}$.

Afin de simplifier la syntaxe, nous nous permettons d'omettre des parenthèses lorsque ces dernières ne sont pas nécessaires. De plus, $\exists x.(\exists y.\phi)$ (resp. $\forall x.(\forall y.\phi)$) est souvent abrégé en $\exists x, y.\phi$ (resp. $\forall x, y.\phi$). Enfin, quand X est un ensemble de variables, on abrège $\forall x_1, \ldots, x_n.\phi$ (resp. $\exists x_1, \ldots, x_n.\phi$) en $\forall X.\phi$ (resp. $\exists X.\phi$).

Exemple 1.4 ($\mathcal{QBF}$**)**
La formule Σ suivante est une $\mathcal{QBF}$:

$$\Sigma = \exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee b)) \wedge \forall b.(a \vee ((\neg b) \wedge c)))$$

La figure 1.1 est une représentation graphique de Σ.

■

Les occurrences de variables propositionnelles x dans une formule Σ de $QPROP_{PS}$ peuvent être partitionnées en trois ensembles : les occurrences *quantifiées*, *liées* et *libres* de x. Les occurrences **quantifiées** sont celles apparaissant dans une quantification, i.e., juste après un quantificateur $\forall$ ou $\exists$. Dans toute sous-formule $\forall x.\phi$ (resp. $\exists x.\phi$) de Σ, toutes les occurrences de x dans ϕ sont **liées** ; de telles occurrences de x sont dites **dans la portée de la quantification** $\forall x$ (resp. $\exists x$). Enfin, toutes les occurrences restantes de x dans Σ sont **libres**.

Une variable $x \in PS$ est **libre** dans Σ si et seulement si x a une occurrence libre dans Σ.

Exemple 1.5 (Suite de l'exemple 1.4)
Considérons à nouveau $\Sigma = \exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee b)) \wedge \forall b.(a \vee ((\neg b) \wedge c)))$.

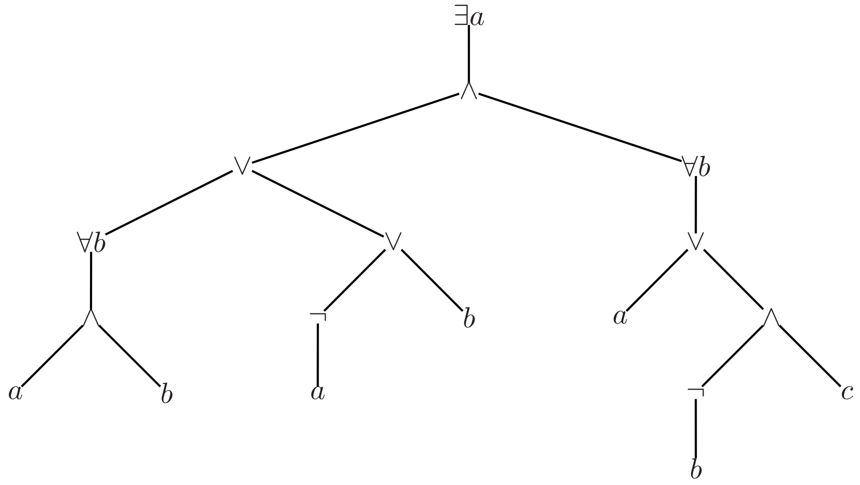


FIG. 1.1 – *Une formule booléenne quantifiée.*

La première occurrence de a dans Σ est quantifiée, la seconde occurrence de b dans Σ est liée et la troisième est libre. b et c sont les variables libres de Σ. ■

Une formule Σ est dite **polie** si et seulement si chaque occurrence liée d'une variable x dans Σ est dans la portée d'un quantificateur unique, et chaque variable libre n'a pas d'occurrence liée. Une formule Σ est dite sous forme **prénexe** si et seulement si $\Sigma = Qx_1(\ldots Qx_n(\phi)\ldots)$ où chaque occurrence de Q vaut soit $\forall$, soit $\exists$, et ϕ ne contient pas d'occurrence quantifiée de variable. ϕ est la **matrice** de Σ et la suite $Qx_1 \ldots Qx_n$ de quantifications est le **préfixe** de Σ. Qx_1 est la quantification **la plus externe** de Σ et Qx_n la quantification **la plus interne**. Une formule est dite **fermée** si et seulement si elle ne contient pas de variable libre. Une formule est dite **sans quantification** si et seulement si elle ne contient pas de quantification (de telles formules peuvent être considérées comme des formules propositionnelles « standard »). Aussi, le sous-ensemble de $QPROP_{PS}$ ne contenant que des formules sans quantification est ainsi $PROP_{PS}$.

Exemple 1.6 (Suite de l'exemple 1.4)
Σ est ni polie, ni fermée, ni prénexe. ■

Exemple 1.7 ($\mathcal{QBF}$ polie, fermée et prénexe)
La $\mathcal{QBF}$ suivante :

$$\Sigma = \exists\, a, b\, \forall\, c, d\, \exists\, e\, . \begin{bmatrix} (b \vee c \vee \neg d \vee e) & \wedge \\ (\neg a \vee \neg b \vee \neg e) & \wedge \\ (\neg a \vee d \vee e) & \wedge \\ (\neg c \vee e) & \wedge \\ (a \vee b \vee \neg c) & \wedge \\ (b \vee \neg e) & \wedge \\ (c \vee d \vee \neg e) & \end{bmatrix}$$

est polie, fermée, prénexe. ■

Définition 1.23 ($h\mathcal{QBF}$)
Soit h un entier. Une $h\mathcal{QBF}$ est une $\mathcal{QBF}$ polie, fermée, prénexe $Q_1X_1 \ldots Q_hX_h.\phi$ dans laquelle les quantificateurs sont appliqués à h ensembles disjoints de variables $X_1, \ldots, X_h$ et dans laquelle les quantificateurs sont alternés (pour tout $i \in 1 \ldots h-1, Q_i \neq Q_{i+1}$).

Parfois, nous ajoutons un indice à $h\mathcal{QBF}$ dénotant le type du quantificateur le plus externe de la formule.

Exemple 1.8 (3$\mathcal{QBF}_\exists$)
Soient X_1, X_2 et X_3 des ensembles disjoints de variables propositionnelles et ϕ une formule propositionnelle telle que $Var(\phi) = X_1 \cup X_2 \cup X_3$. La formule $\exists X_1\ \forall X_2\ \exists X_3.\phi$ est une $3\mathcal{QBF}_\exists$. ■

Définition 1.24 ($\mathcal{CNF}$-$\mathcal{QBF}$)
Une $\mathcal{CNF}$-$\mathcal{QBF}$ est une $\mathcal{QBF}$ polie, fermée, prénexe dont la matrice est une formule sous forme normale conjonctive.

Une $\mathcal{CNF}$-$\mathcal{QBF}$ est $k\mathcal{CNF}$-$\mathcal{QBF}$ si toutes les clauses de sa matrice ϕ contiennent au plus k littéraux. Il a été montré que cette forme de $\mathcal{CNF}$-$\mathcal{QBF}$ conserve la généralité et la difficulté du problème de la validité (Cadoli *et al.* 1998; Kleine-Büning *et al.* 1995; Stockmeyer & Meyer 1973).

Exemple 1.9 ($k\mathcal{CNF}$-$h\mathcal{QBF}$)
Soient $X = \{x\}$, $Y = \{y_1, y_2\}$, $W = \{w_1, w_2\}$ et $Z = \{z\}$.
La formule

$$\forall W\ \exists Z\ \forall Y\ \exists X. \begin{bmatrix} (y_1 \vee \neg y_2 \vee z) & \wedge \\ (w_1 \vee \neg w_2 \vee \neg z) & \wedge \\ (x \vee \neg w_1 \vee \neg y_2) & \end{bmatrix}$$

est une $3\mathcal{CNF}$-$4\mathcal{QBF}$. ■

Comme les connecteurs $\wedge$ et $\vee$ sont associatifs, commutatifs et idempotents pour les $\mathcal{QBF}$, on représente souvent la matrice d'une formule $\mathcal{CNF}$-$\mathcal{QBF}$ comme un ensemble de clauses et chaque clause comme un ensemble de littéraux. Si une clause α voit par conditionnement un de ses littéraux l remplacé par Vrai (resp. Faux), elle est supprimée de la matrice (resp. l est supprimé de l'ensemble associé à α). En particulier, l'ensemble vide de littéraux, appelé clause vide, représente la clause Faux et l'ensemble vide de clauses représente la formule sous forme normale conjonctive Vrai. Enfin, si α est une clause, alors $\alpha_\exists$ (resp. $\alpha_\forall$) désigne la clause α réduite à l'ensemble de ses littéraux existentiels (resp. universels).

On définit maintenant des classes génériques de formules booléennes quantifiées.

Définition 1.25 ($\mathcal{QC}$)
Soit $\mathcal{C} \subseteq PROP_{PS}$. On note $\mathcal{QC}$ l'ensemble des formules booléennes quantifiées polies, prénexes, fermées dont la matrice appartient à $\mathcal{C}$.

Ainsi, $\mathcal{QCNF}$ représente aussi la classe désignée par $\mathcal{CNF}$-$\mathcal{QBF}$.
Enfin, les notions qui suivent constituent de simples extensions des notions correspondantes en logique propositionnelle aux $\mathcal{QBF}$.

Définition 1.26 (Littéral existentiel, universel, unitaire, monotone)
Soit Σ une $\mathcal{CNF}$-$n\mathcal{QBF}$ de la forme $Q_1X_1 \ldots Q_nX_n.\phi$. Un littéral l de $Lit(\phi)$ est :
- **existentiel** *(resp.* **universel***) dans Σ si $\exists|l|$ (resp. $\forall|l|$) appartient au préfixe de Σ ; on dit aussi que l est un $\exists$-littéral (resp. $\forall$-littéral).*
- **unitaire** *dans Σ si l est existentiel et, pour tout $k \geq 1$,*
  - *il existe une clause équivalente à $l \vee l_1 \vee \cdots \vee l_k$ dans ϕ, et*
  - *pour tout $i \in 1 \ldots k$, toute occurrence quantifiée $\forall|l_i|$ est plus interne que $\exists|l|$ dans le préfixe de Σ.*

– **monotone** *(ou* **pur***) si,*
  – *soit l est existentiel, $l \in Lit(\Sigma)$, et $l^c \notin Lit(\Sigma)$;*
  – *soit l est universel, $l \notin Lit(\Sigma)$, et $l^c \in Lit(\Sigma)$.*

Exemple 1.10 (Littéral existentiel, universel, unitaire, monotone)
Considérons la $\mathcal{QBF}$ de l'exemple 1.9. Les littéraux y_1, y_2, w_1, et w_2 et leurs négations respectives sont universels alors que les littéraux x et z et leurs négations sont existentiels.
Le littéral existentiel z est unitaire par la première clause $(y_1 \vee \neg y_2 \vee z)$.
Enfin, le littéral existentiel x est monotone, ainsi que le littéral universel w_2. ■

Définition 1.27 ($\forall$-clause pure)
Soit Σ une $\mathcal{CNF}$-$\mathcal{QBF}$. Une **$\forall$-clause pure** *de Σ est une clause non tautologique de la matrice composée uniquement de $\forall$-littéraux dans Σ.*

En particulier, la clause vide est une $\forall$-clause pure.

1.2.2 Aspects sémantiques

Considérons à présent les aspects sémantiques des $\mathcal{QBF}$; commençons par la notation utile de conditionnement suivante (également appelée restriction ou cofacteur (Bryant 1986)). Pour toute formule booléenne quantifiée Σ et toute variable x, $\Sigma_{x\leftarrow 0}$ (resp. $\Sigma_{x\leftarrow 1}$) désigne la $\mathcal{QBF}$ obtenue en remplaçant toute occurrence libre de x dans Σ par Faux (resp. Vrai). Formellement :

Définition 1.28 (Conditionnement)
Soit $\Sigma \in QPROP_{PS}$, $x \in PS$ et $* \in \{1, 0\}$. Le **conditionnement** *de x par $*$ dans Σ est la $\mathcal{QBF}$ définie inductivement comme suit :*

$$\Sigma_{x\leftarrow *} = \begin{cases} si\ \Sigma = x\ et\ * = 1\ alors\ \mathsf{Vrai} \\ si\ \Sigma = x\ et\ * = 0\ alors\ \mathsf{Faux} \\ si\ \Sigma \in PS\ et\ \Sigma \neq x\ alors\ \Sigma \\ si\ \Sigma = \neg\phi\ alors\ \neg(\phi_{x\leftarrow *}) \\ si\ \Sigma = \phi \odot \psi\ alors\ \phi_{x\leftarrow *} \odot \psi_{x\leftarrow *}\ avec\ \odot\ \text{\textit{un connecteur logique binaire}} \\ si\ \Sigma = \forall x.\phi\ ou\ \Sigma = \exists x.\phi\ alors\ \Sigma \\ si\ \Sigma = \forall y.\phi\ avec\ y \neq x\ alors\ \forall y.(\phi_{x\leftarrow *}) \\ si\ \Sigma = \exists y.\phi\ avec\ y \neq x\ alors\ \exists y.(\phi_{x\leftarrow *}) \end{cases}$$

Exemple 1.11 (Suite de l'exemple 1.4)
Considérons à nouveau $\Sigma = \exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee b)) \wedge \forall b.(a \vee ((\neg b) \wedge c)))$.
Le conditionnement de b par 1 dans Σ est la $\mathcal{QBF}$

$$\Sigma_{b\leftarrow 1} = \exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee \mathsf{Vrai})) \wedge \forall b.(a \vee ((\neg b) \wedge c)))$$

■

On étend facilement la notion de conditionnement à des littéraux : si $\Sigma \in QPROP_{PS}$ et $l \in L_{PS}$, alors

$$\Sigma_{l\leftarrow 1} = \begin{cases} \Sigma_{|l|\leftarrow 1} \text{ si } l \text{ est un littéral positif} \\ \Sigma_{|l|\leftarrow 0} \text{ sinon} \end{cases}$$

et

$$\Sigma_{l\leftarrow 0} = \begin{cases} \Sigma_{|l|\leftarrow 0} \text{ si } l \text{ est un littéral positif} \\ \Sigma_{|l|\leftarrow 1} \text{ sinon} \end{cases}$$

Il est aisé de prouver que le conditionnement de x par $*$ dans Σ peut être construit en temps linéaire en $|\Sigma|$. De plus, on peut montrer facilement par induction structurelle que des conditionnements successifs commutent : pour tout $\Sigma \in QPROP_{PS}$, tout $x, x' \in PS$ et $*, *' \in \{0, 1\}$, nous avons

$$(\Sigma_{x \leftarrow *})_{x' \leftarrow *'} \equiv (\Sigma_{x' \leftarrow *'})_{x \leftarrow *}.$$

Nous utiliserons aussi la convention d'écriture suivante :

Définition 1.29 (Σ_S)
Soit $\Sigma \in QPROP_{PS}$ et S un sous-ensemble cohérent de L_{PS} de la forme $\{x_1, \ldots, x_n, \neg x'_1, \ldots, \neg x'_m\}$. On utilisera la notation Σ_S comme abréviation de $(((\Sigma_{x_1 \leftarrow 1}) \cdots_{x_n \leftarrow 1})_{x'_1 \leftarrow 0}) \cdots_{x'_m \leftarrow 0}$.

Nous pouvons à présent définir la sémantique d'une $\mathcal{QBF}$:

Définition 1.30 (Sémantique d'une formule booléenne quantifiée)
Soit I une interprétation sur PS (i.e., une fonction de PS vers $\mathcal{B} = \{0, 1\}$). La sémantique d'une formule booléenne quantifiée Σ dans I est la valeur de vérité $[\![\Sigma]\!](I)$ de $\mathcal{B}$ définie inductivement comme suit :
- *si $\Sigma =$ Vrai (resp. Faux), alors $[\![\Sigma]\!](I) = 1$ (resp. 0).*
- *si $\Sigma \in PS$, alors $[\![\Sigma]\!](I) = I(\Sigma)$.*
- *si $\Sigma = \neg\phi$, alors $[\![\Sigma]\!](I) = 1 - [\![\phi]\!](I)$.*
- *si $\Sigma = \phi \wedge \psi$, alors $[\![\Sigma]\!](I) = [\![\phi]\!](I) \times [\![\psi]\!](I)$.*
- *si $\Sigma = \phi \vee \psi$, alors $[\![\Sigma]\!](I) = max(\{[\![\phi]\!](I), [\![\psi]\!](I)\})$.*
- *si $\Sigma = \phi \Rightarrow \psi$, alors $[\![\Sigma]\!](I) = [\![\neg\phi \vee \psi]\!](I)$.*
- *si $\Sigma = \phi \Leftrightarrow \psi$, alors $[\![\Sigma]\!](I) = [\![(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)]\!](I)$.*
- *si $\Sigma = \phi \oplus \psi$, alors $[\![\Sigma]\!](I) = [\![\neg(\phi \Leftrightarrow \psi)]\!](I)$.*
- *si $\Sigma = \forall x.\phi$, alors $[\![\Sigma]\!](I) = min(\{[\![\phi_{x \leftarrow 0}]\!](I), [\![\phi_{x \leftarrow 1}]\!](I)\})$.*
- *si $\Sigma = \exists x.\phi$, alors $[\![\Sigma]\!](I) = max(\{[\![\phi_{x \leftarrow 0}]\!](I), [\![\phi_{x \leftarrow 1}]\!](I)\})$.*

Cette définition montre clairement que le langage des $\mathcal{QBF}$ constitue une extension conservative du langage des formules propositionnelles (la définition 1.15 s'applique aussi pour définir la sémantique de toute $\mathcal{QBF}$ sans quantification, i.e. des formules propositionnelles) ce qui explique la coïncidence des notations.

Les notions sémantiques utilisées en logique propositionnelle s'appliquent encore. Ainsi, une interprétation I est appelée un **modèle** de $\Sigma \in \mathcal{QBF}$, noté $I \models \Sigma$, si et seulement si $[\![\Sigma]\!](I) = 1$. Si Σ a un modèle, elle est **satisfiable** ; sinon, elle est **insatisfiable**. Si toute interprétation I sur PS est un modèle de Σ, Σ est **valide**, noté $\models \Sigma$. Si tout modèle de Σ est un modèle de $\mu \in \mathcal{QBF}$, alors μ est une **conséquence logique** de Σ, notée $\Sigma \models \mu$. Enfin, lorsque à la fois $\Sigma \models \mu$ et $\mu \models \Sigma$, Σ et μ sont **équivalents**, noté $\Sigma \equiv \mu$.

Il n'est pas difficile de prouver (encore par induction) que la sémantique de toute formule booléenne quantifiée Σ dépend uniquement de ses variables libres, dans le sens où, pour toute interprétation J sur PS qui coïncide avec une interprétation I donnée sur toutes les variables libres de Σ, alors I est un modèle de Σ si et seulement si J est un modèle de Σ. Notamment, la sémantique d'une formule fermée est la même dans toute interprétation. **Clairement, une telle formule est équivalente à une des constantes booléennes Vrai ou Faux, ainsi elle est satisfiable si et seulement si elle est valide.**

Comme en logique propositionnelle, on a $\Sigma \models \mu$ si et seulement si la formule $(\Sigma \wedge \neg\mu)$ est incohérente si et seulement si la formule $(\Sigma \Rightarrow \mu)$ est valide. Plus généralement, puisque les connecteurs sont vérifonctionnels, un méta-théorème de substitution existe pour les formules booléennes quantifiées : remplacer toute sous-formule par une sous-formule équivalente préserve l'équivalence par rapport à la formule initiale.

D'autres méta-théorèmes intéressants sont :

Proposition 1.1
Soient Σ, Ψ des formules de $QPROP_{PS}$ et x, y des variables de PS.

1. $\forall x.\Sigma \equiv \Sigma_{x\leftarrow 0} \wedge \Sigma_{x\leftarrow 1}$.
2. $\exists x.\Sigma \equiv \Sigma_{x\leftarrow 0} \vee \Sigma_{x\leftarrow 1}$.
3. $\forall x.\Sigma \equiv \neg(\exists x.(\neg\Sigma))$.
4. Si x n'est pas libre dans Σ, alors $\forall x.\Sigma \equiv \exists x.\Sigma \equiv \Sigma$.
5. $\forall x.(\Sigma \wedge \Psi) \equiv (\forall x.\Sigma) \wedge (\forall x.\Psi)$.
6. $\exists x.(\Sigma \vee \Psi) \equiv (\exists x.\Sigma) \vee (\exists x.\Psi)$.
7. $\forall x.(\forall y.\Sigma) \equiv \forall y.(\forall x.\Sigma)$.
8. $\exists x.(\exists y.\Sigma) \equiv \exists y.(\exists x.\Sigma)$.
9. Si x n'est pas libre dans Σ, alors $\forall x.(\Sigma \vee \Psi) \equiv \Sigma \vee (\forall x.\Psi)$.
10. Si x n'est pas libre dans Σ, alors $\exists x.(\Sigma \wedge \Psi) \equiv \Sigma \wedge (\exists x.\Psi)$.

Preuve 1.1
Ces méta-théorèmes sont tellement bien connus qu'il est difficile d'en trouver l'origine ; dans un souci de complétude, nous en donnons des preuves :

1. Immédiate depuis la définition 1.30.
2. Idem.
3. $\forall x.\Sigma \equiv \Sigma_{x\leftarrow 0} \wedge \Sigma_{x\leftarrow 1}$ selon le point 1.
   $\equiv \neg(\neg(\Sigma_{x\leftarrow 0}) \vee \neg(\Sigma_{x\leftarrow 1}))$ selon les lois de De Morgan
   $\equiv \neg((\neg\Sigma)_{x\leftarrow 0} \vee (\neg\Sigma)_{x\leftarrow 1}))$ selon la définition du conditionnement
   $\equiv \neg(\exists x.(\neg\Sigma))$ selon le point 2.
4. Si x n'est pas libre dans Σ, alors, selon la définition du conditionnement, nous avons $\Sigma_{x\leftarrow 0} \equiv \Sigma_{x\leftarrow 1} \equiv \Sigma$. Alors les points 1. et 2. donnent le résultat.
5. $\forall x.(\Sigma \wedge \Psi) \equiv (\Sigma \wedge \Psi)_{x\leftarrow 0} \wedge (\Sigma \wedge \Psi)_{x\leftarrow 1}$ selon le point 1.
   $\equiv (\Sigma_{x\leftarrow 0} \wedge \Psi_{x\leftarrow 0}) \wedge (\Sigma_{x\leftarrow 1} \wedge \Psi_{x\leftarrow 1})$
   selon la définition du conditionnement
   $\equiv \Sigma_{x\leftarrow 0} \wedge \Sigma_{x\leftarrow 1} \wedge \Psi_{x\leftarrow 0} \wedge \Psi_{x\leftarrow 1}$
   selon l'associativité et la commutativité de $\wedge$
   $\equiv (\forall x.\Sigma) \wedge (\forall x.\Psi)$ selon le point 1.
6. Directe à partir des points 3. et 5.
7. $\forall x.(\forall y.\Sigma) \equiv \forall x.(\Sigma_{y\leftarrow 0} \wedge \Sigma_{y\leftarrow 1})$ selon le point 1.
   $\equiv (\Sigma_{y\leftarrow 0} \wedge \Sigma_{y\leftarrow 1})_{x\leftarrow 0} \wedge (\Sigma_{y\leftarrow 0} \wedge \Sigma_{y\leftarrow 1})_{x\leftarrow 1}$ selon le point 1.
   $\equiv ((\Sigma_{y\leftarrow 0})_{x\leftarrow 0} \wedge (\Sigma_{y\leftarrow 1})_{x\leftarrow 0}) \wedge ((\Sigma_{y\leftarrow 0})_{x\leftarrow 1} \wedge (\Sigma_{y\leftarrow 1})_{x\leftarrow 1})$
   selon la définition du conditionnement
   $\equiv (\Sigma_{x\leftarrow 0})_{y\leftarrow 0} \wedge (\Sigma_{x\leftarrow 0})_{y\leftarrow 1} \wedge (\Sigma_{x\leftarrow 1})_{y\leftarrow 0} \wedge (\Sigma_{x\leftarrow 1})_{y\leftarrow 1}$
   par la commutativité des conditionnements successifs
   ainsi que l'associativité et la commutativité de $\wedge$
   $\equiv (\forall y.\Sigma_{x\leftarrow 0}) \wedge (\forall y.\Sigma_{x\leftarrow 1})$ selon les points 5. et 1.
   $\equiv \forall y.(\forall x.\Sigma)$ selon les points 5. et 1.
8. Directe à partir des points 3. and 7.
9. $\forall x.(\Sigma \vee \Psi) \equiv (\Sigma \vee \Psi)_{x\leftarrow 0} \wedge (\Sigma \vee \Psi)_{x\leftarrow 1}$ selon le point 1.
   $\equiv (\Sigma_{x\leftarrow 0} \vee \Psi_{x\leftarrow 0}) \wedge (\Sigma_{x\leftarrow 1} \vee \Psi_{x\leftarrow 1})$

selon la définition du conditionnement
$\equiv (\Sigma \vee \Psi_{x\leftarrow 0}) \wedge (\Sigma \vee \Psi_{x\leftarrow 1})$
car $\Sigma \equiv \Sigma_{x\leftarrow 0} \equiv \Sigma_{x\leftarrow 1}$ quand x n'est pas libre dans Σ
$\equiv \Sigma \vee (\Psi_{x\leftarrow 0} \wedge \Psi_{x\leftarrow 1})$ selon la distributivité de $\vee$ sur $\wedge$
$\equiv \Sigma \vee (\forall x.\Psi)$ selon le point 1.

10. Directe à partir des points 3. and 9.

□

Les points 1. et 2. montrent que toute formule booléenne quantifiée Σ peut être transformée en une formule propositionnelle « standard », mais la transformation suggérée (en considérant les équivalences comme des règles de réécriture orientées de gauche à droite) ne peut être achevée en espace polynomial (et plus généralement, il est très peu probable qu'une fonction de $QPROP_{PS}$ vers $PROP_{PS}$ qui préserve l'équivalence existe, puisque cela provoquerait l'effondrement de la hiérarchie polynomiale **HP**, cf. paragraphe 2.3). Par conséquent, il est vraisemblable que $QPROP_{PS}$ permet un encodage de l'information bien plus compact que $PROP_{PS}$.

Le point 3. montre que les quantificateurs universels et existentiels sont duaux.

Le point 4. donne une condition suffisante pour l'élimination des quantifications (la condition n'est pas nécessaire comme le montre l'exemple $\Sigma = x \vee \neg x$).

Les points 5., 6., 9. et 10. précisent l'interaction entre les quantificateurs et les connecteurs $\wedge$ et $\vee$.

Les points 7. et 8. montrent qu'il est possible d'inverser deux quantifications successives de la même nature dans une $\mathcal{QBF}$ Σ sans effet sur l'équivalence et justifient ainsi a posteriori l'abréviation $\forall X$ (avec X un ensemble de variables) que nous avons adoptée.

A contrario, il n'est pas possible d'inverser deux quantifications successives de natures différentes tout en préservant la sémantique. Ainsi, $\forall y.(\exists x.\Sigma)$ est une conséquence logique de $\exists x.(\forall y.\Sigma)$ mais les deux formules ne sont pas équivalentes (considérer par exemple $\Sigma = x \Leftrightarrow y$). De plus, dans le cas général, nous n'avons ni $\forall x.(\Sigma \vee \Psi) \equiv (\forall x.\Sigma) \vee (\forall x.\Psi)$, ni $\exists x.(\Sigma \wedge \Psi) \equiv (\exists x.\Sigma) \wedge (\exists x.\Psi)$ ($\Sigma = x$ et $\Psi = \neg x$ est un contre-exemple).

En se basant sur les méta-théorèmes donnés en proposition 1.1 et le méta-théorème de substitution, il est facile de prouver que toute formule de $QPROP_{PS}$ peut être transformée en une formule prénexe et polie équivalente en temps polynomial (les variables liées peuvent être renommées sans remettre en cause l'équivalence). Notons que diverses stratégies pour rendre une $\mathcal{QBF}$ prénexe existent (dépendantes de la manière dont les quantifications sont placées), et que le choix d'une $\mathcal{QBF}$ prénexe équivalente à une $\mathcal{QBF}$ Σ lorsque plusieurs sont possibles peut avoir un impact pratique sur l'efficacité de la résolution de Σ pour de nombreux prouveurs QBF (Egly *et al.* 2003).

Exemple 1.12 (Suite de l'exemple 1.4)
Considérons à nouveau $\Sigma = \exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee b)) \wedge \forall b.(a \vee ((\neg b) \wedge c)))$.
Σ est équivalente à la $\mathcal{QBF}$ prénexe, polie :

$$\exists a.(\forall u.(\forall v.(((a \wedge u) \vee ((\neg a) \vee b)) \wedge (a \vee ((\neg v) \wedge c))))).$$

u et v sont les nouvelles variables utilisées à rendre la formule polie. ■

Enfin, il existe des relations étroites entre les formules booléennes quantifiées (générales), les formules fermées, et les formules sans quantification (i.e., les formules propositionnelles « standard »). Nous avons notamment :

Proposition 1.2
Soit Σ une formule de $QPROP_{PS}$ et x une variable de PS.

1. Σ est satisfiable si et seulement si la formule fermée $\exists Var(\Sigma).(\Sigma)$ est valide.

2. Σ est valide si et seulement si la formule fermée $\forall Var(\Sigma).(\Sigma)$ est valide.

Preuve 1.2

1. Selon la définition 1.30, on obtient immédiatement que pour toute $\Sigma \in QPROP_{PS}$ et $x \in PS$, nous avons $\Sigma \models \exists x.\Sigma$. En conséquence, nous avons aussi $\Sigma \models \exists Var(\Sigma).\Sigma$. Donc si Σ est satisfiable, $\exists Var(\Sigma).\Sigma$ est satisfiable également. Puisque $\exists Var(\Sigma).\Sigma$ ne contient pas de variable libre, elle est satisfiable si et seulement si elle est valide.
   Réciproquement, si Σ est insatisfiable, alors $\Sigma \equiv \mathsf{Faux}$. Grâce au méta-théorème de substitution, on obtient que $\exists Var(\Sigma).\Sigma$ est équivalente à $\exists Var(\Sigma).\mathsf{Faux}$, qui est équivalente à Faux également, étant donnés le point 2. de la proposition 1.1 et la définition du conditionnement.
2. Directe à partir du point 1. ci-dessus, du point 3. de la proposition 1.1 et le fait que $\exists Var(\Sigma).\Sigma$ ne contient pas de variable libre.

Les points 1. et 2. montrent que le problème de satisfiabilité (resp. le problème de validité) des formules booléennes quantifiées (générales) peut être réduit en temps polynomial au problème de validité des formules booléennes quantifiées fermées. En particulier, le problème de satisfiabilité d'une formule propositionnelle « standard » Σ peut être réduit en temps polynomial au problème de validité de la formule booléenne quantifiée fermée $\exists Var(\Sigma).(\Sigma)$.

Exemple 1.13 (Suite de l'exemple 1.4)
Considérons à nouveau $\Sigma = \exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee b)) \wedge \forall b.(a \vee ((\neg b) \wedge c)))$.
Σ est satisfiable puisque

$$\exists a, b, c.(\exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee b)) \wedge \forall b.(a \vee ((\neg b) \wedge c))))$$

qui peut être simplifiée en

$$\exists b, c.(\exists a.(((\forall b.(a \wedge b)) \vee ((\neg a) \vee b)) \wedge \forall b.(a \vee ((\neg b) \wedge c))))$$

puisque a n'est pas libre dans Σ, est une $\mathcal{QBF}$ valide, fermée. ■

Pour cette raison, le problème QBF est généralement défini comme suit.

Définition 1.31 (QBF)
QBF *est le problème de décision suivant :*
– **Entrée :** *Une formule sous forme prénexe, polie, fermée* Σ *de* $QPROP_{PS}$ *;*
– **Question :** Σ *est-elle valide ?*

Exemple 1.14 (Suite de l'exemple 1.4)
Σ est satisfiable si et seulement si l'instance de QBF suivante est valide :

$$\exists a, b, c\, \forall u, v.(((a \wedge u) \vee ((\neg a) \vee b)) \wedge (a \vee ((\neg v) \wedge c)))$$

■

Définition 1.32 (QC)
Soit $\mathcal{QC}$ *un ensemble de formules booléennes quantifiées polies, prénexes, fermées. On note* QC *la restriction de* QBF *aux formules* $\mathcal{QC}$.

On observe que le test de validité d'une $1\mathcal{QBF}_{\exists}$ et le test de satisfiabilité SAT d'une formule propositionnelle sont un seul et même test.

2

Notions de complexité

Sommaire

« *La complexité ne donne pas de la valeur aux choses, elle les rend seulement moins accessibles.* »
Faya Dequoy

Dans ce chapitre, nous présentons les bases de la théorie de la complexité. Nous traitons dans un premier temps des machines de Turing. Nous définissons ensuite quelques classes de complexité. Enfin, nous présentons la hiérarchie polynomiale.

2.1 Machines de Turing

Afin de définir la complexité computationnelle, il convient au préalable de définir un modèle de calcul. Nous esquissons dans la suite le célèbre modèle des *machines de Turing* qui, bien que simple et primitif, est capable d'exprimer toute procédure de calcul.

2.1.1 Définitions

Une machine de Turing est un modèle abstrait du fonctionnement d'un ordinateur et de sa mémoire, créé par Alan Turing en vue de donner une définition précise au concept d'algorithme ou « procédure mécanique ». Ce modèle est toujours largement utilisé en informatique théorique, en particulier pour résoudre les problèmes de complexité algorithmique et de calculabilité.

La thèse de Church-Turing postule que tout procédé de calcul de type algorithmique correspond à une machine de Turing. Cette thèse n'est pas un énoncé mathématique, puisqu'elle ne suppose pas une définition précise de la notion de procédé de calcul. En revanche, il est possible de définir une notion de « système acceptable de programmation » et de démontrer que le pouvoir de tels systèmes est équivalent à celui des machines de Turing (Turing-complet). Ainsi, un langage de programmation est dit Turing-complet s'il permet de représenter toutes les fonctions calculables au sens de Turing (nonobstant le caractère fini de la mémoire des ordinateurs actuels). La plupart des langages usuels de programmation (C, C++, Java, ...) sont Turing-complets.

Il existe différentes définitions du modèle des machines de Turing (qui varient par exemple en fonction du nombre de rubans retenus). Le choix d'une définition peut avoir un impact sur ce que recouvrent certaines classes de complexité, mais pas celles considérées dans les chapitres suivants (les définitions usuelles caractérisent des machines qui peuvent se simuler mutuellement en temps polynomial). Voici la définition que nous avons retenue :

Définition 2.1 (Machine de Turing)
Une **machine de Turing** *est la donnée de :*

1. *Un « ruban » divisé en cases consécutives. Chaque case contient un symbole parmi un alphabet fini. L'alphabet contient un symbole spécial « blanc » ('0' dans les exemples qui suivent), et un ou plusieurs autres symboles. Le ruban est supposé être de longueur infinie vers la gauche ou vers la droite, en d'autres termes la machine doit toujours avoir assez de longueur de ruban pour son exécution. On considère que les cases non encore écrites du ruban contiennent le symbole « blanc ».*
2. *Une « tête de lecture/écriture » qui peut lire et écrire les symboles sur le ruban, et se déplacer vers la gauche ou vers la droite du ruban.*
3. *Un « registre d'état » qui mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est toujours fini, et il existe un état spécial appelé « état de départ » qui est l'état initial de la machine avant son exécution.*
4. *Une « table d'actions » qui indique quel symbole écrire, comment déplacer la tête de lecture ('G' pour une case vers la gauche, 'D' pour une case vers la droite), et quel est le nouvel état, en fonction du symbole lu sur le ruban et de l'état courant de la machine. Quand la machine n'est pas déterministe, plusieurs triplets $\langle$symbole à écrire, mouvement de la tête, nouvel état$\rangle$ résultants sont éventuellement possibles. Si aucune action n'existe pour une combinaison donnée d'un symbole lu et d'un état courant, la machine s'arrête.*

Exemple 2.1 (Machine de Turing)
La machine de Turing M qui suit possède un alphabet '0', '1', '0' étant le « blanc ». On suppose que le ruban contient au départ une suite finie m de '1', et que la tête de lecture/écriture repère initialement le '1' le plus à gauche. Cette machine a pour effet de doubler le nombre de '1', en intercalant un '0' entre les deux séries. Par exemple, « 111 » devient « 1110111 ».

L'ensemble d'états possibles de la machine est e1, e2, e3, e4, e5 et l'état initial est e1.

La table d'actions est donnée en table 2.1.

L'exécution de cette machine sur un ruban initialisé à « 11 » pourrait être par exemple telle qu'elle est décrite en table 2.2.

Le comportement de M peut être décrit comme suit : la machine démarre son exécution dans l'état e1, remplace le premier 1 par un 0, puis utilise l'état e2 pour se déplacer vers la droite, en « sautant » les 1, et le premier 0 qu'elle rencontre. L'état e3 est alors utilisé pour « sauter » la séquence suivante de 1 (initialement aucun) et remplacer le premier 0 rencontré par un 1. e4 permet de revenir vers la gauche jusqu'à trouver un 0, et passer dans l'état e5. e5 permet ensuite à nouveau de se déplacer vers la gauche

Ancien état	Symbole lu		Symbole écrit	Mouvement	Nouvel état
e1	1	→	0	D	e2
e2	1	→	1	D	e2
e2	0	→	0	D	e3
e3	0	→	1	G	e4
e3	1	→	1	D	e3
e4	1	→	1	G	e4
e4	0	→	0	G	e5
e5	1	→	1	G	e5
e5	0	→	1	D	e1

TAB. 2.1 – *Table d'actions de la machine de Turing M.*

Étape	État	Ruban	Étape	État	Ruban
1	e1	**1**1	9	e2	10**0**1
2	e2	0**1**	10	e3	100**1**
3	e2	01**0**	11	e3	1001**0**
4	e3	010**0**	12	e4	100**1**1
5	e4	01**0**1	13	e4	10**0**11
6	e5	0**1**01	14	e5	1**0**011
7	e5	**0**101	15	e1	11**0**11
8	e1	1**1**01	– STOP –		

TAB. 2.2 – *Trace de l'exécution de M sur un ruban initialisé à « 11 ». La case en caractères gras indique la position de la tête de lecture/écriture.*

jusqu'à trouver un 0, écrit quand la machine est dans l'état e1. La machine remplace alors ce 0 par un 1, se déplace d'une case vers la droite et passe à nouveau dans l'état e1 pour une nouvelle itération.

Ce processus se répète jusqu'à ce que e1 « tombe » sur un 0 (c'est le 0 du milieu entre les deux suites de 1). À ce moment là, la machine s'arrête. Le résultat $M(m)$ du calcul de M sur l'entrée m est le mot figurant sur le ruban lorsque M s'arrête. Le temps de calcul est le nombre de transitions réalisées et l'espace de calcul le nombre de cases utilisées sur le ruban. ■

2.1.2 Machines de Turing universelles

Toute machine de Turing calcule le résultat d'une fonction partielle sur des mots composés à partir des caractères de son alphabet. En ce sens, une machine de Turing se comporte comme un ordinateur avec un programme déterminé. Mais on peut aussi encoder la table d'actions d'une machine de Turing sous la forme d'un mot. On peut donc construire une machine de Turing qui, à partir d'un mot encodant une machine M et d'un mot m constituant les données effectives de M, calcule $M(m)$. Comme Alan Turing l'a montré, il est possible de créer une telle machine de Turing et, puisqu'elle peut simuler le comportement de n'importe quelle autre machine de Turing, on l'appelle « machine de Turing universelle ». Une machine de Turing universelle n'est en fait rien d'autre que la contrepartie formelle de la notion d'interpréteur d'un langage de programmation (soit une procédure de calcul qui émule une autre procédure de calcul).

Grâce à cet encodage des machines par des mots, il devient en principe possible que les machines de Turing répondent à des questions à propos du comportement d'autres machines de Turing. Cependant, la plupart de ces questions sont indécidables, c'est-à-dire que la fonction en question ne peut pas être calculée par une machine de Turing. Par exemple, la question de savoir si une machine de Turing atteint à un moment donné un état d'arrêt ou ne l'atteint jamais pour une entrée particulière, connue sous le nom de problème de l'arrêt, fut démontrée comme étant indécidable par Turing. Le théorème de Rice montre que toute question non triviale sur le comportement ou la sortie d'une machine de Turing est indécidable.

2.1.3 Autres modèles de calcul

Dans la suite de ce mémoire, on considère plusieurs modèles de calcul s'appuyant sur le modèle de la machine de Turing.

Le premier est une spécialisation du modèle présenté et concerne les machines de Turing à états d'arrêts oui/non : on suppose que l'ensemble des états de la machine contient deux états particuliers oui et non, à partir desquels aucune transition n'est possible, et que ce sont les deux seuls états vérifiant cette propriété. On définit alors la sortie $M(m)$ d'une telle machine sur un mot d'entrée m par l'état obtenu lorsque la machine s'arrête. Ce type de machine est adapté à la résolution de problèmes de décision, autrement dit au codage de fonction à valeur booléenne.

Le second modèle considéré est une généralisation du modèle des machines de Turing et concerne les machines de Turing à oracle . Un oracle pour un langage L est un élément du dispositif de calcul de la machine (un ruban supplémentaire et trois états spécifiques) permettant de décider pour tout mot m (une fois écrit sur le ruban supplémentaire) en une étape de calcul si $m \in L$ ou pas.

Enfin, on décrit au paragraphe 5.3 les machines de Turing à avis qui généralisent aussi le modèle standard.

2.2 Théorie de la complexité

2.2.1 Classes de complexité

En théorie de la complexité (cf. e.g. (Papadimitriou 1994)), un problème est formalisé de la manière suivante : un ensemble de données en entrée, et une question sur ces données. La théorie de la complexité traite surtout des problèmes de décision, c'est-à-dire posant une question dont la réponse est soit oui, soit non.

Un problème de décision est usuellement formalisé comme un langage, l'ensemble des mots d'entrée pour lesquels la réponse à la question posée est oui.

Différentes classes de complexité regroupent des problèmes de décision selon la capacité qu'a une machine de Turing (plus ou moins sophistiquée) d'en permettre la résolution en consommant plus ou moins de ressources (temps et espace) :

- Classe **L** : un problème de décision est dans **L** s'il peut être décidé par une machine de Turing déterministe en espace logarithmique par rapport à la taille de l'instance.
- Classe **NL** : un problème de décision est dans **NL** s'il peut être décidé par une machine de Turing non déterministe en espace logarithmique par rapport à la taille de l'instance.
- Classe **P** : un problème de décision est dans **P** s'il peut être décidé par une machine de Turing déterministe en un temps polynomial par rapport à la taille de l'instance. Par abus de langage, on qualifie alors le problème de polynomial.
- Classe **NP** : un problème de décision est dans **NP** s'il peut être décidé par une machine de Turing non déterministe en temps polynomial par rapport à la taille de l'instance (i.e. en utilisant une machine de Turing non déterministe à états d'arrêts oui/non).
- Classe **co-NP** : un problème de décision est dans **co-NP** s'il est formalisé par un langage dont le complément est dans **NP**.

– Classe **PSPACE** : un problème de décision est dans **PSPACE** s'il peut être décidé par une machine de Turing déterministe en espace polynomial par rapport à la taille de son instance.
– Classe **NPSPACE** : un problème de décision est dans **NPSPACE** s'il peut être décidé par une machine de Turing non déterministe en espace polynomial par rapport à la taille de son instance.
– Classe **EXPTIME** : un problème de décision est dans **EXPTIME** s'il peut être décidé par une machine de Turing déterministe en temps exponentiel par rapport à la taille de son instance.

La figure 2.1 (reprise de (Bordeaux 2003)) donne un aperçu de l'inclusion des classes de complexité « classiques », en temps et espace, avec non déterminisme et complémentation (de nombreuses autres classes existent). Pour chacune de ces classes, un ou plusieurs problèmes caractéristiques sont indiqués. **HP** désigne la hiérarchie polynomiale présentée au paragraphe 2.3. Le préfixe **N** (e.g. **NP**) indique une classe *non déterministe*, c'est-à-dire une classe de problèmes pour laquelle la vérification des solutions candidates est un problème de la classe préfixée (e.g. **P**). Le préfixe **co** représente la complémentation. Enfin, **RE** représente l'ensemble de problèmes récursivement énumérables. Certains problèmes de cette classe sont donc seulement semi-décidables.

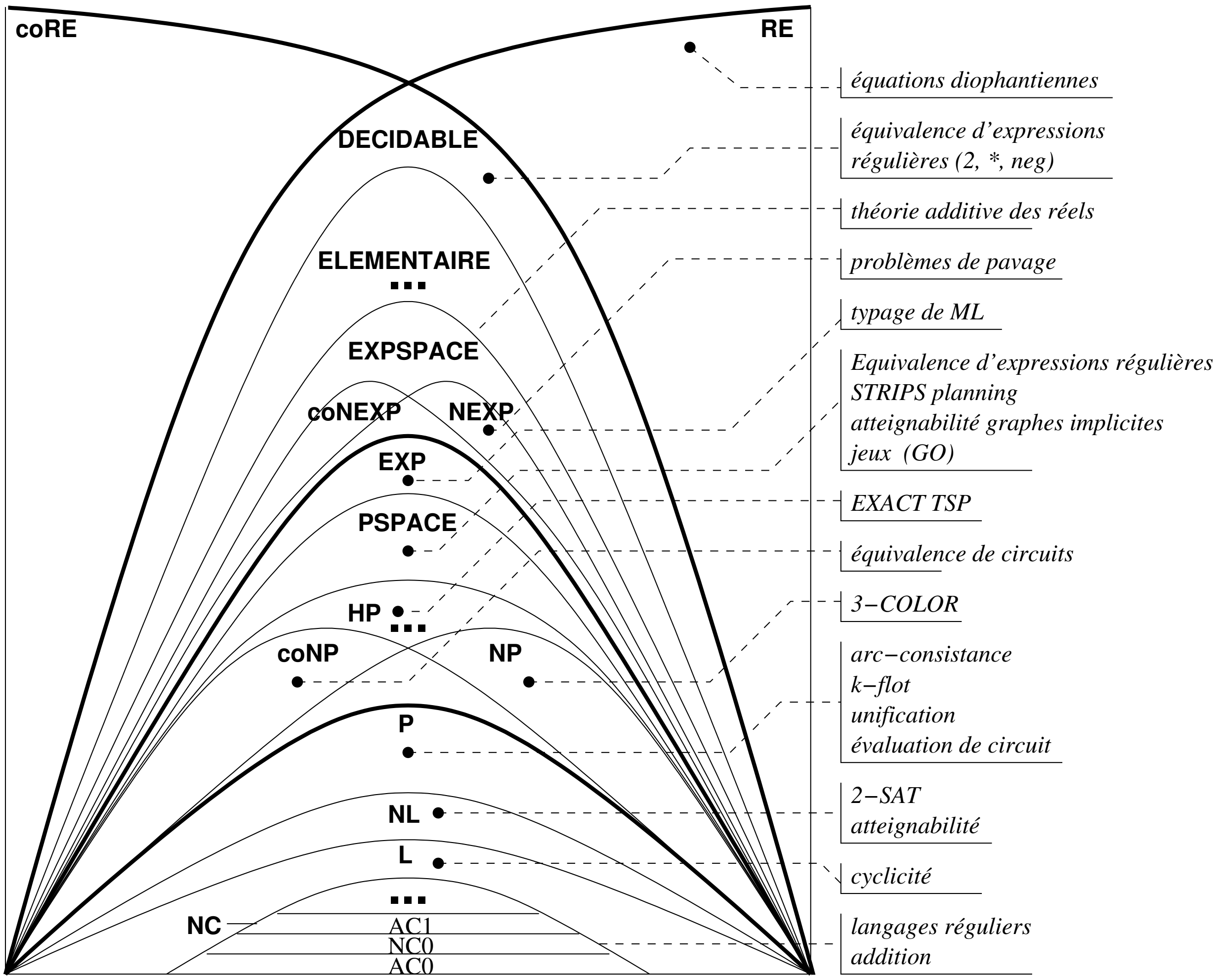


FIG. 2.1 – *Un aperçu des classes de complexité « classiques » (repris de (Bordeaux 2003)).*

Une notion fondamentale en théorie de la complexité et de la calculabilité est celle de réduction. Intuitivement, si un problème A se réduit à un problème B, alors B est au moins aussi difficile que A. Pour une classe de complexité C donnée, un problème de décision est dit C-complet lorsqu'il appartient à C et que tout problème de C se réduit à lui. Ainsi, les problèmes C-complets sont les problèmes les plus difficiles à résoudre de la classe C.

Plusieurs notions de réduction existent. Pour définir les problèmes C-complets (où C contient la classe **P**), on utilise usuellement la notion de réduction fonctionnelle polynomiale, ou réduction de Karp (Karp 1972) : on dit qu'un langage L_1 se réduit fonctionnellement polynomialement en un langage L_2 si et seulement si il existe une machine de Turing déterministe M qui pour tout mot m calcule $M(m)$ en un nombre d'étapes polynomial en $|m|$ et vérifie $\forall m.(m \in L_1$ si et seulement si $M(m) \in L_2)$.

2.2.2 Problèmes NP-complets

Les problèmes **NP**-complets sont très étudiés depuis que leur existence a été mise en évidence par Stephen Cook en 1971 (Cook 1971). Ceci est dû au fait que beaucoup de problèmes intéressants sont **NP**-complets et que l'on ne sait pas résoudre un problème **NP**-complet efficacement, c'est-à-dire en temps déterministe polynomial.

Dire qu'un problème peut être décidé à l'aide d'un algorithme non déterministe polynomial signifie qu'il est facile, pour une solution potentielle donnée, de vérifier en un temps polynomial si celle-ci répond au problème pour une instance donnée. Le non déterminisme permet de « masquer » le nombre exponentiel des solutions potentielles à tester.

Des problèmes **NP**-complets « célèbres » sont :

– problème SAT et variante 3SAT (mais 2SAT est polynomial),
– problème du voyageur de commerce,
– problème du cycle hamiltonien,
– problème de la clique maximum,
– problèmes de colorations de graphes,
– problème d'ensemble dominant dans un graphe,
– problème de couverture de sommets dans un graphe.

2.2.3 Déterministe contre non déterministe

Il est trivialement établi que $\mathbf{P} \subseteq \mathbf{NP}$ car une machine de Turing déterministe est une machine de Turing non déterministe particulière. En revanche la réciproque $\mathbf{NP} \subseteq \mathbf{P}$, que l'on résume généralement à $\mathbf{P} = \mathbf{NP}$ du fait de la trivialité de l'autre inclusion, est l'un des problèmes ouverts les plus fondamentaux et intéressants en informatique théorique. Cette question a été posée en 1970 pour la première fois et celui qui arrivera à prouver que **P** et **NP** sont différents ou égaux recevra le prix Clay (plus de 1.000.000 US$).

Le problème de fond est que les algorithmes que l'on programme sont tous déterministes. Et ceci pour la bonne et simple raison que l'on ne sait pas construire de machine non déterministe, ce qui fait que l'on ne peut que simuler un algorithme non déterministe par un algorithme déterministe. Or il est démontré qu'un algorithme déterministe qui simule un algorithme non déterministe qui fonctionne en temps polynomial, fonctionne en temps exponentiel. De ce fait, on ne peut pas résoudre en pratique des instances de grande taille, quelle que soit la puissance de la machine.

La question $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ revient à déterminer si un problème **NP**-complet peut être résolu en temps déterministe polynomial. En effet, si cela était le cas, alors on pourrait résoudre tous les problèmes de **NP** en un temps polynomial, sur une machine déterministe ; or, les problèmes **NP**-complets sont très fréquents et nul n'a jamais réussi à trouver un algorithme déterministe polynomial résolvant l'un d'entre eux. En conséquence, on conjecture que les problèmes **NP**-complets ne sont pas décidables en temps déterministe polynomial. À partir de là, plusieurs voies peuvent être explorées. En particulier :

– on peut restreindre le problème à un sous-problème (i.e. considérer un sous-ensemble du langage associé au problème) plus facile à résoudre ;
– on peut tenter de développer des algorithmes incomplets pour résoudre le problème, i.e. des algorithmes qui fournissent toujours une réponse correcte mais peuvent terminer en échec (i.e. sans fournir de réponse exploitable).

Pour le cas de **L** et **NL**, on ne sait pas non plus si $\mathbf{L} = \mathbf{NL}$ mais cette question est moins primordiale que $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ car $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP}$. Ce qui fait que les problèmes qui sont dans **L** et dans **NL** sont décidables efficacement.

Inversement on sait, selon de théorème de Savitch[4], que $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

Pour résumer on a $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} = \mathbf{NPSPACE}$, et, de plus, on sait que **NL** est strictement inclus dans **PSPACE**. Donc il existe au moins deux classes entre **NL** et **PSPACE** qui ne sont pas égales.

2.3 Hiérarchie polynomiale

2.3.1 Définitions

La hiérarchie polynomiale (cf. e.g. (Stockmeyer 1976)) est une classe de complexité structurée en sous-classes. Il est possible de définir la hiérarchie polynomiale (**HP**) de deux façons différentes : à l'aide de quantificateurs ou via un oracle. Nous présentons ici ces deux définitions.

Définition 2.2 (HP via les quantificateurs)
Soit p un polynôme. On définit $P_p(x) = \{m \mid |m| \leq p(|x|)\}$. Soit L un langage. On définit :

$$\exists^p L = \{x \mid (\exists w \in P_p(x))\, \langle x, w\rangle \in L\}\,.$$

Intuitivement, $x \in \exists^p L$ quand il existe un mot w « relativement petit » qui peut en témoigner. De la même façon, on définit

$$\forall^p L = \{x \mid (\forall w \in P_p(x))\, \langle x, w\rangle \in L\}\,.$$

Notons que les lois de dualité sont respectées : $(\exists^p L)^c = \forall^p L^c$ et $(\forall^p L)^c = \exists^p L^c$, où L^c est le complémentaire de L.

Soit $\mathcal{C}$ une classe de langages. On étend les deux définitions précédentes aux classes de langages. Ainsi,

$$\exists^{\mathbf{P}}\mathcal{C} = \{\exists^p L \mid p \text{ est un polynome et } L \in \mathcal{C}\}\,,$$
$$\forall^{\mathbf{P}}\mathcal{C} = \{\forall^p L \mid p \text{ est un polynome et } L \in \mathcal{C}\}\,.$$

Notons que les lois de dualité sont toujours respectées : $\mathbf{co}\exists^{\mathbf{P}}\mathcal{C} = \forall^{\mathbf{P}}\mathbf{co}\mathcal{C}$ *et* $\mathbf{co}\forall^{\mathbf{P}}\mathcal{C} = \exists^{\mathbf{P}}\mathbf{co}\mathcal{C}$*, où* $\mathbf{co}\mathcal{C} = \{L^c | L \in \mathcal{C}\}$*. Les classes* **NP** *et* **co-NP** *peuvent être définies par* $\mathbf{NP} = \exists^{\mathbf{P}}\mathbf{P}$*, et* $\mathbf{co\text{-}NP} = \forall^{\mathbf{P}}\mathbf{P}$.

On peut enfin donner les définitions de certaines classes de la hiérarchie polynomiale :

4. Du nom de celui qui l'a démontré en 1970 : Walter Savitch. Ce théorème stipule que $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{DSPACE}(f^2(n))$, avec

$$\mathbf{NPSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{NSPACE}(n^k) \text{ et } \mathbf{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathbf{DSPACE}(n^k).$$

En d'autres termes, si une machine de Turing non déterministe peut résoudre un problème en utilisant un espace $f(n)$, une machine de Turing ordinaire déterministe peut résoudre le même problème en le carré de l'espace.

$$\Sigma_0^{\mathbf{P}} = \Pi_0^{\mathbf{P}} = \mathbf{P}$$
$$\Sigma_{k+1}^{\mathbf{P}} = \exists^{\mathbf{P}}\Pi_k^{\mathbf{P}}$$
$$\Pi_{k+1}^{\mathbf{P}} = \forall^{\mathbf{P}}\Sigma_k^{\mathbf{P}}$$
$$\mathbf{HP} = \bigcup_{i\geq 0}\Sigma_i^{\mathbf{P}}$$

En faisant abstraction des contraintes de polynomialité et en choisissant la classe **R** des langages récursifs (i.e. décidables) pour origine, on retrouve la hiérarchie arithmétique (Kleene 1943; Mostowski 1947) permettant de classer les problèmes indécidables selon leur difficulté.

Définition 2.3 (HP via une machine de Turing avec oracle)

Soient C et C' des classes de complexité en temps et L un langage. On note C^L la classe des langages décidés par des machines de Turing à oracle L qui sont du même type que les machines utilisées dans la définition de C et sont soumis à la même contrainte sur le temps de calcul.

On note

$$C^{C'} = \bigcup_{L\in C'} C^L$$

On pose

$$\Delta_0^{\mathbf{P}} = \Sigma_0^{\mathbf{P}} = \Pi_0^{\mathbf{P}} = \mathbf{P}$$

Puis pour tout entier $i \geq 0$, on définit les classes $\Delta_i^{\mathbf{P}}$, $\Sigma_i^{\mathbf{P}}$ et $\Pi_i^{\mathbf{P}}$ de la hiérarchie polynomiale :

$$\Delta_{i+1}^{\mathbf{P}} = \mathbf{P}^{\Sigma_i^{\mathbf{P}}}$$
$$\Sigma_{i+1}^{\mathbf{P}} = \mathbf{NP}^{\Sigma_i^{\mathbf{P}}}$$
$$\Pi_{i+1}^{\mathbf{P}} = \textbf{co-NP}^{\Sigma_i^{\mathbf{P}}}$$
$$\mathbf{HP} = \bigcup_{i\geq 0}\Sigma_i^{\mathbf{P}}$$

En particulier, **NP =** $\Sigma_1^{\mathbf{P}}$ et **co-NP** $= \Pi_1^{\mathbf{P}}$. De plus, $\Delta_2\mathbf{P} = \mathbf{P}^{\mathbf{NP}}$.

2.3.2 Structure de la hiérarchie polynomiale

Les définitions précédentes impliquent les inclusions :

$$\Sigma_i^{\mathbf{P}} \subseteq \Delta_{i+1}^{\mathbf{P}} \subseteq \Sigma_{i+1}^{\mathbf{P}},$$
$$\Pi_i^{\mathbf{P}} \subseteq \Delta_{i+1}^{\mathbf{P}} \subseteq \Pi_{i+1}^{\mathbf{P}}.$$

Le fait de savoir si ces inclusions sont propres reste une question ouverte, même si la plupart des spécialistes de la question le pensent. Si, pour un entier k, $\Sigma_k^{\mathbf{P}} = \Sigma_{k+1}^{\mathbf{P}}$, alors la hiérarchie polynomiale s'effondre au niveau k : pour tout $i \geq k$, $\Sigma_i^{\mathbf{P}} = \Sigma_k^{\mathbf{P}}$. En particulier, si $\mathbf{P} = \mathbf{NP}$, alors la hiérarchie s'effondre complètement.

La figure 2.2 reprise de (Lagasquie-Schiex 1995) est une représentation graphique de la hiérarchie polynomiale. Cette représentation est fidèle à la conjecture $\mathbf{P} \neq \mathbf{NP}$.

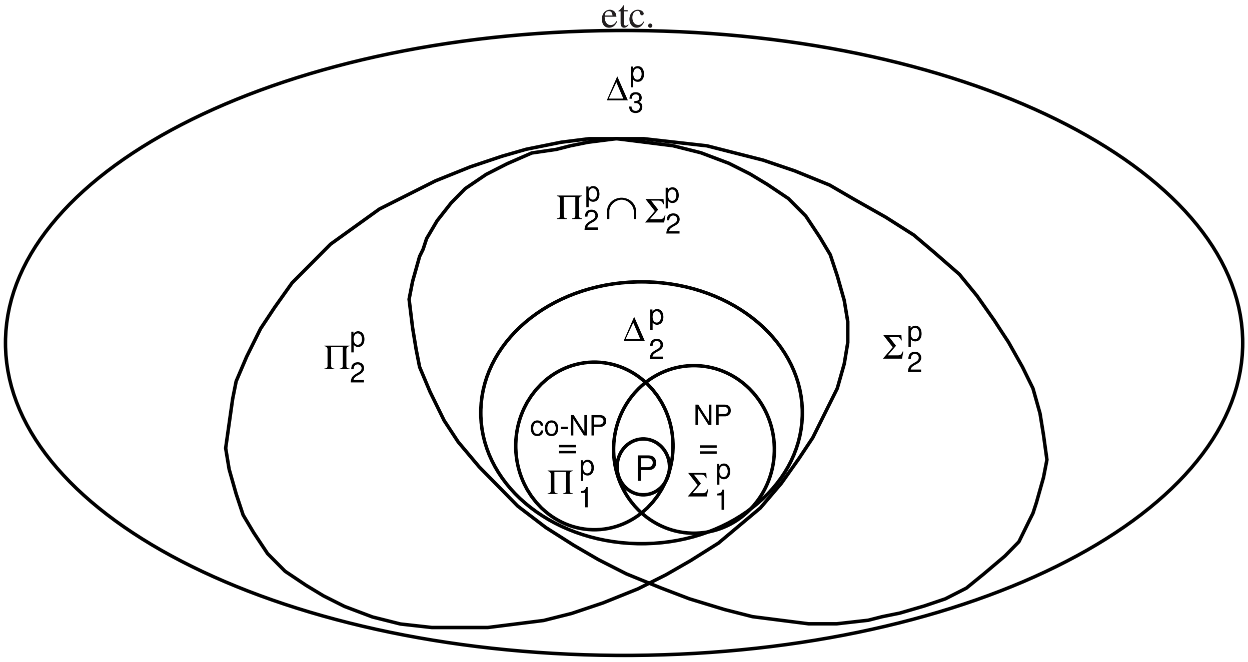


FIG. 2.2 – *Représentation d'une partie de la hiérarchie polynomiale (reprise de (Lagasquie-Schiex 1995)).*

On sait que **HP** est inclus dans **PSPACE** mais on ne sait pas à l'heure actuelle si ces deux classes sont égales.

Toutefois, si la hiérarchie polynomiale possède un problème complet, alors elle a seulement un nombre fini de niveaux différents. Puisque **PSPACE** a des problèmes complets, nous savons que si **PSPACE** $=$ **HP**, alors la hiérarchie polynomiale s'effondre.

HP et a fortiori **PSPACE** contiennent de nombreux problèmes intéressants, issus de différents domaines. Nous avons mis l'accent dans l'introduction sur quelques problèmes de décision considérés en intelligence artificielle, qui appartiennent à **PSPACE**. Voici un exemple de problème important dans la problématique de la conception de circuits.

Exemple 2.2 (Problème dans Σ_2^P)
La minimisation de circuits est un exemple de problème naturellement dans Σ_2^P : étant donnés un nombre k et un circuit A représentant une fonction booléenne f, déterminer s'il existe un circuit B avec au plus k portes qui calcule la fonction f. Soit $\mathcal{C}$ l'ensemble de tous les circuits booléens. Le langage

$$L = \{\langle A, k, B, x\rangle \in \mathcal{C} \times \mathbb{N} \times \mathcal{C} \times \{0,1\}^* \mid B \text{ a au plus } k \text{ portes, et } A(x) = B(x)\}$$

est décidable en temps polynomial. Le langage

$$CM = \left\{ \langle A, k\rangle \in \mathcal{C} \times \mathbb{N} \;\middle|\; \begin{array}{l} \text{il existe un circuit } B \text{ avec au plus } k \text{ portes} \\ \text{tel que } A \text{ et } B \text{ calculent la même fonction} \end{array} \right\}$$

appartient à Σ_2^P ($= \exists^{\mathbf{P}}\forall^{\mathbf{P}}\mathbf{P}$) car, $\langle A, k\rangle \in CM$, si et seulement si il existe un circuit B avec au plus k portes, tel que pour toute entrée x de ce circuit, $\langle A, k, B, x\rangle \in L$, et L est décidable en temps déterministe polynomial. ■

2.3.3 QBF

Le problème QBF de la validité de formules booléennes quantifiées (resp. ses restrictions $h\text{QBF}_\exists$ et $h\text{QBF}_\forall$ ($h \geq 0$)) jouent un rôle important en tant que problèmes complets canoniques pour **PSPACE** (resp. les classes $\Sigma_h^{\mathbf{P}}$ et $\Pi_h^{\mathbf{P}}$ de la hiérarchie polynomiale). En effet, on sait que QBF est **PSPACE**-complet et que pour tout $h \geq 0$, $h\text{QBF}_\exists$ est $\Sigma_h^{\mathbf{P}}$-complet alors que $h\text{QBF}_\forall$ est $\Pi_h^{\mathbf{P}}$-complet.

Comme **PSPACE** et les classes $\Sigma_h^{\mathbf{P}}$ et $\Pi_h^{\mathbf{P}}$ ($h \geq 0$) de la hiérarchie polynomiale sont fermées par réduction fonctionnelle polynomiale (dans le sens où si L_1 appartient à une de ces classes et que L_2 se réduit fonctionnellement polynomialement à L_1, alors L_2 appartient à cette même classe), on sait que tout problème L de **PSPACE** (resp. $\Sigma_h^{\mathbf{P}}$, $\Pi_h^{\mathbf{P}}$ ($h \geq 0$)) se réduit fonctionnellement polynomialement à QBF (resp. $h\text{QBF}_\exists$, $h\text{QBF}_\forall$). Pour décider L, une voie possible consiste donc à mettre en évidence une réduction fonctionnelle polynomiale M de L vers QBF (resp. $h\text{QBF}_\exists$, $h\text{QBF}_\forall$) puis d'utiliser un prouveur QBF pour déterminer si $M(m)$ est une $\mathcal{QBF}$ valide ou pas ; la réponse à cette question indique aussi si $m \in L$ ou pas.

Deuxième partie

Algorithmique et classes polynomiales pour QBF

3

Algorithmique pour QBF

Sommaire

« *Pourquoi apprendre alors que l'ignorance est instantanée ?* »
Bill Watterson[5], extrait de *Calvin et Hobbes – Enfin seuls !*.

Nous décrivons dans ce chapitre les fondements algorithmiques utilisés dans les prouveurs QBF existants : Decide (Rintanen 1999b), Evaluate (Cadoli *et al.* 1998), OpenQbf, QSolve (Feldmann *et al.* 2000), QKN (Kleine-Büning *et al.* 1995), QuBE (Giunchiglia *et al.* 2001b), Quaffle (Zhang & Malik 2002), Semprop (Letz 2001), Quantor (Biere 2004), sKizzo (Benedetti 2005). Ces prouveurs sont disponibles sur le site web dédié à QBF : qbflib[6].

3.1 Techniques générales

3.1.1 Q-résolution

La Q-résolution, décrite dans (Kleine-Büning *et al.* 1995) en 1995 par H. Kleine-Büning, M. Karpinski et A. Flögel, est une adaptation aux $\mathcal{QBF}$ du célèbre principe de résolution de (Robinson 1965).

La Q-résolution s'applique aux formules $\mathcal{CNF}$-$h\mathcal{QBF}$. Soit $\Sigma = QX_1 \dots QX_h.\phi$. La Q-résolution correspond au principe de résolution ordinaire en logique propositionnelle où :

– seules les clauses de ϕ contenant des littéraux *existentiels* complémentaires peuvent être mises en résolution ; la résolvante de deux clauses $\gamma_i = l \vee \gamma'_i$ et $\gamma_j = l^c \vee \gamma'_j$ vérifiant cette propriété est la

5. Watterson (Bill), dessinateur de bandes dessinées américain né à Washington D.C. en 1958.
6. `http://www.qbflib.org`

clause $\gamma_i' \vee \gamma_j'$ contenant tous les littéraux des clauses mises en résolution, privés de l pour γ_i et le l^c pour γ_j.

– chaque littéral universel l d'une clause α de ϕ dans laquelle aucune variable existentielle n'apparaît plus interne à la variable universelle $|l|$ associée à ce littéral dans le préfixe est supprimée de α.

Définition 3.1 (Q-résolution)

Soit $\Sigma = QX_1 \ldots QX_n.\phi$ *une* $\mathcal{CNF}$*-h*$\mathcal{QBF}$. *Une* **preuve d'une clause** γ **par Q-résolution** *à partir de* Σ *est une suite finie* $\gamma_1 \ldots \gamma_m$ *de clauses telles que :*

– $\gamma_m = \gamma$

– *pour* $i = 1$ *à* m, γ_i *est :*

  – *une clause de* ϕ*, ou*

  – *la résolvante de* γ_j *et* γ_k *avec* $j, k < i$ *à condition que les littéraux complémentaires de* γ_j *et* γ_k *utilisés pour produire* γ_i *correspondent à des variables quantifiées existentiellement, ou*

  – *la restriction de* γ_j *avec* $j < i$ *obtenue en supprimant de* γ_j *tous les littéraux associés à des variables quantifiées universellement* v *pour lesquelles il n'existe pas dans* γ_j *de littéraux associés à des variables quantifiées existentiellement qui suivent* v *dans le préfixe de* Σ.

On note $\Sigma \vdash_{\textit{Q-Res}} \gamma$ lorsque γ a une preuve par Q-résolution à partir de Σ.

On représente souvent de telles preuves par des arbres de résolution dans lesquels on fait apparaître les résolvantes et leurs clauses parentes et on effectue au niveau de chaque clause les étapes de restriction.

Notons que l'étape de résolution permet de réaliser l'élimination des quantifications existentielles (on peut vérifier que $\exists x.((\gamma_1 \vee x) \wedge (\gamma_2 \vee \neg x)) \equiv \gamma_1 \vee \gamma_2$ alors que l'étape de restriction permet de réaliser l'élimination des quantifications universelles ($\forall x.(\gamma_1 \vee x) \equiv \forall x.(\gamma_1 \vee \neg x) \equiv \gamma_1$). On observera en particulier que toute $\forall$-clause pure équivaut à Faux.

Exemple 3.1 (Q-résolution)

1. $\Sigma_1 = \forall\, y_1, y_2 \,\exists\, x_1. \begin{bmatrix} (y_1 \vee x_1) \wedge \\ (y_2 \vee \neg x_1) \end{bmatrix}.$

   $\Sigma_1 \vdash_{\textit{Q-Res}} \forall\, y_1, y_2 \,\exists\, x_1. \begin{bmatrix} [(y_1 \vee x_1) & \wedge \\ (y_2 \vee \neg x_1) & \wedge \\ (y_1 \vee y_2) & \end{bmatrix} \equiv$ Faux (cf. figure 3.1).

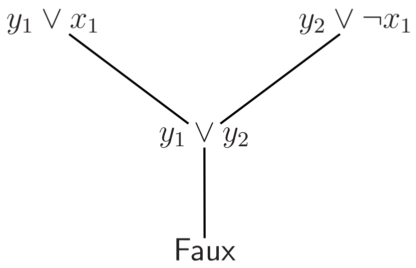


FIG. 3.1 – *Étape de Q-résolution pour* Σ_1

2. $\Sigma_2 = \forall\, y_1 \,\exists\, x_1 \,\forall y_2. \begin{bmatrix} (\neg x_1 \vee y_2) & \wedge \\ (y_1 \vee x_1) & \end{bmatrix}.$

   $\Sigma_2 \vdash_{\textit{Q-Res}} \forall\, y_1 \,\exists\, x_1 \,\forall y_2. \begin{bmatrix} (\neg x_1 \vee y_2) & \wedge \\ (y_1 \vee x_1) & \wedge \\ (\neg x_1) & \wedge \\ (y_1) & \end{bmatrix} \equiv$ Faux (cf. figure 3.2).

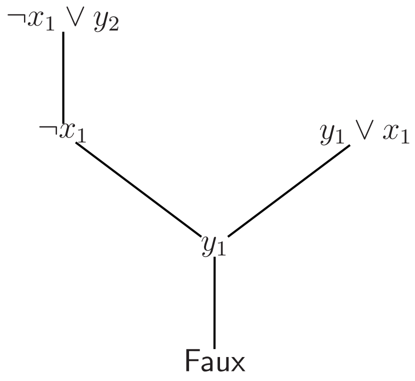


FIG. 3.2 – *Étape de Q-résolution pour* Σ_2

3. $\Sigma_3 = \forall\, y_1\, \exists\, x_1,\ x_2\, \forall y_2\, \exists x_3\, \forall y_3. \begin{bmatrix} (x_1 \vee y_2 \vee x_3) & \wedge \\ (x_2 \vee \neg x_3 \vee y_3) & \wedge \\ (y_1 \vee \neg x_1) & \wedge \\ (y_1 \vee \neg x_2) & \end{bmatrix}.$

$$\Sigma_3 \quad \vdash_{\overline{Q\text{-}Res}} \quad \forall\, y_1\, \exists\, x_1,\ x_2\, \forall y_2\, \exists x_3\, \forall y_3. \begin{bmatrix} (x_1 \vee y_2 \vee x_3) & \wedge \\ (x_2 \vee \neg x_3 \vee y_3) & \wedge \\ (y_1 \vee \neg x_1) & \wedge \\ (y_1 \vee \neg x_2) & \wedge \\ (x_1 \vee x_2 \vee y_2) & \end{bmatrix}$$

$$\vdash_{\overline{Q\text{-}Res}} \quad \forall\, y_1\, \exists\, x_1,\ x_2\, \forall y_2\, \exists x_3\, \forall y_3. \begin{bmatrix} (x_1 \vee y_2 \vee x_3) & \wedge \\ (x_2 \vee \neg x_3 \vee y_3) & \wedge \\ (y_1 \vee \neg x_1) & \wedge \\ (y_1 \vee \neg x_2) & \wedge \\ (x_1 \vee x_2 \vee y_2) & \wedge \\ (y_1 \vee x_2) & \end{bmatrix} \qquad \text{(cf. figure 3.3)}$$

$$\vdash_{\overline{Q\text{-}Res}} \quad \forall\, y_1\, \exists\, x_1,\ x_2\, \forall y_2\, \exists x_3\, \forall y_3. \begin{bmatrix} (x_1 \vee y_2 \vee x_3) & \wedge \\ (x_2 \vee \neg x_3 \vee y_3) & \wedge \\ (y_1 \vee \neg x_1) & \wedge \\ (y_1 \vee \neg x_2) & \wedge \\ (x_1 \vee x_2 \vee y_2) & \wedge \\ (y_1) & \end{bmatrix} \equiv \mathsf{Faux}$$

■

Théorème 3.1 ((Kleine-Büning *et al.* 1995))
Une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ est contradictoire si et seulement si $\Sigma \vdash_{\overline{Q\text{-}Res}} \mathsf{Faux}$.

Les trois formules Σ_1, Σ_2 et Σ_3 de l'exemple 3.1 sont donc bien contradictoires.

Puisque la Q-résolution est une extension de la résolution en logique propositionnelle, les limitations de cette dernière portent aussi sur la Q-résolution. Ainsi, il existe des formules contradictoires Σ pour lesquelles toute réfutation (i.e. toute preuve de Faux) à partir de Σ a une longueur exponentielle en la taille de l'entrée (Haken 1985; Urquhart 1987; Urquhart 1995).

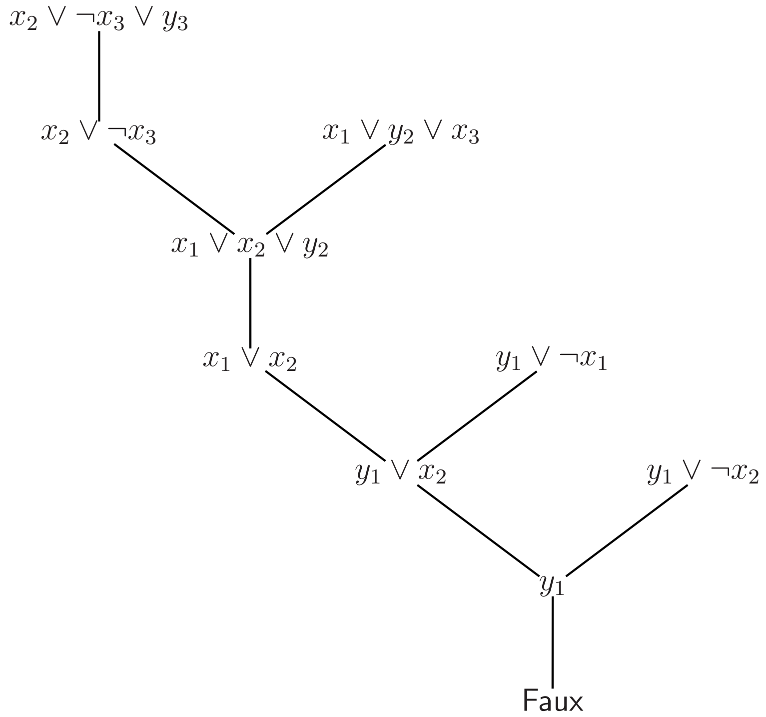


FIG. 3.3 – *Étapes de Q-résolution pour* Σ_3

3.1.2 Q-DPLL

La plupart des prouveurs QBF actuels utilisent pour procédure principale une procédure à la DPLL (Davis *et al.* 1962). Cette technique est une généralisation de la procédure de Davis-Putnam (DP ou DPLL) (Davis & Putnam 1960) pour SAT qui s'exécute en espace polynomial. Pour déterminer si une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ est valide ou pas, l'algorithme Q-DPLL consiste tout simplement à parcourir l'ensemble des interprétations partielles sur $Var(\Sigma)$ de façons systématique.

L'algorithme 3.1 décrit une procédure à la DPLL pour $\mathcal{QBF}$ telle qu'elle est présentée usuellement.

Dans cet algorithme, la $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ d'entrée est vue comme sa matrice, elle-même représentée comme un ensemble de clauses, chaque clause étant représentée pas l'ensemble de ses littéraux. Chaque variable est ainsi implicitement associée à une quantification et le test $\lambda \in \exists$ permet de décider si $|\lambda|$ est quantifiée existentiellement ou pas. Nous utiliserons également cette convention d'écriture dans la suite de ce chapitre afin d'abréger les notations.

Il convient de définir aussi les fonctions suivantes utilisées dans Q-DPLL :

– SIMPLIFIER : utilise les techniques de propagation (paragraphe 3.2.2) et de fausseté et vérité triviale (paragraphe 3.2.1) pour élaguer l'arbre de recherche. Elle prend pour paramètre une $\mathcal{CNF}$-$h\mathcal{QBF}$ et retourne une $\mathcal{CNF}$-$h\mathcal{QBF}$ équivalente simplifiée.

– CHOIX_LITTÉRAL : cette fonction utilise une heuristique de branchement pour choisir un littéral à affecter. Le choix des littéraux à affecter respecte l'ordre du préfixe, i.e. pour tout $i, j \in 1 \ldots h$, si $x \in X_i$ et $x' \in X_j$ et $i < j$, alors x est toujours choisie avant x'. Si plus aucun choix n'est possible (tous les littéraux ont été affectés), cette fonction retourne Inconnu. Dans le cas contraire, cette fonction retourne le littéral à affecter à Vrai. Certaines des heuristiques les plus utilisées sont décrites au paragraphe 3.3.

– $\vee$ (resp. $\wedge$) : cet opérateur n'est pas considéré comme l'opérateur logique classique $\vee$ (resp. $\wedge$) mais comme un opérateur logique « de programmation ». De ce fait, si le premier opérande est évalué à

Algorithme 3.1 : Q-DPLL récursif

fonction RECURSIFDPLL
Données : Une $\mathcal{CNF}$-$h\mathcal{QBF}$ $\Sigma = QX_1, \ldots, QX_h.\phi$
Résultat : Vrai si Σ est valide, Faux sinon.
début
  $\Sigma \leftarrow$ SIMPLIFIER(Σ) ;
  si $\Sigma = \{\}$ **alors**
    retourner Vrai ;
  sinon si $\{\} \in \Sigma$ **alors**
    retourner Faux ;
  $\lambda \leftarrow$ CHOIX_LITTÉRAL ;
  si $\lambda \in \exists$ **alors**
    retourner RECURSIFDPLL($\Sigma_{\lambda \leftarrow 1}$) $\vee$ RECURSIFDPLL($\Sigma_{\lambda \leftarrow 0}$) ;
  sinon retourner RECURSIFDPLL($\Sigma_{\lambda \leftarrow 1}$) $\wedge$ RECURSIFDPLL($\Sigma_{\lambda \leftarrow 0}$) ;
fin

Vrai (resp. Faux), alors le second opérande ne sera pas calculé.

La correction partielle de l'algorithme Q-DPLL repose sur les points 1. et 2. de la proposition 1.1, et sa terminaison sur le fait que la taille de Σ diminue après chaque étape de conditionnement suivie d'une simplification.

3.2 Règles de simplification et parcours de l'espace de recherche

Dans ce paragraphe, nous présentons les principales techniques de simplification et de parcours de l'espace de recherche utilisées dans les prouveurs de type Q-DPLL : fausseté triviale sur $\forall$ et $\exists$, vérité triviale, propagation de littéraux purs, propagation unitaire, inversion de quantificateurs et échantillonnage, *backtracking*, *backjumping* et apprentissage. Si la propagation unitaire et la fausseté triviale sur $\forall$ est exploitée dans tous les prouveurs de type Q-DPLL, ce n'est pas le cas pour les autres techniques présentées. Ainsi, la détection de fausseté triviale sur $\exists$ est réalisée dans les prouveurs QSat (Rintanen 2001) et ssolve (Feldmann *et al.* 2000), la vérité triviale et la propagation de littéraux purs sont exploitées dans les mêmes prouveurs ainsi que dans QuBE (Giunchiglia *et al.* 2003) et Semprop (Letz 2001). L'inversion de quantificateurs et l'échantillonnage sont des techniques utilisées dans les prouveurs QSat et ssolve. Enfin, seuls OpenQbf, QSat et ssolve intègrent un retour arrière systématique. Les autres utilisent un parcours en *backjumping* de l'espace de recherche et Semprop et une version de QuBE (Giunchiglia *et al.* 2002) implantent une technique d'apprentissage.

3.2.1 Fausseté et vérité triviale

Avant d'introduire ces notions de vérité et de fausseté triviale, il convient, à l'instar des auteurs de ces techniques (Cadoli *et al.* 2002b), de définir les trois sous-ensembles suivants d'un ensemble de clauses représentant la matrice d'une formules $\mathcal{CNF}$-$h\mathcal{QBF}$[7] :

– $H(\exists)$: ensemble des clauses dans lesquelles seuls des $\exists$-littéraux apparaissent,
– $G(\forall)$: ensemble des clauses dans lesquelles seuls des $\forall$-littéraux apparaissent,

7. Par souci de lisibilité, nous avons choisi de renommer les ensembles décrits dans (Cadoli *et al.* 2002b). Ici, $H(\Sigma)$ (resp. $G(\Pi)$, $L(\Sigma, \Pi)$) est remplacé par $H(\exists)$ (resp. $G(\forall)$, $L(\exists, \forall)$).

– $L(\exists,\forall)$: ensemble des autres clauses.

Ainsi, toute $\mathcal{CNF}$-$h\mathcal{QBF}$ peut s'écrire sous la forme

$$Q_1X_1 \dots Q_nX_n.\left[H(\exists) \wedge G(\forall) \wedge L(\exists,\forall)\right].$$

Exemple 3.2 (Ensembles $H(\exists)$, $G(\forall)$ et $L(\exists,\forall)$)
Considérons la $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ suivante :

$$\Sigma = \exists x_1, x_2\ \forall y_1, y_2, y_3\ \exists x_3, x_4. \begin{bmatrix} (y_1 \vee \neg y_2) & \wedge \\ (x_1 \vee \neg y_1 \vee x_4) & \wedge \\ (\neg x_1 \vee x_3) & \wedge \\ (x_1 \vee x_3) & \wedge \\ (x_1 \vee \neg x_3) & \wedge \\ (\neg x_1 \vee \neg x_3) & \wedge \\ (x_1 \vee y_2 \vee y_3 \vee \neg x_4) & \wedge \\ (x_2 \vee y_1 \vee y_2 \vee x_4) & \end{bmatrix}.$$

Les ensembles $H_\Sigma(\exists)$, $G_\Sigma(\forall)$ et $L_\Sigma(\exists,\forall)$ de Σ sont :

$$H_\Sigma(\exists) = \left\{ \begin{matrix} (\neg x_1 \vee x_3), \\ (x_1 \vee x_3), \\ (x_1 \vee \neg x_3), \\ (\neg x_1 \vee \neg x_3) \end{matrix} \right\},$$

$$G_\Sigma(\forall) = \left\{ (y_1 \vee \neg y_2) \right\},$$

$$L_\Sigma(\exists,\forall) = \left\{ \begin{matrix} (x_1 \vee \neg y_1 \vee x_4), \\ (x_1 \vee y_2 \vee y_3 \vee \neg x_4), \\ (x_2 \vee y_1 \vee y_2 \vee x_4) \end{matrix} \right\}.$$

■

Les auteurs réalisent ensuite les observations suivantes, exprimées sous forme de lemmes, qui permettent de simplifier des $\mathcal{QBF}$.

Lemme 3.1 (Fausseté triviale sur $\forall$ (Cadoli *et al.* 2002b))
Une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ est fausse si $G'(\forall) \neq \emptyset$, où G' est obtenu en supprimant de $G(\forall)$ les clauses tautologiques.

Exemple 3.3 (Fausseté triviale sur $\forall$)
Dans la $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ de l'exemple 3.2, nous avons

$$G_\Sigma(\forall) = \left\{ (y_1 \vee \neg y_2) \right\} \neq \emptyset.$$

Donc Σ est trivialement fausse sur $\forall$. ■

Ce lemme découle de manière évidente du fait que toute $\forall$-clause pure est contradictoire. La simple vérification requise pour le lemme 3.1 peut être accomplie en temps linéaire en la taille de la matrice. Par conséquent, les cas intéressants présentent toujours la propriété $G(\forall) = \emptyset$. Pour la suite, nous considérons donc que c'est le cas. Ainsi, la forme générique d'une $\mathcal{CNF}$-$h\mathcal{QBF}$ devient :

$$Q_1X_1 \dots Q_nX_n.[H(\exists) \wedge L(\exists,\forall)].$$

Lemme 3.2 (Fausseté triviale sur $\exists$ (Cadoli *et al.* 2002b))
Une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ est fausse si $H(\exists)$ est insatisfiable.

Exemple 3.4 (Fausseté triviale sur $\exists$)
Soit $\Sigma' = \Sigma \setminus G_\Sigma(\forall)$, avec Σ la $\mathcal{CNF}$-$h\mathcal{QBF}$ de l'exemple 3.2. Σ' est trivialement fausse sur $\exists$ car

$$H_\Sigma(\exists) = \left\{ \begin{array}{c} (\neg x_1 \vee x_3), \\ (x_1 \vee x_3), \\ (x_1 \vee \neg x_3), \\ (\neg x_1 \vee \neg x_3) \end{array} \right\}$$

n'est pas satisfiable. ■

Le lemme 3.3 présente une condition suffisante mais pas nécessaire pour la validité de Σ.

Lemme 3.3 (Vérité triviale (Cadoli *et al.* 2002b))
Une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ est vraie si $H(\exists) \wedge L'(\exists)$ est satisfiable, où $L'(\exists)$ est $L(\exists, \forall)$ auquel on supprime toutes les $\forall$-variables des clauses qui en contiennent.

Exemple 3.5 (Vérité triviale)
Considérons la $\mathcal{CNF}$-$h\mathcal{QBF}$ $\Sigma'' = \Sigma' \setminus \{(\neg x_1 \vee x_3)\}$.

$$\Sigma'' = \exists x_1, x_2\, \forall y_1, y_2, y_3\, \exists x_3, x_4 . \left[\begin{array}{cc} (x_1 \vee \neg y_1 \vee x_4) & \wedge \\ (x_1 \vee x_3) & \wedge \\ (x_1 \vee \neg x_3) & \wedge \\ (\neg x_1 \vee \neg x_3) & \wedge \\ (x_1 \vee y_2 \vee y_3 \vee \neg x_4) & \wedge \\ (x_2 \vee y_1 \vee y_2 \vee x_4) & \end{array} \right].$$

Or,

$$H_{\Sigma''}(\exists) \wedge L'_{\Sigma''}(\exists) = \left\{ \begin{array}{c} (x_1 \vee x_3), \\ (x_1 \vee \neg x_3), \\ (\neg x_1 \vee \neg x_3), \\ (x_1 \vee x_4), \\ (x_1 \vee \neg x_4), \\ (x_2 \vee x_4) \end{array} \right\}.$$

Cette formule propositionnelle est satisfiable et un de ses modèles (sur $\{x_1, x_2, x_3, x_4\}$) est $\{x_1, x_2, \neg x_3, \neg x_4\}$. ■

3.2.2 Propagation

Les techniques suivantes sont décrites dans (Cadoli *et al.* 2002b). Il s'agit d'adaptations de techniques utilisées dans les procédures de Davis et Putnam et de Davis, Logemann et Loveland (Davis & Putnam 1960; Davis *et al.* 1962). Elles sont utilisées par l'ensemble des prouveurs Q-DPLL, i.e. les prouveurs QBF s'appuyant sur la procédure DPLL.

3.2.2.1 Propagation de littéraux monotones

Lemme 3.4 ($\exists$-littéral monotone (Cadoli *et al.* 2002b))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$ de la forme $Q_1X_1 \ldots Q_nX_n.\phi$ et l un $\exists$-littéral monotone dans ϕ. Alors Σ est valide si et seulement si $\Sigma_{l \leftarrow 1}$ est valide.

En effet, Σ est valide si $\Sigma_{x\leftarrow 1}$ est valide. Si $\Sigma_{x\leftarrow 1}$ est valide, alors $\Sigma_{x\leftarrow 1} \vee \Sigma_{x\leftarrow 0}$ est valide. Si $\Sigma_{x\leftarrow 1}$ est incohérent, alors $\Sigma_{x\leftarrow 0}$ l'est aussi car $\Sigma_{x\leftarrow 1} \subseteq \Sigma_{x\leftarrow 0}$ donc $\Sigma_{x\leftarrow 1} \vee \Sigma_{x\leftarrow 0}$ est aussi incohérent.

Une vision duale du lemme 3.4 pour les littéraux universels existe également.

Lemme 3.5 ($\forall$-littéral monotone (Cadoli *et al.* 2002b))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$ de la forme $Q_1X_1 \ldots Q_nX_n.\phi$ et l un $\forall$-littéral monotone dans ϕ. Alors Σ est valide si et seulement si $\Sigma_{l\leftarrow 0}$ est valide.

En effet, Σ est valide si $\Sigma_{x\leftarrow 0}$ est valide. Si $\Sigma_{x\leftarrow 0}$ est incohérent, alors $\Sigma_{x\leftarrow 1} \wedge \Sigma_{x\leftarrow 0}$ est incohérent. Si $\Sigma_{x\leftarrow 0}$ est valide, alors $\Sigma_{x\leftarrow 1}$ l'est aussi car $\Sigma_{x\leftarrow 1} \subseteq \Sigma_{x\leftarrow 0}$ donc $\Sigma_{x\leftarrow 1} \wedge \Sigma_{x\leftarrow 0}$ est aussi valide.

Exemple 3.6 (Propagation de littéraux monotones)
Soient $X = \{x_1, x_2\}$, $Y = \{y_1, y_2, y_3\}$ et Σ la $\mathcal{CNF}$-$2\mathcal{QBF}_\forall$ suivante :

$$\forall Y\, \exists X. \begin{bmatrix} (y_1 \vee y_2 \vee \neg x_1) & \wedge \\ (y_1 \vee \neg x_2 \vee \neg x_3) & \wedge \\ (\neg x_1 \vee x_2 \vee x_3) & \wedge \\ (\neg y_2 \vee \neg x_2 \vee x_3) & \end{bmatrix}.$$

Le $\forall$-littéral y_1 est monotone dans Σ, ainsi, selon le lemme 3.5, Σ est valide si et seulement si la formule Σ' est valide, où Σ' vaut

$$\forall Y\, \exists X. \begin{bmatrix} (y_2 \vee \neg x_1) & \wedge \\ (\neg x_2 \vee \neg x_3) & \wedge \\ (\neg x_1 \vee x_2 \vee x_3) & \wedge \\ (\neg y_2 \vee \neg x_2 \vee x_3) & \end{bmatrix},$$

i.e., y_1 est affecté à Faux. À présent, le $\exists$-littéral $\neg x_1$ est monotone dans Σ'. Alors, nous définissons Σ'' comme

$$\forall Y\, \exists X. \begin{bmatrix} (\neg x_2 \vee \neg x_3) & \wedge \\ (\neg y_2 \vee \neg x_2 \vee x_3) & \end{bmatrix},$$

i.e., $\neg x_1$ est affecté à Vrai. Selon le lemme 3.4, Σ'' est valide si et seulement si Σ' l'est également. À présent, le $\forall$-littéral $\neg y_2$ est monotone dans Σ''. Nous définissons Σ''' comme

$$\forall Y\, \exists X. \begin{bmatrix} (\neg x_2 \vee \neg x_3) & \wedge \\ (\neg x_2 \vee x_3) & \end{bmatrix},$$

i.e., $\neg y_2$ est affecté à Faux. Désormais, le $\exists$-littéral $\neg x_2$ est monotone dans Σ''' et peut être affecté à vrai. La $\mathcal{QBF}$ résultante a une matrice vide. Elle est donc valide et la $\mathcal{QBF}$ d'origine Σ l'est également. ■

Les conditions décrites dans des lemmes 3.4 et 3.5 peuvent être vérifiées en temps polynomial en $|\Sigma|$ (Cadoli *et al.* 2002b).

3.2.2.2 Propagation unitaire

Lemme 3.6 (Affectation forcée d'un $\exists$-littéral (Cadoli *et al.* 2002b))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$ de la forme $Q_1X_1 \ldots Q_hX_h.\phi$ et γ une clause de ϕ telle que :

1. il existe un $\exists$-littéral $l \in X_i$ dans γ, et
2. tous les autres littéraux de γ appartiennent aux ensembles de littéraux universels $X_{i+1} \cup X_{i+3} \cup \cdots \cup X_{n-1}$.

Alors Σ est valide si et seulement si $\Sigma' = Q_1X_1 \dots Q_nX_n.\phi'$ est valide, où ϕ' est obtenue en conditionnant l par 1 dans ϕ.

Exemple 3.7 (Affectation forcée d'un $\exists$-littéral)
Soit la $\mathcal{QBF}$ Σ suivante :

$$\Sigma = \exists x_1 \, \forall y_1, y_2 \, \exists x_2. \left[\begin{array}{ll} (x_1 \lor \neg y_1) & \land \\ (\neg x_2 \lor y_1) & \land \\ (\neg x_1 \lor x_2 \lor y_2) & \end{array} \right].$$

Dans la première clause, x_1 n'apparaît qu'avec y_1 qui est plus interne que x_1 dans l'ordre des quantifications. Il est donc possible de forcer l'affectation de x_1 à Vrai dans Σ. En revanche, la seconde clause ne permet pas d'affectation forcée car, bien que x_2 soit la seule variable existentielle de la clause, l'autre variable présente est plus externe que x_2. En conséquence, nous avons :

$$\Sigma \equiv \forall y_1, y_2 \, \exists x_2. \left[\begin{array}{ll} (\neg x_2 \lor y_1) & \land \\ (x_2 \lor y_2) & \end{array} \right].$$

Là encore, aucune affectation forcée n'est possible (dans aucune des deux clauses restantes). ∎

L'affectation forcée d'un $\exists$-littéral décrite dans le lemme 3.6 peut être réalisée en temps polynomial (Cadoli *et al.* 2002b).

3.2.3 Inversion de quantificateurs et échantillonnage

Le prouveur *Decide* proposé par J. Rintanen (Rintanen 1999b; Rintanen 2001) met en œuvre deux autres méthodes d'élagage de l'espace de recherche exploré par Q-DPLL : l'inversion de quantificateurs et l'échantillonnage.

La première technique d'élagage donnée est l'inversion de quantificateurs. L'idée est la suivante : étant donnée une $\mathcal{QBF}$ de la forme $\exists X \, \forall Y.\Psi$, il est intéressant de considérer également la $\mathcal{QBF}$ $\forall Y \, \exists X.\Psi$ qui est une conséquence logique de $\exists X \, \forall Y.\Psi$. En réalité, seule la contraposée nous intéresse. En effet, si $\exists X \, \forall Y.\Psi \models \forall Y \, \exists X.\Psi$, alors $\neg(\forall Y \, \exists X.\Psi) \models \neg(\exists X \, \forall Y.\Psi)$. Ainsi, si pour certaines valeurs des variables de Y, certaines valeurs de X sont impossibles pour que $\forall Y \, \exists X.\Psi$ soit valide, ces valeurs sont aussi impossibles pour que la formule $\exists X \, \forall Y.\Psi$ soit valide. De cette manière, on évite d'explorer des valeurs erronées pour les variables de X. Notons que la formule Ψ peut elle-même contenir des quantificateurs.

Sur un plan pratique, la technique consiste à considérer des affectations de valeurs de vérité aux variables de Y (éventuellement toutes les affectations possibles) choisies aléatoirement, selon une distribution uniforme, puis de détecter les littéraux sur X impliqués par propagation unitaire. Les valeurs de vérité obtenues pour les variables de X sont celles qui doivent être choisies pour l'évaluation de $\exists X \, \forall Y.\Psi$.

La technique présentée ici peut en fait être vue comme une généralisation aux $\mathcal{QBF}$ des « failed literals » utilisés dans les prouveurs SAT SATZ (Li & Anbulagan 1997) et CSAT (Dubois *et al.* 1996).

La description de la procédure d'inversion de quantificateurs est donnée à l'algorithme 3.2. Cet algorithme est exécuté avant le premier appel à la procédure principale et n'est applicable qu'aux formules dont le groupe de variables le plus externe est quantifié existentiellement.

La fonction UNIT$(L) = \{\{l\} \mid l \in L\}$ produit un ensemble de clauses unitaires à partir d'un ensemble de littéraux. Les sous-ensembles $A_1, \dots, A_n$ cohérents des littéraux de Y correspondent aux affectations de valeurs de vérité, choisies aléatoirement, des variables de Y.

La fonction UNITPROPAGATION est la propagation unitaire de complexité linéaire adaptée aux $\mathcal{CNF}$-$h\mathcal{QBF}$ telle qu'elle est décrite au paragraphe 3.2.2.2 (Cadoli *et al.* 2002b). Cette fonction prend en entrée

Algorithme 3.2 : Inversion de quantificateurs

fonction INVERTQUANT
Données : Une $\mathcal{QBF}$ $\Sigma = \exists X\ \forall Y.\Psi$, avec Ψ une formule prénexe
dont la matrice est la formule $\mathcal{CNF}$ ϕ
$A_1, \ldots, A_n$ des sous-ensembles cohérents de L_Y
Résultat : Faux si une incohérence est détectée, Vrai sinon.
début
  pour chaque A_i **faire**
    # On conserve les littéraux propagés par propagation unitaire
    $\phi' \leftarrow$ UNITPROPAGATION($\phi \cup$UNIT(A_i)) ;
    si $\{\} \notin \phi'$ **alors**
      $\phi \leftarrow$ UNITPROPAGATION($\phi \cup (\phi' \cap$UNIT(L_X))) ;
    # On applique la technique des « failed literals »
    pour chaque $l \in L_X$ **faire**
      $\phi'' \leftarrow$ UNITPROPAGATION($\phi' \cup \{\{l^c\}\}$) ;
      si $\{\} \in \phi''$ **alors**
        $\phi \leftarrow$ UNITPROPAGATION($\phi \cup \{\{l\}\}$) ;
      si $\{\} \in \phi$ **alors**
        retourner Faux ;
  retourner Vrai ;
fin

une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ et retourne une nouvelle $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ' selon le processus donné au lemme 3.6 et réitère cela tant qu'une clause vérifiant les points 1. et 2. du lemme existe.

Exemple 3.8 (Inversion de quantificateurs)
Soit Σ la formule $\mathcal{CNF}$-3$\mathcal{QBF}$ suivante :

$$\Sigma = \exists\, a, b\, \forall\, c, d\, \exists\, e\ .\ \begin{bmatrix} (\neg c \vee e) & \wedge \\ (a \vee \neg e) & \wedge \\ (\neg a \vee d \vee e) & \wedge \\ (b \vee \neg d) & \wedge \\ (a \vee \neg b \vee c \vee d) & \end{bmatrix}.$$

Choisissons d'affecter c et d de toutes les façons possibles :

– $\neg c, \neg d$: on obtient la matrice $((a \vee \neg e) \wedge (\neg a \vee e) \wedge (a \vee \neg b))$ et aucun littéral parmi a et b n'est engendré.
– $\neg c, d$: on obtient $((a \vee \neg e) \wedge (b))$ et b est engendré par propagation unitaire ($\neg b$ est impossible).
– $c, \neg d$: alors on obtient $((e) \wedge (a \vee \neg e) \wedge (\neg a \vee e))$ et a est engendré par propagation unitaire via le littéral e ($\neg a$ est impossible).
– c, d : on obtient $((e) \wedge (a \vee \neg e) \wedge (b))$ et ainsi a et b sont engendrés ($\neg a$ et $\neg b$ sont impossibles).

Nous pouvons donc conclure que seuls les littéraux a et b conviendront et qu'il faudra choisir ces valeurs pour l'évaluation de Σ. De plus, l'espace de recherche est élagué : il devient inutile d'explorer les branches où a ou b sont négatifs dans l'arbre de recherche.
Notons que le seul fait de propager c et d suffit à fournir l'information nécessaire. ■

La seconde technique d'élagage de l'espace de recherche proposé par Rintanen est assez proche de la première. Elle est appelée échantillonnage (« *sampling* »). Au choix d'une valeur de vérité pour une variable $x \in X$ dans une formule $\exists X\ \forall Y.\Psi$, il serait intéressant de pouvoir vérifier que ce choix est possible pour toutes les valeurs des variables de Y. Or, ceci est trop coûteux en pratique. Rintanen propose donc d'essayer certaines valeurs de vérité pour Y et d'effectuer une propagation unitaire pour aider à la détection de mauvais choix. Il propose ainsi de tester différentes valeurs possibles pour un petit nombre de variables de Y choisies aléatoirement.

La description de cette fonction d'échantillonnage est donnée à l'algorithme 3.3.

Algorithme 3.3 : Échantillonnage

```
fonction SAMPLING
Données : Une QBF Σ = ∃X ∀Y.Ψ, avec Ψ une formule prénexe
          dont la matrice est la formule CNF φ
          nbs le nombre d'échantillonnages (un entier)
Résultat : Faux si une incohérence dans Σ est détectée, Vrai sinon.
début
    pour chaque l ∈ L_X tel que l n'est pas unitaire dans φ faire
        pour i ← 1 ... nbs faire
            R ← sous-ensemble aléatoire cohérent de L_Y ;
            φ' ← UNITPROPAGATION(φ∪UNIT({l} ∪ R)) ;
            si {} ∈ φ' alors
                φ ← UNITPROPAGATION(φ ∪ {{l^c}}) ;
            si {} ∈ φ alors
                retourner Faux ;
    retourner Vrai ;
fin
```

Exemple 3.9 (Échantillonnage)
Soit Σ la $\mathcal{CNF}$-$2\mathcal{QHF}$ suivante :

$$\Sigma = \exists a, b\, \forall c, d. \begin{bmatrix} (a \vee c) & \wedge \\ (\neg a \vee d) & \wedge \\ (b \vee c) & \wedge \\ (b \vee d) & \end{bmatrix}.$$

On prend au hasard $a =$ Vrai, on essaie $c =$ Vrai et $d =$ Faux. Par propagation unitaire, on obtient la clause vide. L'affectation $a =$ Vrai ne doit donc pas être explorée.
En revanche, le choix $c = d =$ Vrai ne permet aucune conclusion sur le choix de l'affectation de a. ■

3.2.4 Retour arrière

Il existe deux manières de procéder à un retour arrière : systématiquement ou « intelligemment ». La manière systématique consiste à remonter dans l'arbre de recherche sans chercher à utiliser les raisons qui ont conduit à l'affectation courante, de manière à élaguer l'espace de recherche. Cette technique est désignée dans la littérature sous l'appellation de « *backtracking* » ou plus précisément « *chronological backtracking* » (retour arrière systématique). La seconde technique de retour arrière est plus subtile et permet généralement de réduire l'espace de recherche restant à parcourir après une analyse des causes du retour arrière. Elle est désignée par « *backjumping* ».

3.2.4.1 Backtracking

Le retour arrière systématique est une méthode simple et naturelle de retour en arrière dans l'arbre de recherche. L'algorithme 3.1 montre que le retour arrière, par l'effet des opérateurs $\vee$ et $\wedge$, aura un comportement différent selon que l'on a réalisé un branchement sur un littéral existentiel ou universel.

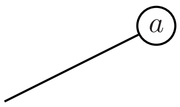
Exemple 3.10 (Backtracking)
Considérons à nouveau Σ, la formule $\mathcal{CNF}$-$h\mathcal{QBF}$ de l'exemple 1.7 suivante :

$$\Sigma = \exists\, a, b\, \forall\, c, d\, \exists\, e\, . \left[\begin{array}{ll} (b \vee c \vee \neg d \vee e) & \wedge \\ (\neg a \vee \neg b \vee \neg e) & \wedge \\ (\neg a \vee d \vee e) & \wedge \\ (\neg c \vee e) & \wedge \\ (a \vee b \vee \neg c) & \wedge \\ (b \vee \neg e) & \wedge \\ (c \vee d \vee \neg e) & \end{array}\right].$$

L'heuristique choisit de brancher sur la variable a. Si elle choisit d'affecter a à Vrai, on obtient

$$\Sigma_{\{a\}} = \exists\, b\, \forall\, c, d\, \exists\, e\, . \left[\begin{array}{ll} (b \vee c \vee \neg d \vee e) & \wedge \\ (\neg b \vee \neg e) & \wedge \\ (d \vee e) & \wedge \\ (\neg c \vee e) & \wedge \\ (b \vee \neg e) & \wedge \\ (c \vee d \vee \neg e) & \end{array}\right]$$

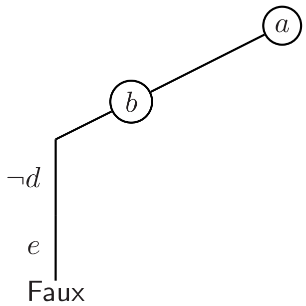
et l'arbre de recherche associé est (un arc en pointillés partant d'un nœud α désigne une affectation à Faux de la variable α et un arc plein désigne une affectation à Vrai de cette même variable) :



Ensuite, nous sommes obligés de brancher sur la variable b. Si le littéral b est choisi, alors $\neg d$ est engendré par monotonie et e par propagation unitaire. On obtient

$$\Sigma_{\{a,b,\neg d,e\}} \equiv \forall\, c\, . \left[\begin{array}{ll} \text{Faux} & \wedge \\ (c) & \end{array}\right]$$

et



qui résulte en une contradiction.

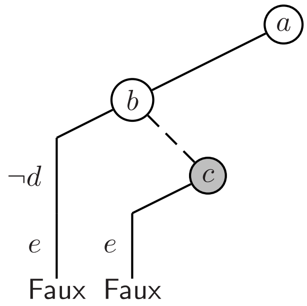
Ainsi, il est nécessaire d'effectuer un retour en arrière dans l'arbre de recherche.

Il convient donc de revenir au dernier branchement et de tester une autre valeur de vérité pour b : comme on a atteint une contradiction et que b est existentiellement quantifié, alors il faut explorer la branche $\neg b$.

Si b est affecté à Faux et que c est ensuite affecté à Vrai, alors on obtient

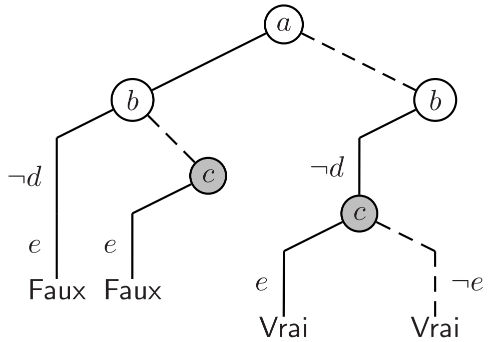
$$\Sigma_{\{a,\neg b,c\}} \equiv \forall\, d\, \exists\, e\, . \begin{bmatrix} (d \vee e) & \wedge \\ (e) & \wedge \\ (\neg e) & \end{bmatrix}$$

et e peut être propagé. On obtient une contradiction (les nœuds grisés représentent les branchements sur des variables universelles) :



Comme c est universellement quantifiée, il faut retourner en arrière jusqu'à a et explorer la branche pour laquelle a est affecté à Faux.

On obtient finalement l'arbre de recherche suivant



qui nous permet de conclure que Σ est valide. ■

3.2.4.2 Backjumping

Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$ de la forme $Q_1x_1 \ldots Q_hx_h.\phi$ et $\mu = \{l_1, \ldots, l_m\}$ un ensemble cohérent de littéraux. Tout comme dans le domaine de la satisfaction de contraintes CSP (cf. e.g. (Prosser 1993)), il est possible que seul un ensemble de littéraux v dans une affectation courante μ soit responsable de la valeur de vérité de Σ_μ (valide ou non). Ainsi, en considérant qu'il est possible de déterminer un tel sous-ensemble $v \subseteq \mu$, nous pourrions éviter d'explorer l'arbre de recherche pour la branche non encore parcourue d'un littéral l si $l \notin v$. Ce procédé est connu sous le nom de « *retour arrière intelligent* ».

Définition 3.2 (Affectation pour une $\mathcal{QBF}$)
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$ de la forme $Q_1x_1 \ldots Q_hx_h.\phi$. Un ensemble totalement ordonné et cohérent de littéraux $\mu = \{l_1, \ldots, l_m\}$ est une **affectation** *pour Σ si pour chaque littéral $l_i \in \mu$*

– *l_i est unitaire ou monotone dans $\Sigma_{\{l_1,\ldots,l_{i-1}\}}$, ou*

- l_i *apparaît dans* $\Sigma_{\{l_1,\ldots,l_{i-1}\}}$ *et pour chaque variable* x *apparaissant de manière plus interne que* $|l_i|$ *dans le préfixe de la* $\mathcal{QBF}$*,*
  - x *et* $\neg x$ *n'apparaissent pas dans* $\Sigma_{\{l_1,\ldots,l_{i-1}\}}$*, ou*
  - x *est existentielle si et seulement si* l_i *est existentiel.*

Exemple 3.11 (Affectation pour une $\mathcal{QBF}$)
Soit Σ la $\mathcal{CNF}$-4$\mathcal{QBF}$ définie comme suit :

$$\Sigma = \forall\, y_1\; \exists\, x_1\; \forall\, y_2\; \exists\, x_2, x_3\; . \begin{bmatrix} (y_1 \vee y_2 \vee x_2) & \wedge \\ (y_1 \vee \neg y_2 \vee x_2 \vee \neg x_3) & \wedge \\ (y_1 \vee \neg x_2 \vee x_3) & \wedge \\ (\neg y_1 \vee x_1 \vee x_3) & \wedge \\ (\neg y_1 \vee y_2 \vee x_2) & \wedge \\ (\neg y_1 \vee y_2 \vee \neg x_2) & \wedge \\ (\neg y_1 \vee \neg x_1 \vee \neg y_2 \vee \neg x_3) & \end{bmatrix}.$$

$\{\neg y_1, \neg y_2, x_2, x_3\}$ est une affectation pour Σ. ■

Selon la convention prise au paragraphe précédent, dans la figure 3.4, un arc en pointillés partant d'un nœud α désigne une affectation à Faux de la variable α et un arc plein désigne une affectation à Vrai de cette même variable. De plus, les nœuds grisés représentent les branchements sur des variables universelles. Dans cette figure, nous représentons un arbre de recherche que nous pouvons obtenir en appliquant sur la formule Σ de l'exemple 3.11 l'algorithme de retour arrière systématique évoqué au paragraphe précédent.

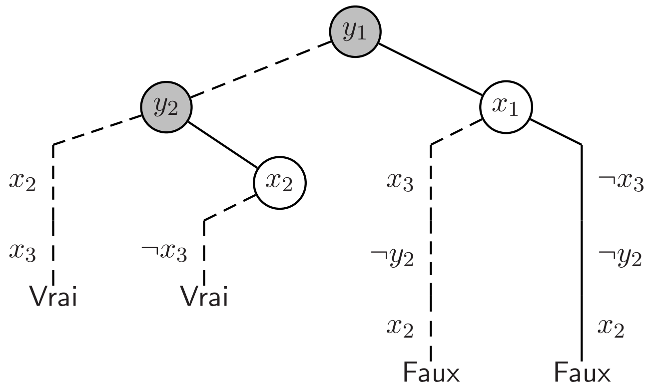


FIG. 3.4 – *Arbre de recherche pour une procédure Q-DPLL munie d'une retour arrière systématique sur* Σ *(exemple 3.11).*

Pour toute $\mathcal{QBF}$ fermée Σ, on appelle *valeur de vérité* de Σ sa validité ou son incohérence.

Définition 3.3 (Raison pour une valeur de vérité)
Soient Σ *une* $\mathcal{CNF}$-h$\mathcal{QBF}$ *de la forme* $Q_1x_1 \ldots Q_nx_n.\phi$ *et* $\mu = \{l_1, \ldots, l_m\}$ *une affectation pour* Σ*. Un ensemble de littéraux* v *est une* **raison** *pour la valeur de vérité de* Σ_μ *si*

- $v \subseteq Lit(\mu)$*, et*
- *pour chaque affectation* μ' *pour* Σ *telle que*
  - $v \subseteq Lit(\mu')$*,*
  - $Var(\mu') = Var(\mu)$*,*

  $\Sigma_{\mu'}$ *est valide si et seulement si* Σ_μ *est valide.*

Exemple 3.12 (Raison pour une valeur de vérité)
Si l'on considère à nouveau la formule Σ de l'exemple 3.11,

– si μ vaut $\{x_1, x_2\}$, alors $\{x_1\}$ est une raison pour la valeur de vérité de Σ_μ : pour toute affectation $\mu' \in \{\{x_1, \neg x_2\}, \{x_1, x_2\}\}$, $\Sigma_{\mu'}$ est incohérente, et
– si μ vaut $\{\neg x_1, \neg x_3\}$, alors $\{\neg x_1\}$ est une raison pour la valeur de vérité de Σ_μ : pour toute affectation
$\mu' \in \{\{\neg x_1, \neg x_3\}, \{\neg x_1, x_3\}\}$, $\Sigma_{\mu'}$ est valide.

■

Proposition 3.1 ((Giunchiglia *et al.* 2003))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$ de la forme $Q_1x_1 \dots Q_nx_n.\phi$ et $\mu = \{l_1, \dots, l_m\}$ une affectation pour Σ. Soit μ' un sous-ensemble de μ.

– μ' est une raison pour l'incohérence de Σ_μ si et seulement si la $\mathcal{QBF}$

$$\exists|l_1| \dots \exists|l_m|Q_1x_1 \dots Q_nx_n.\phi_{\mu'}$$

est incohérente,
– μ' est une raison pour la validité de Σ_μ si et seulement si la $\mathcal{QBF}$

$$\forall|l_1| \dots \forall|l_m|Q_1x_1 \dots Q_nx_n.\phi_{\mu'}$$

est valide.

Le *backjumping* peut s'utiliser de deux manières différentes : « dirigé par un conflit » (*conflict-directed backjumping* ou CBJ) ou « dirigé par une solution » (*solution-directed backjumping* ou SBJ).

Penchons-nous tout d'abord sur le CBJ .

Définition 3.4 (Littéral non pertinent)
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$, μ une affectation vérifiant une propriété P sur Σ (i.e. Σ_μ vérifie P), et $\lambda \in \mu$ un littéral. Si $\mu' = \mu \setminus \lambda$ et $\Sigma_{\mu'}$ vérifie P, alors λ est un **littéral non pertinent** *pour la propriété P.*

Dans la suite, FALSE-irrelevant (resp. TRUE-irrelevant) désigne un littéral non pertinent pour l'incohérence (resp. la validité) d'une $\mathcal{QBF}$ Σ, c'est-à-dire lorsque Σ_μ contient une clause contradictoire (resp. lorsque la matrice de Σ_μ est vide).

Le théorème suivant nous permet de construire une raison pour la valeur de vérité de Σ_μ quand la matrice de Σ_μ contient une clause contradictoire.

Théorème 3.2 ((Giunchiglia *et al.* 2003))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$, μ une affectation pour Σ telle que Σ_μ contient une clause contradictoire. Soit μ' l'affectation μ privée de ses littéraux FALSE*-irrelevant. Alors μ' est une raison pour l'incohérence de Σ_μ.*

L'étape suivante consiste à montrer comment il est possible de calculer les raisons de l'incohérence de Σ_μ pendant le *backtracking*. Notons $Val(\Sigma_\mu)$ (resp. $InCoh(\Sigma_\mu)$) la validité de Σ_μ (resp. l'incohérence de Σ_μ).

Théorème 3.3 ((Giunchiglia *et al.* 2003))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$, λ un littéral, $\{\mu, \lambda\}$ une affectation pour Σ et v une raison pour $InCoh(\Sigma_{\{\mu,\lambda\}})$.

1. *si $\lambda \notin v$, alors v est une raison pour $InCoh(\Sigma_\mu)$,*

2. *si $\lambda \in v$ et λ est universel, alors $v \setminus \{\lambda\}$ est une raison pour $InCoh(\Sigma_\mu)$,*
3. *si $\lambda \in v$ et λ est existentiel et monotone dans Σ_μ, alors $v \setminus \{\lambda\}$ est une raison pour $InCoh(\Sigma_\mu)$,*
4. *si $\lambda \in v$, $\neg\lambda \in v'$, v' une raison pour $InCoh(\Sigma_{\{\mu,\neg\lambda\}})$ et λ est existentiel, alors $(v \cup v') \setminus \{\lambda, \neg\lambda\}$ est une raison pour $InCoh(\Sigma_\mu)$.*

Exemple 3.13 (CBJ)
Considérons à nouveau la formule Σ de l'exemple 3.11.

1. Étant donné que $\{y_1, \neg y_2, x_2\}$ et $\{y_1, \neg y_2, \neg x_2\}$ (cf. exemple 3.15) sont des raisons pour $InCoh(\Sigma_{\{y_1,\neg x_1,x_3,\neg y_2,x_2\}})$ et $InCoh(\Sigma_{\{y_1,\neg x_1,x_3,\neg y_2,\neg x_2\}})$ respectivement, le quatrième point du théorème 3.3 nous permet de conclure que $\{y_1, \neg y_2\}$ est une raison pour $InCoh(\Sigma_\mu)$ quand μ vaut $\{y_1, \neg x_1, x_3, \neg y_2\}$.
2. Le second point de ce théorème nous permet ensuite de conclure que $\{y_1\}$ est une raison pour $InCoh(\Sigma_\mu)$ quand μ vaut $\{y_1, \neg x_1, x_3\}$.
3. Nous pouvons alors conclure du premier point du théorème que $\{y_1\}$ est une raison pour $InCoh(\Sigma_\mu)$ quand μ vaut $\{y_1, \neg x_1\}$ et de la même manière quand μ vaut $\{y_1\}$.
4. Finalement, le second point du théorème nous permet de conclure que l'ensemble vide $\{\}$ est une raison pour $InCoh(\Sigma_\mu)$ quand μ est vide.

Nous pouvons déduire du point 3. de l'exemple ci-dessus qu'il est inutile de vérifier si $\{y_1, x_1\}$ satisfait Σ. En effet, $\{y_1, x_1\}$ concorde avec $\{y_1, \neg x_1\}$ par $\{y_1\}$, et nous pouvons donc déduire que $\Sigma_{\{y_1,x_1\}}$ est incohérent. À partir de là, une procédure de retour arrière intelligent nous évite d'explorer la branche dans laquelle x_1 est affectée à Vrai comme présenté à la figure 3.4. ■

Intéressons-nous maintenant au SBJ .

Le théorème suivant nous permet de construire une raison pour la valeur de vérité de Σ_μ quand la matrice de Σ_μ est vide.

Théorème 3.4 ((Giunchiglia *et al.* 2003))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$, μ une affectation pour Σ telle que la matrice de Σ_μ est vide. Soit $\mu' = \mu$ privé de ses littéraux TRUE*-irrelevant. Alors μ' est une raison pour la validité de Σ_μ.*

Théorème 3.5 ((Giunchiglia *et al.* 2003))
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$, λ un littéral, $\{\mu, \lambda\}$ une affectation pour Σ et v une raison pour $Val(\Sigma_{\{\mu,\lambda\}})$.

1. *Si $\lambda \notin v$, alors v est une raison pour $Val(\Sigma_\mu)$.*
2. *Si $\lambda \in v$ et λ est existentiel, alors $v \setminus \{\lambda\}$ est une raison pour $Val(\Sigma_\mu)$.*
3. *Si $\lambda \in v$ et λ est universel et monotone dans Σ_μ, alors $v \setminus \{\lambda\}$ est une raison pour $Val(\Sigma_\mu)$.*
4. *Si $\lambda \in v$, $\neg\lambda \in v'$, v' est une raison pour $Val(\Sigma_{\{\mu,\neg\lambda\}})$ et λ est universel, alors $(v \cup v') \setminus \{\lambda, \neg\lambda\}$ est une raison pour $Val(\Sigma_\mu)$.*

Exemple 3.14 (SBJ)
Reprenons en compte la $\mathcal{QBF}$ Σ de l'exemple 3.11

- le second point du théorème 3.4 nous permet de conclure que $\{\neg y_1, x_2\}$ est une raison pour $Val(\Sigma_\mu)$ quand μ vaut $\{\neg y_1, \neg y_2, x_2\}$,
- nous pouvons ensuite conclure du second point de ce théorème que $\{\neg y_1\}$ est une raison pour $Val(\Sigma_\mu)$ quand μ vaut $\{\neg y_1, \neg y_2\}$,

– enfin, le premier point du théorème nous permet de conclure que $\{\neg y_1\}$ est une raison pour $Val(\Sigma_\mu)$ quand μ vaut $\{\neg y_1\}$.

Nous pouvons déduire du point (2) qu'il est inutile de vérifier si $\{y_1, x_1\}$ ne satisfait pas Σ. En effet, $\{\neg y_1, \neg y_2\}$ concorde avec $\{\neg y_1, y_2\}$ par $\{\neg y_1\}$, et nous pouvons donc déduire que $\Sigma_{\{\neg y_1, y_2\}}$ est valide. À partir de là, une procédure de retour arrière intelligent nous évite d'explorer la branche dans laquelle y_2 est affectée à Vrai représentée à la figure 3.4. ■

Nous décrivons dans l'algorithme 3.5 une procédure de retour arrière intelligent analogue à celle présentée dans (Giunchiglia *et al.* 2003). Cette procédure est utilisée dans la version itérative d'un algorithme de type DPLL pour QBF présenté également (l'algorithme 3.4). Dans ces algorithmes, λ désigne une littéral, τ et ω des raisons, $\eta \in \{$L-SPLIT, R-SPLIT, PROP, Inconnu$\}$ un mode de branchement, avec L-SPLIT (resp. R-SPLIT, resp. PROP) le branchement correspondant à l'exploration de la branche de gauche (resp. de la branche de droite, resp. par une technique de propagation décrite au paragraphe 3.2.2) de l'espace de recherche et π une pile de triplets $\langle \lambda, \eta, \tau \rangle$.

Algorithme 3.4 : Q-DPLL itératif

```
fonction ITERATIFDPLL
Données : Une CNF-hQBF Σ
Résultat : Vrai si Σ est valide, Faux sinon.
début
    π ← ∅ ;
    répéter
        Σ ← SIMPLIFIER(Σ) ;
        si Σ = {} alors
            ρ ← Vrai ;
            ⟨λ, η, τ⟩ ← BACKJUMPING(ρ, π) ;
        sinon si {} ∈ Σ alors
            ρ ← Faux ;
            ⟨λ, η, τ⟩ ← BACKJUMPING(ρ, π) ;
        sinon
            ρ ← Inconnu ;
            τ ← ∅ ;
            η ← L-SPLIT ;
            λ ← CHOIX_LITTÉRAL ;
        si λ ≠ Inconnu alors
            EMPILER(⟨λ, η, τ⟩, π) ;
            Σ ← Σ_{λ←1} ;
    jusqu'à λ = Inconnu;
    retourner ρ ;
fin
```

La fonction EMPILER (resp. DÉPILER) est la fonction classique d'empilement (resp. de dépilement). Elle prend pour paramètre un triplet $\langle \lambda, \eta, \tau \rangle$ à empiler et une pile (resp. elle prend pour paramètre une pile) de triplets $\langle \lambda, \eta, \tau \rangle$ et ne retourne rien (resp. retourne un triplet $\langle \lambda, \eta, \tau \rangle$ dépilé).

La fonction CHOIX_LITTÉRAL est décrite au paragraphe 3.1.2.

Algorithme 3.5 : Backjumping

fonction *Backjump*
Données : Une variable booléenne ρ
une pile π de triplets $\langle \lambda, \eta, \tau \rangle$, avec $\lambda \in Lit(\Sigma) \cup \{\mathsf{Inconnu}\}$,
$\eta \in \{$L-SPLIT, R-SPLIT, PROP, Inconnu$\}$ et τ un ensemble de littéraux
Résultat : un triplet $\langle \lambda, \eta, \tau \rangle$
début
  $\omega \leftarrow$ INITWR(ρ) ;
  tant que $\langle$*π n'est pas vide*$\rangle$ **faire**
    $\langle \lambda, \eta, \tau \rangle \leftarrow$ DEPILER(π) ;
    si $\lambda \in \omega$ **alors**
      si (ρ = Faux **et** $\lambda \in \exists$) **ou** (ρ = Vrai **et** $\lambda \in \forall$) **alors**
        si η = PROP **or** η = R-SPLIT **alors**
          $\omega \leftarrow (\omega \cup \tau) \setminus \{\lambda, \neg\lambda\}$;
        si η = L-SPLIT **alors**
          $\eta \leftarrow$ R-SPLIT ;
          $\tau \leftarrow \omega$;
          retourner $\langle \lambda^c, \eta, \tau \rangle$;
      sinon
        $\omega \leftarrow \omega \setminus \{\lambda\}$;
  retourner $\langle$Inconnu, Inconnu, $\emptyset\rangle$;
fin

La fonction INITWR initialise une raison ω pour une valeur de vérité ρ de Σ_μ, avec μ une affectation pour Σ. Elle prend pour paramètre cette valeur de vérité ρ et retourne un ensemble de variables ω tel que :

- si ρ vaut Faux, Σ_μ contient alors une clause contradictoire. Si C est une clause de Σ telle que, pour chaque littéral $l \in C, l^c \in \mu$ ou $|l|$ est quantifiée universellement. Alors INITWR retourne l'ensemble des littéraux l existentiellement quantifiés dans μ tels que $l^c \in C$ (cf. théorème 3.2) ;
- si ρ vaut Vrai, alors la matrice de Σ_μ est vide. INITWR retourne l'ensemble des littéraux universellement quantifiés dans l'ensemble obtenu à partir de μ en éliminant récursivement les littéraux universellement quantifiés l tels que, pour chaque clause $C \in \Sigma$, si $l \in C$, alors il existe un autre littéral l' dans μ avec $l' \in C$ (cf. théorème 3.4, notion de littéral nécessaire dans la thèse de T. Castell (Castell 1997)).

Exemple 3.15 (INITWR)
Considérons à nouveau la formule Σ de l'exemple 3.11 :

- si $\mu = \{y_1, \neg x_1, x_3, \neg y_2, x_2\}$, alors INITWR(Faux) retourne $\{x_2\}$ puisque $(\neg y_1 \vee y_2 \vee \neg x_2)$ est falsifiée ;
- si $\mu = \{\neg y_1, \neg y_2, x_2, x_3\}$, alors INITWR(Vrai) retourne $\{\neg y_1\}$ car $\neg y_2$ satisfait la clause $(y_1 \vee \neg y_2 \vee x_2 \vee \neg x_3)$ qui est aussi satisfaite par x_2.

■

La fonction SIMPLIFIER est une variante de celle définie au paragraphe 3.1.2. On suppose que, pour chacun des littéraux l affectés, elle empile un triplet $\langle l, \text{PROP}, \omega\rangle$, avec ω la raison dont la valeur est fixée par la fonction INITWR.

3.2.5 Apprentissage

Nous présentons maintenant la technique d'apprentissage introduite dans (Giunchiglia *et al.* 2002).

L'idée principale est la suivante : les raisons calculées pendant la recherche sont stockées pour éviter de découvrir les mêmes raisons dans différentes branches. Pour SAT, il n'existe qu'un seul type de raison appelées « nogoods ». Pour QBF, la situation diffère légèrement puisqu'il existe deux types de raisons : les « nogoods », qui peuvent être ajoutés à la base de clauses, et les « goods » qui sont des conjonctions de littéraux.

Giunchiglia *et al.* commencent par définir la notion d'*affectation fermée pour un préfixe*.

Définition 3.5 (Affectation fermée pour un préfixe)
Soient une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ et $\mu = \{l_1, \dots, l_m\}$ une affectation pour Σ. μ est **fermée** *pour le préfixe de Σ si pour chaque littéral $l \in \mu$, pour toute variable v précédant $|l|$ dans le préfixe de Σ, on a soit $v \in \mu$, soit $\neg v \in \mu$.*

Nous pouvons en déduire une caractérisation de l'apprentissage des *nogoods*.

Proposition 3.2 (Nogoods (Giunchiglia *et al.* 2002))
Soient une $\mathcal{CNF}$-$h\mathcal{QBF}$ $\Sigma = Q_1X_1 \dots Q_hX_h.\phi$ et $\mu = \{l_1, \dots, l_m\}$ une affectation fermée pour le préfixe de Σ. Si v est une raison pour $InCoh(\Sigma_\mu)$, alors Σ est équivalent à

$$Q_1X_1 \dots Q_hX_h.(\phi \wedge (\bigvee_{\lambda \in v} \neg\lambda)).$$

En effet, si v est une raison pour $InCoh(\Sigma_\mu)$, alors

$$\Sigma \wedge (\bigwedge_{\lambda \in v} \lambda) \vdash_{\overline{Q\text{-}Res}} \mathsf{Faux} \Leftrightarrow \Sigma \vdash_{\overline{Q\text{-}Res}} \bigvee_{\lambda \in v} \neg\lambda.$$

.

Exemple 3.16 (Nogoods)
Soit Σ la $\mathcal{CNF}$-$h\mathcal{QBF}$ définie comme suit :

$$\Sigma = \exists\, x_1 \,\forall\, y \,\exists\, x_2, x_3 \,.\, \begin{bmatrix} (y \vee x_2) & \wedge \\ (\neg y \vee \neg x_2 \vee x_3) & \wedge \\ (\neg x_1 \vee x_2) & \wedge \\ (\neg y \vee \neg x_3) & \wedge \\ (y \vee \neg x_3) & \end{bmatrix}.$$

Σ est valide et

- les ensembles $\{x_1, y, x_2\}$ et $\{x_1, x_2, y\}$ sont des affectations fermées pour le préfixe de Σ,
- $\Sigma_{\{x_1,y,x_2\}} \equiv \Sigma_{\{x_1,x_2,y\}} \equiv \exists x_3.[(x_3) \wedge (\neg x_3)]$ n'est pas valide, et $\{y, x_2\}$ est une raison pour $InCoh(\Sigma_{\{x_1,y,x_2\}})$ et $InCoh(\Sigma_{\{x_1,x_2,y\}})$,
- on peut ajouter $(\neg y \vee \neg x_2)$ à la matrice de Σ et obtenir de ce fait une formule équivalente à Σ. Notons que l'on obtient cette clause par Q-résolution entre $(\neg y \vee \neg x_2 \vee x_3)$ et $(\neg y \vee \neg x_3)$.

Finalement, tout comme pour l'exemple 3.13, une procédure d'apprentissage nous évite d'explorer certaines branches de l'arbre de recherche. De plus, cet élagage devient automatique à l'aide des clauses apprises. ■

De manière symétrique, Giunchiglia *et al.* (Giunchiglia *et al.* 2002) proposent une caractérisation de l'apprentissage des *goods*.

Proposition 3.3 (Goods (Giunchiglia *et al.* 2002))
Soient une $\mathcal{CNF}$-$h\mathcal{QBF}$ $\Sigma = Q_1X_1 \ldots Q_hX_h.\phi$ et $\mu = \{l_1, \ldots, l_m\}$ une affectation fermée pour le préfixe de Σ. Si v est une raison pour $Val(\Sigma_\mu)$, alors Σ est équivalent à

$$Q_1X_1 \ldots Q_hX_h.(\neg(\bigvee_{\lambda \in v} \neg\lambda) \vee \phi).$$

Exemple 3.17 (Goods)
Soit Σ la $\mathcal{CNF}$-3$\mathcal{QBF}$ définie comme suit :

$$\Sigma = \exists\, x_1 \,\forall\, y \,\exists\, x_2, x_3 \,.\, \begin{bmatrix} (\neg x_1 \vee \neg y \vee x_2) & \wedge \\ (x_1 \vee \neg y \vee \neg x_3) & \wedge \\ (\neg y \vee \neg x_2) & \wedge \\ (y \vee x_2 \vee \neg x_3) & \wedge \\ (x_2 \vee x_3) & \end{bmatrix}.$$

On vérifie que :

- l'affectation $\{\neg x_1, \neg y, x_2\}$ est fermée pour le préfixe de Σ ;
- $\Sigma_{\{\neg x_1,\neg y,x_2\}} \equiv \mathsf{Vrai}$ est valide, et $\{\neg y, x_2\}$ est une raison pour $Val(\Sigma_{\{\neg x_1,\neg y,x_2\}})$;
- la clause $(y \vee \neg x_2)$ est un *good*, et sa négation peut être mise en disjonction avec la matrice de Σ.

Finalement, tout comme pour l'exemple 3.14, une procédure d'apprentissage nous évite d'explorer certaines branches de l'arbre de recherche. De plus, cet élagage devient automatique à l'aide des termes appris. ■

3.3 Heuristiques de branchement

Étant donné un ensemble X de variables de même nature (i.e. quantifiées universellement ou alors existentiellement) dans une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ, les heuristiques de branchement pour Q-DPLL ont pour but de mettre en évidence un littéral l tel que $|l| \in X$. Intuitivement, plus l'heuristique est efficace et plus les arbres de recherche développés par Q-DPLL pour $\Sigma_{l\leftarrow 1}$ et $\Sigma_{l\leftarrow 0}$ sont petits (voire non développé pour celui correspondant à $\Sigma_{l\leftarrow 0}$ dans le cas où l'arbre de recherche pour $\Sigma_{l\leftarrow 1}$ suffit à se prononcer sur la validité de Σ).

Différentes heuristiques de branchement pour Q-DPLL ont été proposées dans la « littérature ». Ces dernières peuvent être très simples ou au contraire relativement évoluées. Parmi les plus simples, citons :

– ordre fixe : les littéraux sont toujours choisis selon le même ordre, par exemple, l'ordre d'apparition dans le préfixe, du plus externe vers le plus interne.

– aléatoire : cette heuristique procède à un choix aléatoire des littéraux.

Cependant, les choix de littéraux peuvent se révéler plus fins. Les heuristiques les plus efficaces actuellement sont celles qui différencient deux cas : celui où le littéral à sélectionner l est quantifié existentiellement ($l \in \exists$) et celui où il est quantifié universellement ($l \in \forall$).

Les heuristiques les plus courantes sont :

– Böhm-Speckenmeyer : il s'agit de l'heuristique proposée par M. Böhm et E. Speckenmeyer en 1996 (Böhm & Speckenmeyer 1996) adaptée de la logique propositionnelle pour les $\mathcal{CNF}$-$h\mathcal{QBF}$. Pour tout littéral l tel que $|l|$ est quantifiée existentiellement, soit $n_i(l)$ (resp. $p_i(l)$) le nombre d'occurrences de l (resp. l^c) dans les clauses de taille i, et

$$h_i(l) = max(n_i(l), p_i(l)) + 2 \times min(n_i(l), p_i(l)).$$

Cette heuristique choisit un littéral l maximisant le vecteur $(h_0(l), \ldots, h_k(l))$ pour l'ordre lexicographique.

– Jeroslow-Wang : cette heuristique est une adaptation de celle proposée par R. J. Jeroslow et J. Wang pour SAT (Jeroslow & Wang 1990). Pour tout littéral l tel que $|l|$ est quantifiée existentiellement, soit

$$jw(l) = \sum_{i=0}^{\infty} N_i(l) 2^{-i}$$

avec $N_i(l) = n_i(l) + p_i(l)$ le nombre de clauses de taille i qui contiennent $|l|$. L'heuristique choisit un l maximisant $jw(l)$.

– Si $|l|$ est quantifiée universellement, on calcule $h_i(l)$ (resp. $jw(l)$) comme $h_i(l^c)$ (resp. $jw(l^c)$).

Parmi les heuristiques les plus courantes, nous pouvons également citer celles proposées par Cadoli *et al.* dans (Cadoli *et al.* 2002b) :

– BinaryFirst : il s'agit d'une simplification de l'heuristique de Böhm-Speckenmeyer, réduisant son attention uniquement sur les clauses binaires, et ignorant le nombre d'occurrences du littéral complémentaire. Cette heuristique procède comme suit :

1. construire l'ensemble de littéraux L qui apparaissent le plus souvent dans les clauses binaires de la matrice de la $\mathcal{CNF}$-$h\mathcal{QBF}$ considérée ;
2. raffiner L en :
   (a) sélectionnant les littéraux qui apparaissent le plus fréquemment dans l'ensemble de la matrice ;
   (b) sélectionnant les littéraux dont les variables apparaissent le plus fréquemment dans la matrice.

– BinaryFirst2 : cette heuristique, contrairement à la précédente, n'ignore pas la distinction entre les variables existentielles et universelles. En particulier, si le littéral à sélectionner est universel, L est raffiné en sélectionnant les littéraux qui apparaissent le plus souvent dans les clauses ne comprenant aucune autre variable universellement quantifiée. De cette façon, au moins une clause résultante est « purement existentielle », et le problème résultant est intuitivement plus simple à résoudre (spécialement dans le cas des $\mathcal{CNF}$-2$\mathcal{QBF}$).
– NonHornFirst : cette heuristique pour SAT fondée sur des idées décrites par Crawford et Auton dans (Crawford & Auton 1993) s'efforce de maximiser le nombre de clauses de Horn/reverse-Horn dans la matrice après simplification (par propagation unitaire).

3.4 Expérimentations

Depuis 2003, une évaluation comparative des prouveurs QBF est organisée chaque année. Lors de ces évaluations comparatives ouvertes à tous, il est possible de soumettre des prouveurs QBF ainsi que des instances qui pourront servir de base de test pour les évaluations des prouveurs soumis. Ces évaluations ont dans un premier temps permis de fixer le format d'entrée des prouveurs QBF. Le format choisi est QDimacs[8].

Une instance de formule au format QDimacs est composé de trois parties : le préambule, le préfixe de la formule et la matrice de la formule.

Préambule Le préambule contient des informations sur l'instance. Ces informations sont regroupées en lignes. Chacune de ces lignes commence par un caractère simple (suivi d'un espace) qui détermine le type de la ligne. Ces types sont les suivants :

Commentaire Les lignes de commentaire donnent des informations sur l'instance (e.g. son mode de génération) et sont ignorées par les prouveurs. Elles apparaissent au début du préambule et commencent par le caractère `c`.

Exemple

`c Ceci est un exemple de ligne de commentaire`

Problème Chaque fichier d'entrée ne contient qu'une ligne de problème. Elle doit apparaître après les commentaires (s'il y en a) et avant le préfixe. Les lignes de problème ont le format suivant :

`p cnf #VAR #CLS`

La chaîne de caractères en minuscules `p cnf` signifie que c'est la ligne de problème et que l'instance est une formule $\mathcal{CNF}$. Le champ `#VAR` (resp. `#CLS`) contient un entier positif spécifiant le nombre total de variables (resp. clauses) dans l'instance. Cette ligne doit apparaître à la dernière ligne du préambule.

Préfixe Le préfixe encode l'information concernant les quantificateurs et la façon dont ils sont appliqués à la proposition. Chaque ligne commence par une lettre suivie par un ensemble d'entiers positifs.

Les lettres sont

– `a` si la ligne représente une liste de quantificateurs universels,
– `e` si la ligne représente une liste de quantificateurs existentiels.

Chaque ligne se termine par un `0`.

Matrice La matrice représente les clauses et elle apparaît immédiatement après les lignes de préfixe. Les variables sont supposées être comprises entre `1` et `#VAR`. Toutes les variables ne doivent pas nécessairement apparaître dans l'instance. Chaque clause est représentée par une liste d'entiers séparés

8. http://www.qbflib.org/Draft/qDimacs.ps.gz

par un espace, une tabulation, ou un caractère de nouvelle ligne. Un littéral positif est représenté par `i` et sa négation par `-i`. Chaque clause est terminée par `0`. Ce format permet d'écrire les clauses sur plusieurs lignes.

Exemple 3.18 (QDimacs)
La $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ de l'exemple 3.17 s'écrit comme suit au format QDimacs :

```
c voici un exemple de QBF au format QDimacs
p cnf 4 5
e 1 0
a 2 0
e 3 4 0
-1 -2 3 0
1 -2 -4 0
-2 -3 0
2 3 -4 0
3 4 0
```

■

Par ailleurs, les évaluations réalisées permettent de comparer les différents prouveurs QBF de façon homogène. Cela donne lieu à une évolution des outils de comparaison (prouveurs et instances). En effet, depuis que cette évaluation existe, de nombreux progrès ont été réalisés dans le domaine, des nouveaux prouveurs ont été implantés et des nouvelles instances proposées.

En revanche, ces évaluations sont encore récentes et leurs organisateurs manquent encore de recul face aux $\mathcal{QBF}$. En particulier, contrairement à ce qui se passe pour le problème SAT, se pose le problème de la vérification des instances positives : comment vérifier qu'une instance $\mathcal{QBF}$ valide l'est effectivement ? Le seul « certificat » que fournissent ces évaluations repose sur la loi de la majorité : si la plupart des prouveurs s'accordent à annoncer une instance $\mathcal{QBF}$ valide (resp. incohérente), alors cette dernière est considérée comme valide (resp. incohérente). Un tel procédé est évidemment un pis-aller et il serait souhaitable de faire mieux. Cela est difficile et l'on sait en particulier que si la hiérarchie polynomiale ne s'effondre pas, il n'existe pas de certificats de taille polynomiale en $|\Sigma|$ pour toute $\mathcal{QBF}$ Σ (même celles de $2\mathcal{QBF}_\forall$) (Coste-Marquis *et al.* 2002).

4

Classes polynomiales pour QBF

Sommaire

« *À part l'argent, à part le confort, à part les diplômes, à part la frime, sur quoi peut donc reposer la distinction des classes ?* »
Jean-Marie Poupart[9]

4.1 Introduction

Puisque QBF est un problème calculatoirement difficile, il importe de mettre en évidence des restrictions plus faciles de ce problème, et si possible des restrictions traitables, i.e. reconnaissables et décidables en temps polynomial. C'est ce que l'on entend par classe polynomiale pour QBF.

Une recherche de telles classes a un intérêt évident du point de vue pratique (une instance d'une classe traitable est typiquement facile à résoudre avec un algorithme spécialisé) mais aussi théorique en permettant de préciser la frontière entre traitabilité et intraitabilité pour le problème de décision considéré, qu'il s'agisse de QBF ou d'autres problèmes comme SAT.

Ainsi, depuis de nombreuses années, les classes polynomiales pour SAT font l'objet d'études très approfondies (Aspvall *et al.* 1979; Even *et al.* 1976; Yamasaki & Doshita 1983). En particulier, le projet inter-PRC sur les classes polynomiales (Belleannée *et al.* 1995), ainsi que le projet BAHIA (Booléens : Algorithmes et Heuristiques pour l'intelligence Artificielle) en 1995 (BAHIA (Groupe) 1995), ont permis de faire avancer la recherche sur le sujet.

Dans sa thèse, E. Benoist (Benoist 2000) met l'accent sur l'utilité d'étudier des classes de formules pour lesquelles le test de satisfiabilité peut se faire en temps polynomial. Il s'agit de classes dérivées des formules de Horn qui conservent les propriétés de ces dernières.

L'étude de telles sous-classes de la logique propositionnelle peut, dans certains cas, s'étendre aux formules booléennes quantifiées. En particulier, H. Kleine-Büning et T. Lettmann s'intéressent à certaines de

9. Poupart (Jean-Marie), écrivain canadien né en 1947.

ces sous-classes et étendent des méthodes déjà connues par la communauté SAT aux formules booléennes quantifiées afin de déterminer certaines classes polynomiales pour QBF (Kleine-Büning & Lettmann 1999).

Dans ce chapitre, nous nous focalisons sur des restrictions de $\mathcal{CNF}$-$h\mathcal{QBF}$ obtenues en imposant aux matrices des instances de vérifier une condition particulière.

Dans une telle approche, un résultat important est le théorème de Schaefer pour QBF (Schaefer 1978). Ce résultat, beaucoup moins connu que le théorème de dichotomie de Schaefer pour SAT, quoique publié dans le même article, indique que les **seules** classes polynomiales pour QBF parmi celles « caractérisées localement » (i.e. par la nature des « clauses » des matrices des formules d'entrée) sont les classes $\mathcal{QKF}$, $\mathcal{QHF}$, reverse-$\mathcal{QHF}$ (définies dans ce chapitre) et $\mathcal{QAF}$ ($\mathcal{QBF}$ prénexes, polies, fermées dont la matrice est une conjonction de clauses XOR, cf. chapitre 5). Pour toutes les autres restrictions de $\mathcal{QBF}$ à matrice conjonctive et caractérisée localement, le problème de la validité reste **PSPACE**-complet. Notons bien que ce théorème de dichotomie ne rend pas compte de toutes les classes polynomiales définies par une condition sur la matrice de la formule d'entrée, en particulier la classe ren$\mathcal{QHF}$ des formules Horn renommables quantifiées décrite plus loin dans ce chapitre (cette classe n'est pas caractérisée localement dans le sens où il ne suffit pas de considérer chaque clause de la matrice d'une formule $\mathcal{CNF}$-$h\mathcal{QBF}$ séparément pour pouvoir décider si cette formule est ren$\mathcal{QHF}$ ou pas).

Nous présentons dans la suite de ce chapitre un aperçu des classes polynomiales existantes pour QBF. Nous exploitons la principale d'entre elles (ren$\mathcal{QHF}$) au chapitre 6.

4.2 Formules 2$\mathcal{CNF}$

Définition 4.1 (2$\mathcal{CNF}$-$\mathcal{QBF}$)
Une formule de 2$\mathcal{CNF}$-$\mathcal{QBF}$ (ou $\mathcal{QKF}$ pour **formule de Krom quantifiée***) est une formule de $\mathcal{CNF}$-$h\mathcal{QBF}$ dont la matrice est composée uniquement de clauses binaires.*

LA classe des $\mathcal{QKF}$ est une des classes de $\mathcal{QBF}$ démontrées polynomiales (Aspvall *et al.* 1979; Gent & Rowley 2002).

Exemple 4.1 ($\mathcal{QKF}$)
La formule suivante est une $\mathcal{QKF}$:

$$\forall a\ \exists b, c. \begin{bmatrix} (a \vee b) & \wedge \\ (\neg a \vee b) & \wedge \\ (\neg b \vee c) & \wedge \\ (\neg b \vee \neg c) & \end{bmatrix}.$$

■

La reconnaissance des $\mathcal{QKF}$ est évidemment triviale.

Nous présentons dans ce paragraphe l'algorithme BTOQSAT (Gent & Rowley 2002), destiné à résoudre les $\mathcal{QKF}$.

Algorithme de Gent et Rowley

L'algorithme 4.1 présente BTOQSAT, l'algorithme polynomial de I. P. Gent et G. D. Rowley pour résoudre QKF (Gent & Rowley 2002). Sa complexité en temps est en $\mathcal{O}(n \times m)$ pour toute instance dont la matrice contient n variables et m clauses.

Cet algorithme constitue une adaptation de l'algorithme de B. Aspvall, M. Plass et R. Tarjan afin de l'inclure plus simplement dans un prouveur QBF (Aspvall *et al.* 1979). Dans cet article, une $\mathcal{QKF}$ Σ est

Algorithme 4.1 : Algorithme proposé par Gent et Rowley pour résoudre les $\mathcal{QKF}$

fonction BTOQSAT
Données : une $\mathcal{QKF}\,\Sigma$
Résultat : Vrai si Σ est valide, Faux sinon.
début
  si $\Sigma = \{\}$ **alors**
    retourner Vrai ;
  si $\{\} \in \Sigma$ **alors**
    retourner Faux ;
  $v \leftarrow$ variable non affectée de Σ la plus externe dans l'ordre du préfixe ;
  $\Sigma' \leftarrow$ UNITPROPAGATION(Σ_v) ;
  si *($v \in \exists$* **et** *$\{\} \in \Sigma'$)* **ou** *($v \in \forall$* **et** *$\{\} \notin \Sigma'$)* **alors**
    $\Sigma' \leftarrow$ UNITPROPAGATION($\Sigma_{\neg v}$) ;
  retourner BTOQSAT*(Σ')* ;
fin

convertie en un graphe $G(\Sigma)$, avec un sommet pour chaque littéral l de $Lit(\Sigma)$ et son complémentaire l^c. Chaque clause $(x \vee y)$ de Σ est convertie en une paire d'arcs $\neg x \rightarrow y$ et $\neg y \rightarrow x$. Le graphe $G(\Sigma)$ est construit de telle façon que s'il existe un chemin d'un littéral x à un littéral y, alors il en existe un de y^c à x^c.

Exemple 4.2 (Graphe de $\mathcal{QKF}$)
La formule Σ de l'exemple 4.1 est associée au graphe $G(\Sigma)$ suivant :

$$
\begin{tikzcd}
a \arrow[r] & b \arrow[r] \arrow[dr] & c \arrow[dl] \\
\neg a \arrow[ur] & \neg b \arrow[l] \arrow[ul] & \neg c \arrow[l]
\end{tikzcd}
$$

■

Rappelons qu'une composante fortement connexe d'un graphe G est un ensemble maximal de sommets de G tels qu'il existe un chemin de chaque sommet vers tout autre sommet dans la composante.

La validité d'une $\mathcal{QKF}$ Σ peut s'exprimer sur la base des chemins et des composantes fortement connexes du graphe associé $G(\Sigma)$. Ainsi, une $\mathcal{QKF}$ Σ est fausse si et seulement si une des conditions suivantes est vérifiée (Aspvall *et al.* 1979) :

1. un littéral existentiel e est dans la même composante fortement connexe que son complémentaire e^c ;
2. un littéral universel u, quantifié par q_u, est dans la même composante fortement connexe qu'un littéral existentiel e, quantifié par q_e où q_e est plus externe que q_u ;
3. il existe un chemin d'un littéral universel u vers un autre littéral universel v (incluant le cas où $v = u^c$).

En effet :

(1) exprime le fait que Σ s'écrit (à l'équivalence logique près) comme $\Sigma \wedge \exists e.(e \Leftrightarrow e^c)$ qui est contradictoire ;

(2) exprime le fait que Σ s'écrit (à l'équivalence logique près) comme $\Sigma' \wedge \exists e\, \forall u.(e \Leftrightarrow u) \equiv \Sigma' \wedge \exists e\, \forall u.[(\neg e \vee u) \wedge (\neg u \vee e)]$. La partie $\exists e\, \forall u.[(\neg e \vee u) \wedge (\neg u \vee e)]$ n'est pas valide : si e est affecté

à Vrai (resp. Faux) et simplifiée par propagation unitaire, alors une $\forall$-clause universelle u (resp. $\neg u$) apparaît ce qui est contradictoire ;

(3) exprime le fait que Σ s'écrit (à l'équivalence logique près) comme $\Sigma' \wedge \forall u\, \forall v.(\neg v \vee u)$ qui contient une $\forall$-clause pure et est donc contradictoire.

L'algorithme présenté dans (Aspvall *et al.* 1979) consiste à déterminer dans le graphe $G(\Sigma)$ de la $\mathcal{QKF}$ Σ si une des trois conditions est vérifiée. Si c'est le cas, la $\mathcal{QBF}$ est fausse. Le problème avec cet algorithme est qu'il n'affecte aucune valeur aux variables de la $\mathcal{QBF}$. Ceci le rend difficile à incorporer dans un algorithme pour QBF généralisé de type Q-DPLL comme l'explique A. Del Val (Del Val 2000).

La fonction UNITPROPAGATION est la propagation unitaire de complexité linéaire adaptée aux $\mathcal{CNF}$-$h\mathcal{QBF}$ telle qu'elle est décrite au paragraphe 3.2.3.

4.3 Formules de Horn et reverse-Horn

Définition 4.2 (Formule de Horn quantifiée/Formule reverse-Horn quantifiée)
Une **formule de Horn quantifiée** *ou $\mathcal{QHF}$ (resp.* **formule reverse-Horn quantifiée** *ou reverse-$\mathcal{QHF}$) est une $\mathcal{CNF}$-$h\mathcal{QBF}$ dont la matrice est une formule de Horn $\mathcal{CNF}$ (resp. une formule reverse-Horn $\mathcal{CNF}$), i.e. une formule dont les clauses contiennent au plus un littéral positif (resp. négatif).*

Pour la suite, $\mathcal{QHF}$ (resp. reverse-$\mathcal{QHF}$) désigne l'ensemble des formules de Horn quantifiées (resp. formules reverse-Horn quantifiées).

Exemple 4.3 ($\mathcal{QHF}$/reverse-$\mathcal{QHF}$)
La formule suivante est une $\mathcal{QHF}$:

$$\forall a, b\, \exists c, d. \begin{bmatrix} (a \vee \neg b \vee \neg d) & \wedge \\ (\neg a \vee c) & \wedge \\ (\neg b \vee \neg c \vee d) & \wedge \\ (\neg a \vee \neg c \vee \neg d) & \end{bmatrix}.$$

La formule suivante est une reverse-$\mathcal{QHF}$:

$$\forall a, b\, \exists c, d. \begin{bmatrix} (\neg a \vee b \vee d) & \wedge \\ (a \vee \neg c) & \wedge \\ (b \vee c \vee \neg d) & \wedge \\ (a \vee c \vee d) & \end{bmatrix}.$$

■

Clairement, à l'instar du problème de reconnaissance des $\mathcal{QKF}$, le problème de reconnaissance des $\mathcal{QHF}$ et des reverse-$\mathcal{QHF}$ est trivial.

Algorithme de Kleine-Büning, Karpinski et Flögel

QKN, décrit en 1995 par H. Kleine-Büning, M. Karpinski et A. Flögel dans (Kleine-Büning *et al.* 1995) est, à notre connaissance, le premier prouveur QBF proposé. Les auteurs proposent un algorithme en temps polynomial, basé sur la résolution unitaire généralisée (Q-unit-résolution), capable de résoudre le problème QBF pour des formules $\mathcal{QHF}$.

La Q-unit-résolution constitue une généralisation de la résolution unitaire utilisée en logique propositionnelle et une restriction de la Q-résolution.

Définition 4.3 (Clause $\exists$-unit (positive))
Une clause est **$\exists$-unit** *si elle contient exactement un littéral existentiel et un nombre arbitraire de variables universelles.*
Une clause **$\exists$-unit positive** *est une clause unitaire avec exactement un littéral existentiel positif.*

Définition 4.4 (Q-unit-résolution)
Soit $\Sigma = QX_1 \dots QX_n.\phi$ une $\mathcal{CNF}$-$h\mathcal{QBF}$. Une **preuve d'une clause γ par Q-unit-résolution** *à partir de Σ est une suite finie $\gamma_1 \dots \gamma_m$ de clauses telles que :*

- $\gamma_m = \gamma$
- *pour $i = 1$ à m, γ_i est :*
  - *une clause de ϕ, ou*
  - *la résolvante de γ_j et γ_k avec $j, k < i$ à condition que les littéraux complémentaires de γ_j et γ_k utilisés pour produire γ_i correspondent à des variables quantifiées existentiellement et que γ_j ou γ_k soit $\exists$-unit positive, ou*
  - *la restriction de γ_j avec $j < i$ obtenue en supprimant de γ_j tous les littéraux associés à des variables quantifiées universellement v pour lesquelles il n'existe pas dans γ_j de littéraux associés à des variables quantifiées existentiellement qui suivraient v dans le préfixe de Σ.*

On note $\Sigma \vdash_{\overline{\textit{Q-Unit-Res}}} \gamma$ lorsque γ a une preuve par Q-unit-résolution à partir de Σ.

Exemple 4.4 (Q-unit-résolution)
Soit Σ la $\mathcal{QHF}$ de l'exemple 4.3.

$$\Sigma = \forall a, b\, \exists c, d. \begin{bmatrix} (a \vee \neg b \vee \neg d) & \wedge \\ (\neg a \vee c) & \wedge \\ (\neg b \vee \neg c \vee d) & \wedge \\ (\neg a \vee \neg c \vee \neg d) & \end{bmatrix}$$

$$\begin{aligned} \Sigma \quad & \vdash_{\overline{\textit{Q-Unit-Res}}} \quad \forall a, b\, \exists c, d. \begin{bmatrix} (a \vee \neg b \vee \neg d) & \wedge \\ (\neg a \vee c) & \wedge \\ (\neg b \vee \neg c \vee d) & \wedge \\ (\neg a \vee \neg c \vee \neg d) & \wedge \\ (\neg a \vee \neg b \vee d) & \end{bmatrix} \qquad \text{(cf. figure 4.1)} \\ & \vdash_{\overline{\textit{Q-Unit-Res}}} \quad \forall a, b\, \exists c, d. \begin{bmatrix} (a \vee \neg b \vee \neg d) & \wedge \\ (\neg a \vee c) & \wedge \\ (\neg b \vee \neg c \vee d) & \wedge \\ (\neg a \vee \neg c \vee \neg d) & \wedge \\ (\neg a \vee \neg b \vee d) & \wedge \\ (\neg b) & \end{bmatrix} \equiv \mathsf{Faux}. \end{aligned}$$

■

Kleine-Büning *et al.* montrent que la Q-unit-résolution est réfutationnellement complète pour les $\mathcal{QHF}$: si Σ est une $\mathcal{QHF}$, alors Σ est contradictoire si et seulement si $\Sigma \vdash_{\overline{\textit{Q-Unit-Res}}} \mathsf{Faux}$.

Les auteurs de (Kleine-Büning *et al.* 1995) montrent que le problème d'évaluation de $\mathcal{QHF}$ par Q-unit-résolution peut être décidé en temps $\mathcal{O}(|\Sigma| \times r)$, où r est le nombre d'occurrences positives de variables universelles dans Σ.

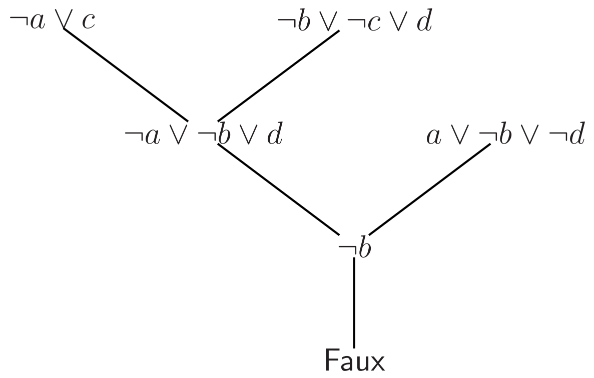


FIG. 4.1 – *Étapes de Q-unit-résolution pour* Σ

Il est aisé d'adapter l'algorithme proposé par Kleine-Büning *et al.* aux formules $\mathcal{CNF}$-$h\mathcal{QBF}$ à matrice reverse-Horn ; ceci est fait dans l'algorithme 4.2. Rappelons que si α désigne une clause, alors $\alpha_\exists = \{l \in \alpha \mid l \text{ est existentiel}\}$.

La fonction UNITPROPAGATION2 utilisée dans l'algorithme 4.2 représente une propagation unitaire standard pour $\mathcal{QBF}$ tel qu'elle est réalisée dans l'algorithme 4.1 mais ne se contente plus de retourner uniquement la formule résultante de la propagation unitaire ; elle renvoie également l'ensemble des variables affectées par propagation unitaire. Ainsi, l'instruction *<HF, UP>*← UNITPROPAGATION2*(HF);* retourne dans *HF* le résultat de la propagation unitaire et dans *UP* l'ensemble des variables affectées lors de cette opération.

De plus, certaines clauses n'ont pas à être considérées, soit parce qu'elles sont satisfaites, soit parce qu'elles sont marquées explicitement par la fonction MARQUER. Dans ce dernier cas, les clauses marquées sont alors exclues de la base de clauses par l'instruction

$$\text{HF} \leftarrow \text{HF} \setminus \{\alpha \mid \alpha \text{ est une clause marquée de HF}\}.$$

Enfin, la fonction DESAFFECTER(y_j) a pour rôle d'annuler l'affectation du littéral y_j. Cela peut avoir pour effet de réinsérer une clause dans HF si elle était satisfaite par y_j.

4.4 Formules Horn renommables

Intuitivement, une formule Horn renommable quantifiée est une formule qui peut être transformée en une $\mathcal{QHF}$ par renommage (i.e. passage au complémentaire) de certains de ses littéraux. Formellement,

Définition 4.5 (Formule Horn renommable quantifiée)
Une **formule Horn renommable quantifiée** *(ren$\mathcal{QHF}$)* $\Sigma = QX_1 \dots QX_h.\phi$ *est une* $\mathcal{CNF}$-$h\mathcal{QBF}$ *telle qu'il existe* $L_\Sigma \subseteq Lit(\Sigma)$ *et une substitution*

$$\begin{aligned} \sigma : L_{PS} &\to L_{PS} \\ l &\mapsto l^c \textit{ si } l \in L_\Sigma \\ l &\mapsto l \textit{ sinon} \end{aligned}$$

telle que $\hat\sigma(\phi)$ *est une formule de Horn* $\mathcal{CNF}$. $\hat\sigma$ *est l'extension de* σ *à* $\mathcal{NNF}_{PS}$ *définie par* $\hat\sigma(\mathsf{Vrai}) = \mathsf{Vrai}$, $\hat\sigma(\mathsf{Faux}) = \mathsf{Faux}$, $\hat\sigma(l) = \sigma(l)$ *si* l *est un littéral et* $\hat\sigma(\alpha \odot \beta) = \hat\sigma(\alpha) \odot \hat\sigma(\beta)$ *pour* $\odot \in \{\wedge, \vee\}$.

Algorithme 4.2 : Algorithme proposé par Kleine-Büning *et al.* pour résoudre les $\mathcal{QHF}$ (resp. reverse-$\mathcal{QHF}$)

fonction Q-UNIT-RES
Données : une $\mathcal{QHF}$ (resp. reverse-$\mathcal{QHF}$) Σ de la forme $Q_1 z_1 \dots Q_n z_n.(\alpha_1 \wedge \dots \wedge \alpha_m)$
Résultat : Vrai si Σ est valide, Faux sinon.
début
$\left.\begin{array}{l} \#P_\Sigma = \{\alpha_i \mid \alpha_i \text{ contient un littéral existentiel positif (resp. négatif)}\} \\ \#N_\Sigma^+ = \{\alpha_i \mid \alpha_i \text{ contient un littéral universel positif (resp. négatif)}\} \\ \#N_\Sigma = \{\alpha_i \mid \alpha_i \text{ ne contient que des littéraux négatifs (resp. positifs)}\} \end{array}\right\}$ partition de la matrice de Σ
    HF ← $\{\alpha_\exists \mid \alpha \in P_\Sigma \cup N_\Sigma\}$;
    HF ← UNITPROPAGATION(HF);
    si $\{\} \in$ *HF* **alors**
        retourner Faux ;
    pour chaque x_i *universel positif (resp. négatif) tel que* $x_i \in Lit(N_\Sigma^+)$ **faire**
        HF ← $\{\alpha_\exists \mid \alpha \in P_\Sigma\}$;
        MARQUER($\{\alpha \mid \alpha \in$ HF **et** $x_i \in \alpha\}$) ;
        <HF, UP>← UNITPROPAGATION2(HF) ;
        # UP est l'ensemble des variables affectées par propagation unitaire
        pour chaque $y_j \in$ *UP tel que* $j > i$ **faire**
            DESAFFECTER(y_j) ;
        MARQUER($\{\alpha \mid \alpha \in P_\Sigma$ **et** $y_j \in \alpha$ **et** $j < i\}$) ;
        HF ← HF $\setminus$ $\{\alpha \mid \alpha$ est une clause marquée de HF$\}$;
        HF ← HF $\cup$ $\{\alpha_\exists \mid \alpha \in N_\Sigma^+$ **et** $x_i \in \alpha\}$;
        HF ← UNITPROPAGATION(HF);
        si $\{\} \in$ *HF* **alors**
            retourner Faux ;
    retourner Vrai ;
fin

ren$\mathcal{QHF}$ désigne l'ensemble des formules Horn renommables quantifiées.

Exemple 4.5 (ren$\mathcal{QHF}$)
La formule suivante est une ren$\mathcal{QHF}$:

$$\forall a, b\, \exists c, d. \begin{bmatrix} (a \vee b \vee \neg d) & \wedge \\ (\neg a \vee \neg c) & \wedge \\ (b \vee c \vee d) & \wedge \\ (\neg a \vee c \vee \neg d) & \end{bmatrix}.$$

Il s'agit de la formule de l'exemple 4.3 dans laquelle nous avons renommé les variables b et c. ■

Contrairement au cas des formules $\mathcal{QKF}$ et $\mathcal{QHF}$, le problème de décider si une formule $\mathcal{CNF}$-$h\mathcal{QBF}$ est ren$\mathcal{QHF}$, s'il est décidable en temps polynomial, n'est pour autant pas trivial ; il importe donc de développer des algorithmes à cet effet.

Algorithme de Hébrard

Nous présentons ici l'algorithme de J. J. Hébrard (Hébrard 1994) de détection de formules Horn renommables pour deux raisons : (1) cet algorithme est exploité dans notre prouveur présenté au chapitre 6 et (2) cet algorithme, en plus de préciser si une formule est ren$\mathcal{QHF}$, réalise en même temps la construction d'un renommage lorsque celui-ci est possible. Cet algorithme d'Hébrard n'a pas besoin d'adaptation particulière aux $\mathcal{QBF}$ puisque la Horn renommabilité ne concerne pas le préfixe des formules.

Il nous faut au préalable définir certaines notions.

Définition 4.6 (Ensembles de littéraux complets)
Soient Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$ et L un ensemble de littéraux. L est **complet** *pour Σ si x appartient à L ou $\neg x$ appartient à L pour chaque x apparaissant dans Σ.*

Définition 4.7 (Renommage)
Soit Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$. Un **renommage** *pour Σ est un ensemble de littéraux cohérent et complet pour Σ. Les littéraux négatifs d'un renommage R pour Σ sont ceux qui doivent être renommés dans Σ. Formellement, à tout renommage R peut être associée une substitution $\sigma_R : L_{Var(\Sigma)} \rightarrow L_{Var(\Sigma)}$ telle que si $|l| \in R$, alors $\sigma_R(l) = l$, sinon $\sigma_R(l) = l^c$.*

Soit R un renommage pour Σ et $p \in PS$. Clairement, pour tout symbole propositionnel p, si $p \in R$, alors $R(p) = p$ et $R(\neg p) = \neg p$. Si $\neg p \in R$, alors $R(p) = \neg p$ et $R(\neg p) = p$.

Définition 4.8 ($\Rightarrow_\Sigma$, $\Rightarrow^*_\Sigma$ et $\text{C}{\small\text{LOS}}_\Sigma(l)$)
Soit Σ une $\mathcal{CNF}$-$h\mathcal{QBF}$.

- *$\Rightarrow_\Sigma$ est la relation binaire sur $L_{Var(\Sigma)}$ définie par $l \Rightarrow_\Sigma t$ si et seulement s'il existe une clause $C \in \Sigma$ telle que $l \in C$, $t^c \in C$ et $l \neq t^c$.*
- *La fermeture réflexive transitive de $\Rightarrow_\Sigma$ est notée $\Rightarrow^*_\Sigma$.*
- *$\text{C}{\small\text{LOS}}_\Sigma(l)$ désigne l'ensemble $\{t \mid l \Rightarrow^*_\Sigma t\}$.*

Exemple 4.6 ($\Rightarrow_\Sigma$, $\Rightarrow^*_\Sigma$)
Soit une formule $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ de matrice suivante :

$$\begin{bmatrix} (a \vee b \vee c) & \wedge \\ (a \vee \neg d \vee e) & \wedge \\ (b \vee \neg c \vee \neg d) & \wedge \\ (\neg c \vee \neg e) & \wedge \\ (\neg b \vee c) & \end{bmatrix}.$$

La première clause montre que $a \Rightarrow_\Sigma \neg b$ et $a \Rightarrow_\Sigma \neg c$; la troisième montre que $\neg c \Rightarrow_\Sigma d$; enfin $a \Rightarrow^*_\Sigma d$ car $a \Rightarrow_\Sigma \neg c$ et $\neg c \Rightarrow_\Sigma d$.
Nous illustrons $\textsc{Clos}_\Sigma(l)$ à l'exemple 4.7 ci-après. ■

Définition 4.9 (Littéraux fermés)
Un ensemble de littéraux $L \subseteq L_{PS}$ est dit **fermé** *pour Σ si $\textsc{Clos}_\Sigma(l) \subseteq L$, quel que soit $l \in L$.*

Proposition 4.1 (Horn renommage (proposition 1.1 de (Hébrard 1994)))
Soit Σ une $\mathcal{CNF}$-$h\mathcal{QHF}$. Un renommage R pour Σ est un Horn renommage pour Σ (i.e. $\hat{\sigma}_R(\Sigma)$ est une $\mathcal{QHF}$) si et seulement s'il est fermé.

L'algorithme 4.3 présente l'algorithme proposé par J. J. Hébrard.

Algorithme 4.3 : Algorithme de détection de Horn renommabilité de J. J. Hébrard

fonction RHCLOS
Données : une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ
Résultat : $\emptyset$ si Σ n'est pas une ren$\mathcal{QHF}$, un Horn renommage R pour Σ sinon.
début
  $R \leftarrow \emptyset$;
  pour chaque $p \in Var(\Sigma)$ **faire**
    si $p \notin R$ **et** $\neg p \notin R$ **alors**
      si $\forall q \in \textsc{Clos}_\Sigma(p) \setminus R,\ q^c \notin \textsc{Clos}_\Sigma(p) \setminus R$ **alors**
        # $\textsc{Clos}_\Sigma(p) \setminus R$ est cohérent
        $R \leftarrow R \cup (\textsc{Clos}_\Sigma(p) \setminus R)$;
      sinon si $\forall q \in \textsc{Clos}_\Sigma(\neg p) \setminus R,\ q^c \notin \textsc{Clos}_\Sigma(\neg p) \setminus R$ **alors**
        # $\textsc{Clos}_\Sigma(\neg p) \setminus R$ est cohérent
        $R \leftarrow R \cup (\textsc{Clos}_\Sigma(\neg p) \setminus R)$;
      sinon retourner $\emptyset$;
fin

Exemple 4.7 (Horn renommage)
Nous illustrons ici le fonctionnement de l'algorithme 4.3 et montrons que la formule Σ de l'exemple 4.5 est Horn renommable. Pour cela, prenons $\neg a$ comme le premier littéral pour lequel nous calculons la fermeture :

- $\neg a$ apparaît dans les clauses $(\neg a \vee \neg c)$ et $(\neg a \vee c \vee \neg d)$ de la matrice de Σ ;
- $(\neg a \vee \neg c) \rightsquigarrow \textsc{Clos}_\Sigma(\neg a) = \{\neg a, c\}$;
- $(\neg a \vee c \vee \neg d) \rightsquigarrow \textsc{Clos}_\Sigma(\neg a) = \{\neg a, c, \neg c, d\}$;
- $\{\neg a, c, \neg c, d\} \equiv \mathsf{Faux}$.

Nous obtenons une contradiction avec le choix du littéral $\neg a$. Il faut dont calculer la fermeture de a :

- a apparaît dans la clause $(a \vee b \vee \neg d)$;
- $(a \vee b \vee \neg d) \rightsquigarrow \textsc{Clos}_\Sigma(a) = \{a, \neg b, d\}$;
- $\neg b$ n'apparaît dans aucune clause ;
- d apparaît dans la clause $(b \vee c \vee d)$;
- $(b \vee c \vee d) \rightsquigarrow \textsc{Clos}_\Sigma(a) = \{a, \neg b, \neg c, d\}$;
- $\neg c$ apparaît dans la clause $(\neg a \vee \neg c)$;

– $\text{CLOS}_\Sigma(a) = \{a, \neg b, \neg c, d\}$.

Ceci prouve que la formule de l'exemple 4.5 est Horn renommable et qu'il faut renommer les variables b et c pour obtenir une formule Horn $\mathcal{CNF}$. ■

Une fois un renommage R calculé (s'il existe) pour une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ, il suffit d'utiliser l'algorithme de Kleine-Büning *et al.* (cf. paragraphe 4.3) sur la $\mathcal{QBF}$ $\hat{\sigma}_R(\Sigma)$ pour déterminer en temps polynomial si Σ est valide ou pas.

4.5 Quelques autres classes

De nombreuses autres classes ont été démontrées polynomiales pour SAT. Nous pouvons citer parmi celles-ci les classes référencées dans (Gu *et al.* 1997) mais également dans (Benoist 2000) :

- les formules Horn généralisées (ou formules de la classe $\mathbb{S}_0$) (Arvind & Biswas 1987; Benoist 2000; Yamasaki & Doshita 1983) ;
- les formules ordonnées (Benoist & Hébrard 1999; Benoist 2000),
- les formules ordonnées renommables (Benoist & Hébrard 1999; Benoist 2000),
- les formules q-Horn (Boros *et al.* 1994),
- les formules Horn étendues simples (Swaminathan & Wagner 1995; Benoist & Hébrard 2003),
- les formules Horn élargies simples (Benoist & Hébrard 2003).

Le tableau 4.1 présente un ensemble de références dans lesquelles on peut trouver des algorithmes de reconnaissance ou de test de satisfiabilité (recherche de solution) pour ces classes.

Classe	Références bibliographiques	
	Reconnaissance	Recherche de solution
2CNF	(Aspvall *et al.* 1979; Even *et al.* 1976; Knuth 1990)	(Aspvall *et al.* 1979; Even *et al.* 1976; Knuth 1990; Del Val 2000)
Horn	(Dowling & Gallier 1984; Itai & Makowsky 1982; Scutella 1990; Yamasaki & Doshita 1983)	(Dowling & Gallier 1984; Itai & Makowsky 1982; Scutella 1990; Schlipf *et al.* 1995; Minoux 1988; Yamasaki & Doshita 1983; Ghallab & Escalada-Imaz 1991)
Horn Renommables	(Aspvall 1980; Lewis 1978; Del Val 2000; Chandru *et al.* 1990; Eiter *et al.* 1995)	(Hébrard 1994; Schlipf *et al.* 1995; Génisson & Rauzy 1995; Del Val 2000)
Horn Généralisées	(Yamasaki & Doshita 1983)	(Arvind & Biswas 1987)
Ordonnées	(Benoist & Hébrard 1999; Benoist 2000)	(Benoist & Hébrard 1999; Benoist 2000)
Ordonnées Renommables	(Benoist & Hébrard 1999; Benoist 2000)	(Benoist & Hébrard 1999; Benoist 2000)
q-Horn	(Boros *et al.* 1994)	(Boros *et al.* 1994)
Horn étendues simples	(Swaminathan & Wagner 1995; Benoist & Hébrard 2003)	(Benoist & Hébrard 2003)
Horn élargies simples	(Benoist & Hébrard 2003)	(Schlipf *et al.* 1995)

TAB. 4.1 – Tableau récapitulatif des classes polynomiales pour SAT (état de l'art).

De façon triviale, il est aisé de montrer que s'il existe un algorithme polynomial pour SAT permettant de déterminer si une formule $\mathcal{CNF}$ ϕ appartient à l'une de ces classes, alors cet algorithme est également polynomial pour QBF. En effet, les restrictions réalisées pour définir l'ensemble des classes citées sont faites sur la forme de la matrice d'une $\mathcal{CNF}$-$h\mathcal{QBF}$. En revanche, on ne peut déterminer aussi facilement si ces classes, polynomiales pour SAT, le sont encore pour QBF – sauf pour le cas trivial des $\mathcal{CNF}$-$2\mathcal{QBF}$ de la forme $\exists X\ \forall Y.\phi$ pour lequel on peut se ramener facilement au cas propositionnel « classique » car on sait éliminer en temps polynomial l'ensemble des variables universelles Y dans ϕ.

Troisième partie

Contributions

5

Complexité de QBF pour certains fragments propositionnels

Sommaire

« *Tout ce que j'ai publié n'est que des fragments d'une grande confession.* »
Goethe [10], extrait de ses *Mémoires*.

5.1 Introduction

Dans ce chapitre, nous considérons certains fragments traitables pour SAT, utilisés comme des langages cibles pour la compilation de « connaissances ». Pour chaque fragment $\mathcal{C}$ considéré, nous nous focalisons sur la restriction du problème QBF obtenu en imposant à la matrice de la formule d'entrée d'appartenir à ce fragment.

La compilation de « connaissances » a été utilisée ces dernière années afin d'améliorer (du point de vue calculatoire) le traitement des tâches basiques tirées de domaines majeurs en intelligence artificielle (IA), comme l'inférence (aussi bien classique que non monotone, cf. parmi d'autres (Darwiche & Marquis 2004; Coste-Marquis & Marquis 2001; Boufkhad *et al.* 1997; Selman & Kautz 1996; Del Val 1994; Schrag 1996; Coste-Marquis & Marquis 2004)), la planification (voir e.g. (Cimatti *et al.* 1997; Geffner 2004)) et le diagnostic (voir e.g. (Coste-Marquis & Marquis 1998; Darwiche 1999)). Ces approches consistent typiquement

10. Goethe, écrivain et savant allemand, né à Francfort-sur-le-Main le 28 août 1749, décédé à Weimar le 22 mars 1832. Fils d'une famille bourgeoise fortunée, Johann Wolfgang Von Goethe reçoit une éducation approfondie : il lit à trois ans et connaît le latin et le grec à sept ans. Sa vie est ponctuée de rencontres féminines qui furent souvent à l'origine de créations poétiques. Il se passionne pour la musique et fait la connaissance de Mozart et Beethoven. Ce dernier compose la musique pour accompagner une des oeuvres de l'écrivain : « Egmont ». Maîtrisant tous les genres : poésie, théâtre, roman... son oeuvre immense a placé l'Allemagne, pendant plus d'un demi-siècle, au premier plan littéraire. Ses deux chefs-d'oeuvre universels, « Faust » et « Les souffrances du jeune Werther » ont influencé toute l'Europe et ont traversé les générations en conservant intact tout leur génie.

à transformer, au cours d'une phase de pré-traitement, des informations représentées par une formule propositionnelle en une formule extraite d'un fragment « plus traitable ». « Plus traitable » signifie que les tâches requises par l'application considérée deviennent plus facilement calculables, et si possible, en temps polynomial (Darwiche & Marquis 2002). Le problème SAT fait partie de ces tâches. Parmi les fragments « traitables » connus de SAT (et relativement significatifs), citons les diagrammes de décision binaires, les impliqués premiers, les impliquants premiers, les « formules » sous forme normale négative décomposable.

Dans la suite de ce chapitre, nous exhibons la complexité de QBF pour des fragments propositionnels complets et incomplets, où un fragment propositionnel $\mathcal{C}$ est complet si et seulement si toute formule propositionnelle a un équivalent dans $\mathcal{C}$. Nous nous intéressons principalement aux fragments considérés dans (Darwiche & Marquis 2002) : $\mathcal{DNF}$, d-$\mathcal{DNNF}$, $\mathcal{DNNF}$, $\mathcal{OBDD}_<$, $\mathcal{FBDD}$, $\mathcal{PI}$, $\mathcal{IP}$, $\mathcal{MODS}$ dont l'importance dans de nombreux domaines d'IA est démontrée. Nous complétons les résultats donnés dans (Darwiche & Marquis 2002) en nous focalisant sur une question supplémentaire : celle de la validité des $\mathcal{QBF}$. Nous présentons des résultats de complexité pour QBF réduit à ces fragments.

5.2 Classes traitables et intraitables pour QBF

Un fragment propositionnel (i.e. un sous-ensemble d'un langage propositionnel) est dit traitable pour QBF (resp. pour SAT) si et seulement si l'appartenance à ce fragment peut être décidée en temps polynomial et s'il existe également un algorithme de décision polynomial pour le problème de validité des formules booléennes quantifiées (fermées, polies, prénexes) dont la matrice appartient à ce fragment (resp. s'il existe un algorithme de décision polynomial pour le problème de satisfiabilité pour les formules de ce fragment).

5.2.1 Fragments incomplets

Considérons tout d'abord les fragments propositionnels incomplets, i.e. les fragments qui ne sont pas suffisamment expressifs pour permettre la représentation de toutes les formules propositionnelles.

Nous avons déjà présenté au chapitre 4 le théorème de Schaefer pour QBF : parmi les restrictions de $h\mathcal{QBF}$ aux formules à matrices conjonctives, définies par la nature des « clauses » contenues dans les matrices, seules les classes $\mathcal{QKF}$, $\mathcal{QHF}$, reverse-$\mathcal{QHF}$ et celles des formules affines sont polynomiales pour QBF. Toutes les autres restrictions restent **PSPACE**-complètes.

Clairement, ce théorème de dichotomie pour QBF[11] ne caractérise pas toutes les classes polynomiales pour QBF tout comme le théorème de dichotomie pour SAT ne caractérise pas toutes les classes polynomiales pour SAT, comme les diagrammes de décision binaires ordonnés ou les formules Horn renommables $\mathcal{CNF}$. En particulier, il ne permet pas de montrer que ren$\mathcal{QHF}$ est une classe polynomiale pour QBF (cf. chapitre 4).

Le théorème de dichotomie de Schaefer pour QBF ne fournit pas non plus directement d'algorithmes en temps polynomial pour la validité de QBF restreint aux classes $\mathcal{QKF}$, $\mathcal{QHF}$, reverse-$\mathcal{QHF}$ et celle des formules affines. Nous avons déjà présenté de tels algorithmes pour $\mathcal{QKF}$, $\mathcal{QHF}$ et reverse-$\mathcal{QHF}$ au chapitre 4. Dans un souci de complétude, nous présentons ici un algorithme pour QBF dans le cas des formules affines. Cet algorithme procède par élimination des quantifications. Commençons par définir formellement la classe des formules affines quantifiées.

Définition 5.1 (Formules affines quantifiées)

11. Dans un passé très récent, QCSP, le problème de satisfaction de contraintes quantifiées – une généralisation de QBF lorsque des contraintes conjonctives sont considérées – a reçu plus d'attention ; des théorèmes de classification donnant la complexité des QCSP dépendant des propriétés algébriques du langage de contraintes considéré ont été démontrés (Börner *et al.* 2003; Chen 2004; Chen 2005). Les résultats obtenus concernant seulement les contraintes conjonctives, ils ne peuvent donc pas être utilisés directement pour identifier la complexité de chaque restriction de QCSP.

Une **formule affine quantifiée** *de* $QPROP_{PS}$ *est une formule polie, prénexe, fermée dont la matrice est une formule propositionnelle affine, i.e. une conjonction de formules de la forme* $x_1 \oplus \ldots \oplus x_k$[12] *où chaque* x_i *(*$i \in 1 \ldots k$*) est un symbole de* PS *ou l'une des constantes booléennes* Vrai*,* Faux*.*

Pour la suite, $\mathcal{QAF}$ désigne les formules affines quantifiées.

Exemple 5.1 ($\mathcal{QAF}$**)**
La formule suivante est affine :

$$\Sigma = \exists\, x_1\, \forall\, x_2\, \exists\, x_3, x_4\, . \begin{bmatrix} (x_2 \oplus x_3 \oplus x_4) & \wedge \\ (x_1 \oplus x_2 \oplus x_4) & \wedge \\ (x_1 \oplus x_2 \oplus \mathsf{Faux}) & \end{bmatrix} .$$

■

Remarquons que, ni la classe des formules Krom $\mathcal{CNF}$, ni la classe des formules Horn renommables $\mathcal{CNF}$[13] n'est plus expressive que celle des formules affines (il n'existe pas de formule Horn renommable $\mathcal{CNF}$ équivalente à la formule affine $x_1 \oplus x_2 \oplus x_3$).

Comme pour les classes $\mathcal{QKF}$, $\mathcal{QHF}$ et reverse-$\mathcal{QHF}$, il est évident que le problème de l'appartenance à la classe $\mathcal{QAF}$ est traitable.

L'algorithme 5.1 pour la validité de $\mathcal{QAF}$ procède en éliminant les quantifications de la formule d'entrée de la plus interne vers la plus externe. Une fois toutes les quantifications éliminées, deux possibilités demeurent pour la formule propositionnelle résultante : soit il s'agit de la conjonction vide, et l'instance est positive (i.e., la $\mathcal{QBF}$ est valide), soit elle se réduit à Faux (et l'instance est négative).

Pour prouver la correction de cet algorithme, montrons, dans un premier temps, comment procéder à l'élimination des quantifications existentielles. Plus précisément, montrons comment associer à toute formule $\mathcal{QBF}$ polie et prénexe (pas nécessairement fermée) dont la matrice est affine et dont le préfixe ne contient que des quantificateurs existentiels, une formule affine équivalente et sans quantification. Considérons une $\mathcal{QBF}$ de la forme $\exists x.\phi$ où ϕ est affine. Nous supposons, sans perte de généralité, que toute clause XOR $x_1 \oplus \ldots \oplus x_k$ est simple, dans le sens où elle ne contient pas plus d'une occurrence de chaque variable (si ce n'est pas le cas, il suffit d'exploiter les deux équivalences $x \oplus x \equiv \mathsf{Faux}$ et $\alpha \oplus \mathsf{Faux} \equiv \alpha$ pour simplifier chaque clause XOR de ϕ (i.e., la rendre simple) ; évidemment, un tel processus de simplification est polynomial en temps dans la taille de ϕ). À présent, deux cas apparaissent : (1) soit x n'apparaît pas dans ϕ, soit (2) x apparaît dans au moins une clause XOR $x \oplus x_2 \oplus \ldots \oplus x_k$ de ϕ (où chaque x_i ($i \in 2 \ldots k$) est une variable ou une constante booléenne Vrai). Dans le cas (1), nous avons $\exists x.\phi \equiv \phi$; dans le cas (2), ϕ est équivalente à la conjonction de la formule $x \oplus x_2 \oplus \ldots \oplus x_k$ avec la formule ψ obtenue en remplaçant toute occurrence de x par $x_2 \oplus \ldots \oplus x_k \oplus \mathsf{Vrai}$ dans toutes les clauses XOR de ϕ, exceptée $x \oplus x_2 \oplus \ldots \oplus x_k$; ainsi, dans tout modèle de $x \oplus x_2 \oplus \ldots \oplus x_k$, la valeur de vérité de x coïncide avec la valeur de vérité de $x_2 \oplus \ldots \oplus x_k \oplus \mathsf{Vrai}$; cette méthode revient à l'élimination gaussienne (effectivement, toute formule affine peut être considérée comme un système d'équations linéaires sur l'anneau booléen $\mathbb{Z}/2\mathbb{Z}$, voir e.g. (Zanuttini 2002)). Donc nous avons $\exists x.\phi \equiv \exists x.((x \oplus x_2 \oplus \ldots \oplus x_k) \wedge \psi)$ et ψ peut être construit en temps polynomial dans la taille de ϕ. Puisque x n'apparaît pas dans ψ, nous avons $\exists x.\phi \equiv (\exists x.(x \oplus x_2 \oplus \ldots \oplus x_k)) \wedge \psi$ (cf. proposition 1.1). Puisque $\exists x.(x \oplus x_2 \oplus \ldots \oplus x_k) \equiv \mathsf{Vrai}$, nous obtenons finalement que $\exists x.\phi \equiv \psi$.

Montrons à présent comment traiter les quantifications universelles. Puisque pour toute $\mathcal{QBF}$ α, β et toute variable $x \in PS$, nous avons $\forall x.(\alpha \wedge \beta) \equiv (\forall x.\alpha) \wedge (\forall x.\beta)$ (cf. proposition 1.1, nous devons principalement considérer la situation où la portée des quantificateurs est de la forme $x_1 \oplus \ldots \oplus x_k$ où

12. Puisque $\oplus$ est associative, nous conservons cette notation en omettant les parenthèses.
13. Notons que toute formule Krom $\mathcal{CNF}$ qui est satisfiable est aussi une formule Horn renommable $\mathcal{CNF}$.

chaque x_i ($i \in 1 \ldots k$) est un symbole propositionnel ou une constante booléenne. Comme précédemment, il existe deux cas : (1) soit x n'apparaît pas dans $x_1 \oplus \ldots \oplus x_k$ soit (2) il existe un $i \in 1 \ldots k$ tel que $x = x_i$. Dans le cas (1), nous avons $\forall x.(x_1 \oplus \ldots \oplus x_k) \equiv x_1 \oplus \ldots \oplus x_k$; dans le cas (2), nous avons $\forall x.(x_1 \oplus \ldots \oplus x_k) \equiv \mathsf{Faux}$. Par conséquent, toute $\mathcal{QBF}$ de la forme $\forall X.\phi$ où ϕ est affine est équivalente à ϕ si $Var(\phi) \cap X = \emptyset$, et équivalente à Faux sinon.

Dans l'algorithme 5.1, la fonction SIMPLIFIER2 permet de rendre la formule simple comme expliqué ci-avant. De plus, si γ est une clause XOR, x une variable et α une clause XOR, alors $\gamma_{x \leftarrow \alpha}$ revient à remplacer x par α dans γ. Enfin, dans cet algorithme, toute formule de la forme $x_1 \oplus \ldots \oplus x_k$ est vue comme l'ensemble $\{x_1, \ldots, x_k\}$ et toute formule propositionnelle comme un ensemble de formules de la forme $x_1 \oplus \ldots \oplus x_k$.

Algorithme 5.1 : Algorithme en temps polynomial pour la validité des $\mathcal{QAF}$

fonction SAFFINE
Données : Une $\mathcal{QAF}$ $Q_1 x_1 \ldots Q_n x_n.\phi$ où ϕ est une formule propositionnelle affine
Résultat : Vrai si $Q_1 x_1 \ldots Q_n x_n.\phi$ est valide, Faux sinon
début
- $\phi \leftarrow$ SIMPLIFIER2(ϕ) ;
- **pour** *i* **de** *n* **à** *1* **faire**
  - **si** $x_i \in \exists$ **alors**
    - **si** $\exists \gamma \in \phi \mid x_i \in \gamma$ **alors**
      - $\phi \leftarrow \{\gamma'_{x_i \leftarrow \gamma \setminus \{x_i\} \cup \{\mathsf{Vrai}\}} \mid \gamma' \in \phi \setminus \{\gamma\}\}$;
  - **sinon**
    - **si** $x_i \in Var(\phi)$ **alors**
      - **retourner** Faux ;
  - SIMPLIFIER(ϕ) ;
- **retourner** Vrai ;

fin

Exemple 5.2 (Résolution de formules affines quantifiées)
Considérons de nouveau la formule de l'exemple 5.1 :

$$\Sigma = \exists\, x_1 \,\forall\, x_2 \,\exists\, x_3, x_4 \,.\, \left[\begin{array}{ll} (x_2 \oplus x_3 \oplus x_4) & \wedge \\ (x_1 \oplus x_2 \oplus x_4) & \wedge \\ (x_1 \oplus x_2 \oplus \mathsf{Faux}) & \end{array}\right].$$

Il faut commencer par simplifier la formule. On obtient :

$$\Sigma \equiv \exists\, x_1 \,\forall\, x_2 \,\exists\, x_3, x_4 \,.\, \left[\begin{array}{ll} (x_2 \oplus x_3 \oplus x_4) & \wedge \\ (x_1 \oplus x_2 \oplus x_4) & \wedge \\ (x_1 \oplus x_2) & \end{array}\right].$$

Procédons ensuite à l'élimination de la variable existentielle x_4. Alors, comme x_4 apparaît dans $(x_2 \oplus x_3 \oplus x_4)$,

$$\Sigma \equiv \begin{array}{c} \exists\, x_1 \,\forall\, x_2 \,\exists\, x_3 \,. \\ \left[\begin{array}{ll} (x_1 \oplus x_2 \oplus x_2 \oplus x_3) & \wedge \\ (x_1 \oplus x_2) & \end{array}\right] \end{array} \equiv \begin{array}{c} \exists\, x_1 \,\forall\, x_2 \,\exists\, x_3 \,. \\ \left[\begin{array}{ll} (x_1 \oplus \mathsf{Faux} \oplus x_3) & \wedge \\ (x_1 \oplus x_2) & \end{array}\right] \end{array} \equiv \begin{array}{c} \exists\, x_1 \,\forall\, x_2 \,\exists\, x_3 \,. \\ \left[\begin{array}{ll} (x_1 \oplus x_3) & \wedge \\ (x_1 \oplus x_2) & \end{array}\right] \end{array}.$$

De même, avec la variable existentielle x_3

$$\Sigma \equiv \exists\, x_1 \,\forall\, x_2 \,.\, [(x_1 \oplus x_2)]\,.$$

Enfin, comme x_2 est universelle et qu'elle apparaît dans la formule obtenue après élimination des autres variables plus internes, alors

$$\Sigma \equiv \mathsf{Faux}.$$

■

Il est clair que la simplification de formule affine peut être réalisé en temps polynomial, donc que l'algorithme 5.1 opère en temps polynomial lui aussi.

5.2.2 Fragments complets

Nous étudions maintenant les fragments complets. Commençons par présenter les résultats d'intraitabilité. Comme évoqué au chapitre 1, la restriction de QBF obtenue en imposant à la matrice d'être une formule $\mathcal{CNF}$ est toujours **PSPACE**-complet. En effet, toute formule propositionnelle Σ sur $\{x_1, \ldots, x_n\}$ peut être associée en temps linéaire à une formule $\mathcal{CNF}$ Σ' sur $\{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ telle que $\Sigma \equiv \exists\{y_1, \ldots, y_m\}.\Sigma'$. Une telle réduction qui préserve la satisfiabilité (et bien plus) est typiquement utilisée pour montrer que CIRCUIT-SAT peut être réduit à SAT restreint aux formules $\mathcal{CNF}$ (l'idée est d'introduire une nouvelle variable y_i par entrée, cf. (Papadimitriou 1994) page 163).

Exemple 5.3 (Transformation polynomiale d'une $\mathcal{QBF}$ polie, fermée, prénexe à une $\mathcal{CNF}$-$\mathcal{QBF}$)
La formule Σ

$$((a \wedge u) \vee ((\neg a) \vee b)) \wedge (a \vee ((\neg v) \wedge c))$$

est équivalente à la formule

$$\exists y_1, y_2, y_3, y_4, y_5, y_6. \left[\begin{array}{l} y_1 \wedge (y_1 \Leftrightarrow (y_2 \wedge y_3))\wedge \\ (y_2 \Leftrightarrow (y_4 \vee y_5)) \wedge (y_4 \Leftrightarrow (a \wedge u))\wedge \\ (y_5 \Leftrightarrow ((\neg a) \vee b))\wedge \\ (y_3 \Leftrightarrow (a \vee y_6)) \wedge (y_6 \Leftrightarrow (\neg v \wedge c)) \end{array}\right],$$

qui est encore équivalente à la formule suivante[14] dont la matrice est en $\mathcal{CNF}$:

$$\exists y_1, y_2, y_3, y_4, y_5, y_6. \left[\begin{array}{l} y_1 \wedge (\neg y_1 \vee y_2)\wedge \\ (\neg y_1 \vee y_3) \wedge (y_1 \vee \neg y_2 \vee \neg y_3)\wedge \\ (\neg y_2 \vee y_4 \vee y_5) \wedge (y_2 \vee \neg y_4)\wedge \\ (y_2 \vee \neg y_5) \wedge (\neg y_4 \vee a) \wedge (\neg y_4 \vee u)\wedge \\ (y_4 \vee \neg a \vee \neg u) \wedge (\neg y_5 \vee \neg a \vee b)\wedge \\ (y_5 \vee a) \wedge (y_5 \vee \neg b) \wedge (\neg y_3 \vee a \vee y_6)\wedge \\ (y_3 \vee \neg a) \wedge (y_3 \vee \neg y_6) \wedge (\neg y_6 \vee \neg v)\wedge \\ (\neg y_6 \vee c) \wedge (y_6 \vee v \vee \neg c) \end{array}\right].$$

Cette dernière formule peut être construite en temps polynomial en la taille de la formule de départ. De plus, d'après le méta-théorème de substitution pour les $\mathcal{QBF}$ (cf. chapitre 1), on obtient que :

$$\exists a, b, c\, \forall u, v.(((a \wedge u) \vee ((\neg a) \vee b)) \wedge (a \vee ((\neg v) \wedge c)))$$

14. Nous n'avons pas introduit de nouvelles variables pour les occurrences de $\neg$ puisque la portée de telles occurrences se réduit à une variable (Σ est une formule $\mathcal{NNF}$).

est équivalente à la $\mathcal{QBF}$ suivante dont la matrice est en $\mathcal{CNF}$:

$$\exists a, b, c\, \forall u, v\, \exists y_1, y_2, y_3, y_4, y_5, y_6. \left[\begin{array}{l} y_1 \wedge (\neg y_1 \vee y_2) \wedge (\neg y_1 \vee y_3) \wedge \\ (y_1 \vee \neg y_2 \vee \neg y_3) \wedge (\neg y_2 \vee y_4 \vee y_5) \wedge \\ (y_2 \vee \neg y_4) \wedge (y_2 \vee \neg y_5) \wedge (\neg y_4 \vee a) \wedge \\ (\neg y_4 \vee u) \wedge (y_4 \vee \neg a \vee \neg u) \wedge \\ (\neg y_5 \vee \neg a \vee b) \wedge (y_5 \vee a) \wedge (y_5 \vee \neg b) \wedge \\ (\neg y_3 \vee a \vee y_6) \wedge (y_3 \vee \neg a) \wedge (y_3 \vee \neg y_6) \wedge \\ (\neg y_6 \vee \neg v) \wedge (\neg y_6 \vee c) \wedge (y_6 \vee v \vee \neg c) \end{array}\right].$$

■

Considérons à présent la restriction de QBF lorsque les matrices des formules d'entrée considérées appartiennent à un fragment cible pour la compilation de « connaissances » ; de tels fragments sont nombreux à avoir été identifiés dans la littérature : $\mathcal{DNF}$, d-$\mathcal{DNNF}$, $\mathcal{DNNF}$, $\mathcal{FBDD}$, $\mathcal{OBDD}_<$, $\mathcal{MODS}$, $\mathcal{PI}$, $\mathcal{IP}$, ... Comme nous le verrons, QBF reste typiquement intraitable pour de telles restrictions.

PSPACE étant fermé pour la complémentation et la négation d'une $\mathcal{QBF}$ à matrice $\mathcal{CNF}$ étant une $\mathcal{QBF}$ à matrice $\mathcal{DNF}$, nous pouvons conclure directement que la restriction de QBF au cas où la matrice de la formule d'entrée est une formule $\mathcal{DNF}$ est également **PSPACE**-complet. Cela nous permet d'éviter de considérer de nombreux fragments traitables pour SAT comme des candidats intéressants pour QBF. Parmi ceux-ci, citons tous les sur-ensembles de $\mathcal{DNF}$, incluant le fragment $\mathcal{DNNF}$ et les disjonctions de formules Horn $\mathcal{CNF}$ qui sont des classes cibles pour certaines méthodes de compilation de « connaissances » (cf. (Schrag 1996; Boufkhad *et al.* 1997; Darwiche & Marquis 2002)).

Considérons à présent les fragments propositionnels basés sur les DAGs (Graphes Dirigés Acycliques). Généralisons quelque peu la définition du langage $\mathcal{NNF}_{PS}$ (définition 1.7). On notera encore $\mathcal{NNF}_{PS}$ le langage obtenu. Une « formule »[15] de $\mathcal{NNF}_{PS}$ est un graphe dirigé acyclique enraciné où chaque feuille est étiquetée par Vrai, Faux, x ou $\neg x$, $x \in PS$; et chaque nœud interne est étiqueté par $\wedge$ ou $\vee$ et peut avoir un nombre arbitraire de fils. Si C est un nœud dans une formule $\mathcal{NNF}_{PS}$, alors $Var(C)$ désigne l'ensemble de toutes les variables qui étiquettent les descendants du nœud C. Par ailleurs, si ϕ est une formule $\mathcal{NNF}_{PS}$ de racine C, alors $Var(\phi)$ est défini comme $Var(C)$. Les fragments intéressants de $\mathcal{NNF}_{PS}$ sont obtenus en imposant certaines des propriétés suivantes (Darwiche 2001) :

– **Décomposabilité** : un nœud « et » C est décomposable si et seulement si les conjoints de C ne partagent pas de variables. C'est-à-dire que si $C_1, \ldots, C_n$ sont les fils du nœud « et » C, alors $Var(C_i) \cap Var(C_j) = \emptyset$ pour tout $i, j \in 1 \ldots n$ avec $i \neq j$. Une formule $\mathcal{NNF}_{PS}$ satisfait la propriété de décomposabilité si et seulement si tout nœud « et » de cette formule est décomposable.
– **Déterminisme** : un nœud « ou » C est déterministe si et seulement si chaque paire de disjoints de C est logiquement contradictoire. C'est-à-dire que si $C_1, \ldots, C_n$ sont les fils du nœud « ou » C, alors $C_i \wedge C_j \models$ Faux pour tout $i, j \in 1 \ldots n$ avec $i \neq j$. Une formule $\mathcal{NNF}_{PS}$ satisfait la propriété de déterminisme si et seulement si tout nœud « ou » de cette formule est déterministe.
– **Décision** : un nœud de décision N dans une formule $\mathcal{NNF}_{PS}$ est un nœud étiqueté par Vrai, Faux, ou un nœud « ou » de la forme $(x \wedge \alpha) \vee (\neg x \wedge \beta)$, où x est une variable, α et β des nœuds de décision. Dans la suite, $dVar(N)$ désigne la variable x. Une formule $\mathcal{NNF}_{PS}$ satisfait la propriété de décision lorsque sa racine est un nœud de décision.
– **Ordonnancement** : soit $<$ un ordre total et strict sur les variables de PS. Une formule $\mathcal{NNF}_{PS}$ satisfaisant la propriété de décision satisfait la propriété d'ordonnancement par rapport à $<$ si et

15. Dans la suite de ce chapitre, nous utiliserons le terme formule pour désigner la représentation sous forme de DAG d'une formule.

seulement si la condition suivante est satisfaite : si N et M sont des nœuds « ou », et si N est un ancêtre du nœud M, alors $dVar(N) < dVar(M)$.

– **Uniformité** : un nœud « ou » C est uniforme si et seulement si chaque disjoint de C mentionne les mêmes variables. C'est-à-dire que si $C_1, \ldots, C_n$ sont les fils du nœud « ou » C, alors $Var(C_i) = Var(C_j)$ pour tout $i, j \in 1 \ldots n$. Une formule $\mathcal{NNF}_{PS}$ satisfait la propriété d'uniformité si et seulement si tout nœud « ou » de cette formule est uniforme.

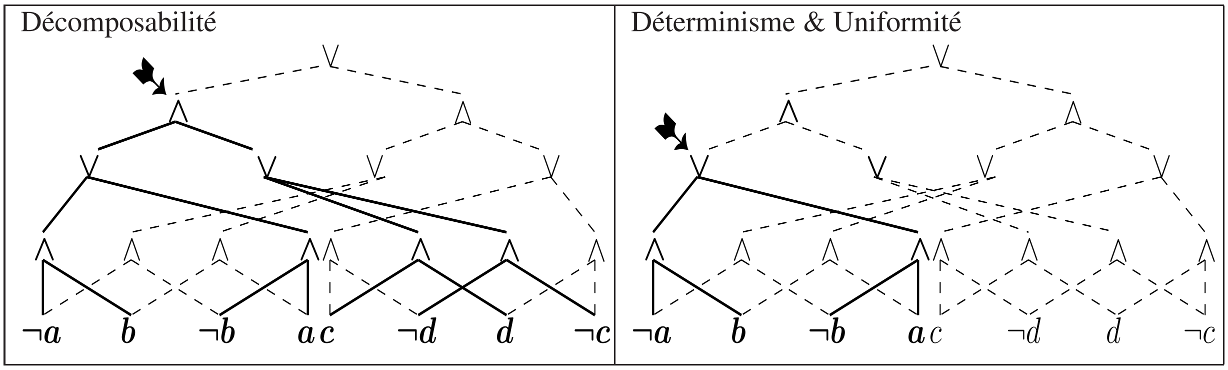


FIG. 5.1 – *Une formule $\mathcal{NNF}_{PS}$. À gauche, le nœud marqué est décomposable, alors que le nœud marqué avec le même symbole sur la figure de droite désigne un nœud déterministe et uniforme.*

Exemple 5.4 (Formule décomposable, déterministe, uniforme)

La figure 5.1 présente une formule en $\mathcal{NNF}_{PS}$.

Considérons le nœud « et » marqué `Décomposable` () sur la partie gauche de la figure 5.1. Le nœud « et » C a deux fils C_1 et C_2 tels que $Var(C_1) = \{a, b\}$ et $Var(C_2) = \{c, d\}$; le nœud C est décomposable car ses deux fils ne partagent pas de variables. Tout autre nœud « et » de la figure 5.1 est aussi décomposable et, ainsi, la formule $\mathcal{NNF}_{PS}$ de la figure est décomposable.

Considérons à présent le nœud « ou » marqué `déterministe`, `uniforme` () sur la partie droite de la figure ; il possède deux fils correspondant aux sous-formules $\neg a \wedge b$ et $\neg b \wedge a$. Ces deux sous-formules sont conjointement contradictoires, donc le nœud « ou » est déterministe. De plus, ses deux fils partagent les mêmes variables a et b, ce nœud « ou » est aussi uniforme. Puisque les autres nœuds « ou » dans la figure 5.1 sont également déterministes et uniformes, la formule $\mathcal{NNF}_{PS}$ de cette figure est déterministe et uniforme. ■

Nous considérons les fragments propositionnels suivants (Darwiche & Marquis 2002) :

Définition 5.2 (Fragments propositionnels)

– *Le langage $\mathcal{DNNF}$ est le sous-ensemble de $\mathcal{NNF}_{PS}$ des formules satisfaisant la propriété de décomposabilité.*
– *Le langage d-$\mathcal{DNNF}$ est le sous-ensemble de $\mathcal{NNF}_{PS}$ des formules satisfaisant la propriété de décomposabilité et la propriété de déterminisme.*
– *Le langage sd-$\mathcal{DNNF}$ est le sous-ensemble de $\mathcal{NNF}_{PS}$ des formules satisfaisant la propriété de décomposabilité, la propriété de déterminisme et la propriété d'uniformité.*
– *Le langage $\mathcal{FBDD}$ est le sous-ensemble de $\mathcal{NNF}_{PS}$ des formules satisfaisant la propriété de décomposabilité et la propriété de décision.*

– *Le langage $\mathcal{OBDD}_<$ est le sous-ensemble de $\mathcal{NNF}_{PS}$ des formules satisfaisant la propriété de décomposabilité, la propriété de décision et la propriété d'ordonnancement. Le langage $\mathcal{OBDD}$ est l'union, pour tous les ordres totaux $<$ possibles des fragments $\mathcal{OBDD}_<$.*
– *Le langage $\mathcal{MODS}$ est le sous-ensemble des $\mathcal{DNF} \cap$ d-$\mathcal{DNNF}$ de formules satisfaisant la propriété d'uniformité.*

Notons que les six langages ci-avant ne sont pas des sous-ensembles de $PROP_{PS}$ au sens strict puisque leurs éléments sont des DAGs enracinés, contrairement aux formules standard « en arbres ». Considérer des représentations basées sur les DAGs est juste un moyen de permettre le partage de sous-formules ; bien que ce soit important du point de vue de l'efficacité spatiale, cela n'a pas d'impact sur la sémantique, les définitions et propriétés données au paragraphe 1.2.2 peuvent facilement être étendues à des « formules » basées sur les DAGs.

Le langage $\mathcal{FBDD}$ correspond aux *diagrammes de décision binaires libres*, bien connus en vérification formelle (Gergov & Meinel 1994), alors que le sous-ensemble obtenu en imposant la propriété d'ordonnancement par rapport à un ordre donné sur les variables contient les *diagrammes de décision binaires ordonnés ($\mathcal{OBDD}$)* (Bryant 1986).

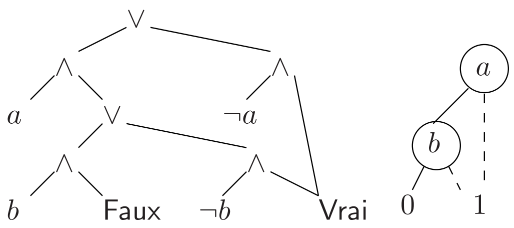
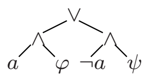
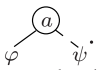


FIG. 5.2 – *Sur la gauche, une formule dans le langage $\mathcal{OBDD}_<$. Sur la droite, une notation plus standard pour celui-ci.*

Les diagrammes de décision binaires sont généralement représentés de façon plus compacte : les étiquettes Vrai et Faux sont désignées par 1 et 0, respectivement ; et chaque nœud de décision 

est désigné par 

. La formule $\mathcal{OBDD}_<$ sur la gauche de la figure 5.2 correspond au diagramme de décision binaire sur la droite de la figure 5.2.

Afin de le rendre encore plus compact, tout $\mathcal{OBDD}_<$ peut être réduit. Les **règles de réduction** suivantes sont suffisantes pour supprimer nœuds et arcs inutiles dans un $\mathcal{OBDD}_<$ (cf. figure 5.3) quand on les applique des feuilles à la racine.

Règle d'élimination si un nœud w est le successeur des fils gauche et droit d'un nœud v, tous les arcs menant au nœud v peuvent être redirigés pour mener au nœud w et le nœud v peut être éliminé.

Règle de fusion si deux nœuds v et w ont le même label, le même fils gauche et le même fils droit, ces deux nœuds peuvent être fusionnés, i.e. tous les arcs menant à w peuvent être redirigés pour mener à v et, ainsi, w peut être éliminé.

Un tel processus de réduction (fonction REDUCE) peut être réalisé en temps polynomial dans la taille du $\mathcal{OBDD}_<$ (Bryant 1986).

Le codage $\mathcal{MODS}$ d'une formule propositionnelle consiste essentiellement en la représentation explicite de l'ensemble de ses modèles (restreints à l'ensemble des variables apparaissant dans la formule).

La figure 5.4 décrit une formule dans $\mathcal{MODS}$.

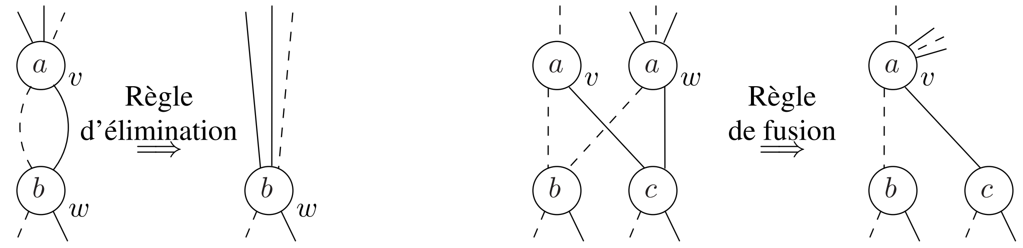


FIG. 5.3 – *Les règles de réduction pour le langage $\mathcal{OBDD}_<$.*

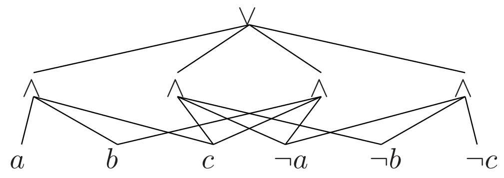


FIG. 5.4 – *Une formule dans le langage $\mathcal{MODS}$.*

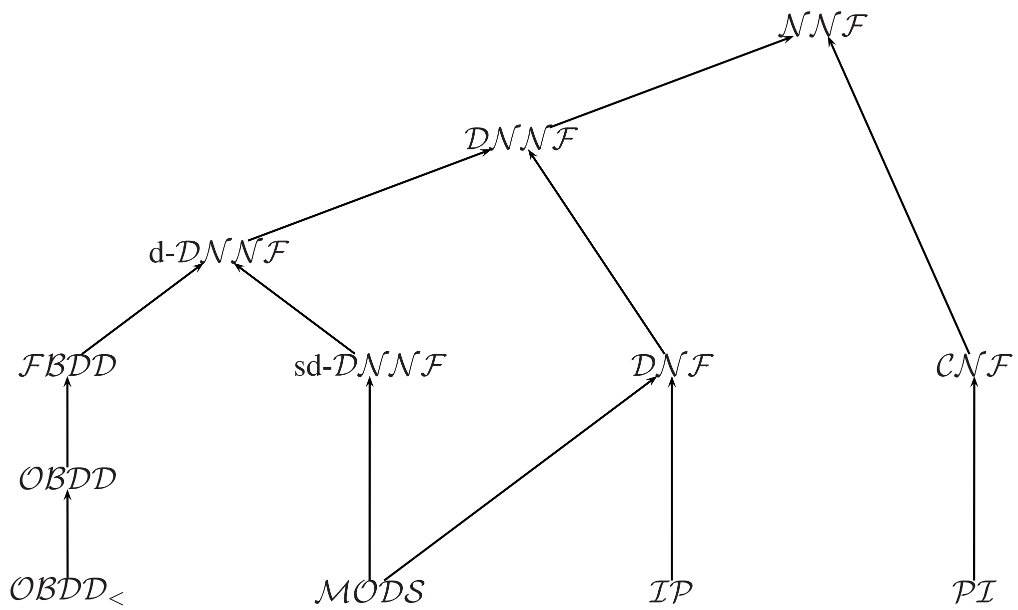


FIG. 5.5 – *Graphe d'inclusion de fragments propositionnels complets (extrait de (Darwiche & Marquis 2002)). Un arc $L_1 \rightarrow L_2$ signifie que L_1 est une sous-ensemble propre de L_2.*

La figure 5.5 décrit le graphe d'inclusion des fragments propositionnels complets considérés dans ce chapitre.

Éliminer une quantification dans une formule $\mathcal{OBDD}_<$ peut être réalisé en un temps quadratique en la taille de l'entrée (une formule $\mathcal{OBDD}_<$ équivalente à $\exists x.\Sigma$ (resp. $\forall x.\Sigma$) est calculée comme $\Sigma_{x \leftarrow 0} \vee \Sigma_{x \leftarrow 1}$, (resp. $\Sigma_{x \leftarrow 0} \wedge \Sigma_{x \leftarrow 1}$), cf. e.g. (Bryant 1986)). Cependant, puisque la taille de la formule résultante peut être quadratique en la taille de l'entrée Σ, nous n'avons aucune garantie qu'une telle élimination conduise à une formule de taille polynomiale en la taille de l'entrée lorsque le processus est itéré afin d'éliminer plus d'un nombre constant de variables. Ainsi, nous ne pouvons garantir que le temps nécessaire à un tel algorithme d'élimination soit polynomial en la taille de l'entrée lorsque celle-ci inclut l'ensemble des variables à éliminer (donc sans borner a priori son cardinal). Effectivement, la proposition suivante montre que, quelle que soit l'approche pour résoudre QBF pour les formules $\mathcal{OBDD}_<$, l'existence d'un algorithme en temps polynomial est peu vraisemblable :

Proposition 5.1
QDNNF, Qd-DNNF, QFBDD et QOBDD$_<$ sont **PSPACE**-complets.

Preuve 5.1
L'appartenance vient directement du fait que QCNF est dans **PSPACE**, le fait que le langage des circuits associé à $PROP_{PS}$ inclut $\mathcal{NNF}_{PS}$ comme un sous-ensemble propre, le fait que tout circuit (codant une fonction booléenne sur $\{x_1, \ldots, x_n\}$) peut être modélisé en temps polynomial en une formule $\mathcal{CNF}$ sur un ensemble étendu de variables équivalente au circuit une fois les nouvelles variables oubliées (i.e., quantifiées existentiellement), et le fait que **PSPACE** soit fermé par réduction polynomiale.

Pour la difficulté, puisque les inclusions suivantes sont vérifiées (cf. figure 5.5)

$$\mathcal{OBDD}_< \subset \mathcal{FBDD} \subset \text{d-}\mathcal{DNNF} \subset \mathcal{DNNF}$$

il suffit de prouver que le problème QBF pour les formules $\mathcal{OBDD}_<$ est **PSPACE**-difficile. La preuve est par réduction du problème QBF pour les formules $\mathcal{DNF}$. La principale étape est de montrer que toute formule $\mathcal{DNF}$ $\phi = \gamma_1 \vee \ldots \vee \gamma_n$ peut être associée en temps polynomial à une $\mathcal{QBF}$ équivalente de la forme $\exists X.\psi$ où $X \cap Var(\phi) = \emptyset$ et ψ est une formule $\mathcal{OBDD}_<$ (pour n'importe quel ordre sur $Var(\phi)$). Avant tout, notons $obdd(\gamma_i)$ la formule $\mathcal{OBDD}_<$ équivalente au terme γ_i $(i \in 1 \ldots n)$; clairement, tout terme γ_i peut être associé en temps polynomial en $|\gamma_i|$ à une telle formule $obdd(\gamma_i)$. Soient $Var(\phi) = \{y_1, \ldots, y_m\}$ et $X = \{x_1, \ldots, x_{n-1}\}$ un ensemble de nouvelles variables ; soit $\psi = \psi^1$, où les formules ψ^i $(i \in 1 \ldots n)$ sont définies par :

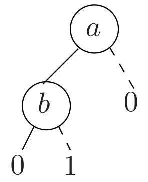
- $\psi^n = obdd(\gamma_n)$, et
- $\psi^i = (obdd(\gamma_i) \wedge x_i) \vee (\psi^{i+1} \wedge \neg x_i)$, pour $i = 1, \ldots, n-1$.

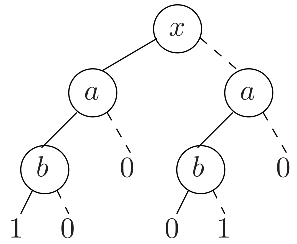
De ces définitions, ψ – qui peut être lue comme une formule $\mathcal{OBDD}_<$ où le nouvel ordre $<$ est l'extension de l'ordre précédent $y_1 < \ldots < y_m$ tel que $x_1 < \ldots < x_{n-1} < y_1 < \ldots < y_m$ – peut être construite en temps polynomial en la taille de ϕ. À présent, puisque pour tout couple de $\mathcal{QBF}$ α, β et toute variable x, nous avons $\exists x.(\alpha \vee \beta) \equiv (\exists x.\alpha) \vee (\exists x.\beta)$ (point 6. de la proposition 1.1), et $\exists x.(\alpha \wedge x) \equiv \exists x.(\alpha \wedge \neg x) \equiv \alpha$ lorsque $x \notin Var(\alpha)$ (une conséquence immédiate du point 10. de la proposition 1.1), on conclut immédiatement que $\phi \equiv \exists Y.\psi$. Enfin, le méta-théorème de substitution pour les $\mathcal{QBF}$ montre que pour tout préfixe P, la $\mathcal{QBF}$ $P.\phi$ est équivalente à la $\mathcal{QBF}$ $P\ \exists Y.\psi$. □

L'exemple suivant illustre la preuve de difficulté :

Exemple 5.5 (Preuve 5.1)
Considérons la formule suivante $\phi = (\gamma_1 \vee \gamma_2)$ où $\gamma_1 = (a \wedge b)$ et $\gamma_2 = (a \wedge \neg b)$.

Nous avons

$\psi^2 \equiv$ 

Alors $\psi \equiv \psi^1 \equiv$ 

et $\phi \equiv \exists x.\psi$. ■

À partir de la réduction fournie dans la preuve de la proposition 5.1, nous pouvons également montrer que le problème QBF pour les formules $\mathcal{OBDD}_<$ atteint toutes les classes Σ_i^p et Π_i^p de la hiérarchie polynomiale quand des restrictions sont imposées sur le préfixe de l'entrée : si aucune alternance de quantificateurs n'apparaît, le problème QBF équivaut au problème de satisfiabilité ou au problème de validité, et les deux sont dans **P** pour les formules $\mathcal{OBDD}_<$; si le préfixe est de la forme $\forall S_1\ \exists S_2$, le problème est Π_1^p-complet (= **co-NP**-complet) ; si le préfixe est de la forme $\exists S_1\ \forall S_2\ \exists S_3$, le problème est Σ_2^p-complet, et ainsi de suite. Puisque la négation d'une formule $\mathcal{OBDD}_<$ peut être transformée en une formule $\mathcal{OBDD}_<$ en temps constant, nous obtenons également que si le préfixe est de la forme $\exists S_1\ \forall S_2$, le problème est Σ_1^p-complet (= **NP**-complet), si le préfixe est de la forme $\forall S_1\ \exists S_2\ \forall S_3$, le problème est Π_2^p-complet, et ainsi de suite.

Ajouter la propriété d'uniformité à la restriction de QBF aux formules d-$\mathcal{DNNF}$ ne rend pas le problème QBF associé plus facile ; en effet, toute formule d-$\mathcal{DNNF}$ peut être uniformisée en temps polynomial.

Corollaire 5.1
Qsd-DNNF est **PSPACE**-complet.

Preuve 5.2
Immédiate en partant du fait que le problème QBF pour d-$\mathcal{DNNF}$ est **PSPACE**-difficile et qu'une formule d-$\mathcal{DNNF}$ peut être transformée en une formule sd-$\mathcal{DNNF}$ équivalente en temps polynomial (voir le lemme A.1 dans (Darwiche & Marquis 2002)). □

Exemple 5.6 (Preuve 5.2)
La figure 5.6 montre un exemple de transformation d'une formule $\mathcal{DNNF}$ en une formule sd-$\mathcal{DNNF}$ équivalente. Dans (Darwiche & Marquis 2002), les auteurs montrent qu'une telle transformation peut s'effectuer en temps polynomial sur l'ensemble des nœuds « ou » de la formule.

■

À l'opposé, il est intéressant de noter que la restriction de QBF aux formules $\mathcal{OBDD}_<$ est traitable pour le sous-ensemble des instances dont les préfixes sont compatibles avec l'ordre total, strict $<$ associé au fragment $\mathcal{OBDD}_<$:

Proposition 5.2
Soit $\Sigma = QS_1 \ldots QS_n.\phi$ une $\mathcal{QBF}$ prénexe, polie, fermée où chaque Q est un quantificateur et $\{S_1, \ldots, S_n\}$

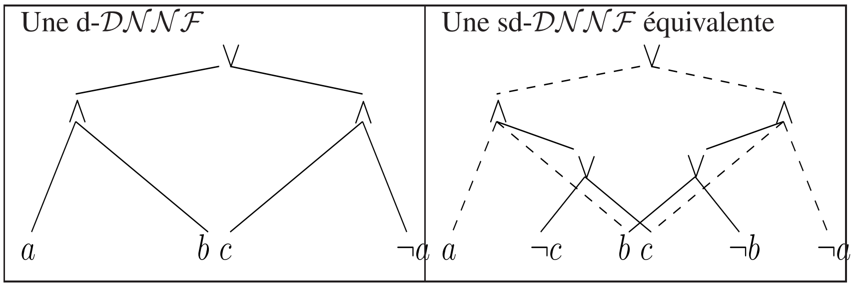


FIG. 5.6 – *À gauche, une formule $\mathcal{DNNF}$. À droite, une formule uniformisée (sd-$\mathcal{DNNF}$) équivalente.*

est une partition de $Var(\phi)$ qui ne contient pas l'ensemble vide. Le préfixe $QS_1 \dots QS_n$ de Σ est dit *compatible* avec un ordre total, strict $<$ sur $Var(\phi)$ si et seulement si pour chaque $x, y \in Var(\phi)$ tel que $x < y$, si $x \in S_i$ et $y \in S_j$, alors $j \geq i$. Le problème QOBDD$_<$ restreint aux formules dont le préfixe est compatible avec $<$ est dans **P**.

Preuve 5.3
La preuve est constructive et repose sur l'algorithme en temps polynomial SOBDD décrit ci-après. Un tel algorithme consiste en l'élimination des quantifications, de la plus interne à la plus externe, dans la formule d'entrée.

Soit x la plus grande variable de $Var(\phi)$ pour l'ordre $<$ associé au fragment $\mathcal{OBDD}_<$.

Supposons que x soit quantifiée existentiellement. Par construction, une interprétation I est un modèle (resp. un contre-modèle) d'une formule $\mathcal{OBDD}_<$ ϕ si et seulement si l'unique chemin de la racine de ϕ qui est compatible avec I finit par le puits 1 (resp. 0) ; un tel chemin correspond à un impliquant de ϕ (resp. la négation d'un tel impliquant).

Soit N un nœud de décision étiqueté par x dans ϕ. Étant donnée l'hypothèse d'ordre pour x, les seuls fils possibles de N dans ϕ sont les feuilles (l'une d'entre elles est la feuille 1 et l'autre est la feuille 0 si le diagramme de décision binaire ordonné ϕ est réduit, ce qui peut être supposé sans perte de généralité puisqu'une telle réduction peut être réalisée en temps polynomial).

Puisque tout modèle de $\exists x.\phi$ coïncide avec un modèle de ϕ, excepté éventuellement sur x, il suffit de supprimer chaque nœud N étiqueté par x et de rediriger les arcs entrants en N vers la feuille 1 afin d'obtenir une formule $\mathcal{OBDD}_<$ équivalente à $\exists x.\phi$; la formule résultante n'est pas nécessairement réduite, mais peut l'être en temps polynomial si ce n'est pas le cas. L'ensemble du processus peut être aisément réalisé en temps polynomial en la taille de ϕ.

Enfin, si la plus grande variable x de $Var(\phi)$ pour $<$ est universellement quantifiée dans Σ, nous tirons avantage de l'équivalence $\forall x.\phi \equiv \neg\exists x.\neg\phi$ (point 3. de la proposition 1.1) et du fait qu'une formule $\mathcal{OBDD}_<$ équivalente à la négation d'une formule $\mathcal{OBDD}_<$ peut être obtenue en temps constant (il suffit d'échanger les étiquettes des deux puits 0 et 1). $\square$

L'algorithme 5.2 illustre la méthode décrite ci-avant.La procédure REDUCE est décrite par R. E. Bryant dans (Bryant 1986) et une version plus efficace, linéaire en temps, est donnée par D. Sieling et I. Wegener dans (Sieling & Wegener 1993). De ce fait, SOBDD a une complexité en temps en $\mathcal{O}(n \times |\phi|)$.

Exemple 5.7 (Exécution de SOBDD**)**
Soit $<$ l'ordre sur $\{a, b, c\}$ tel que $a < b < c$. Considérons une $\mathcal{QBF}$ Σ telle que

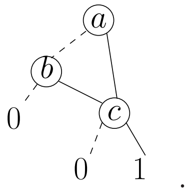
$$\Sigma = \exists a\, \forall b\, \exists c.\phi$$

$$\text{et } \phi = (a \vee b) \wedge c$$

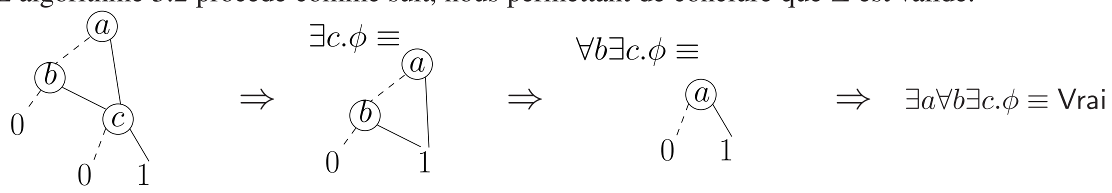
Algorithme 5.2 : Algorithme en temps polynomial pour la validité des $n\mathcal{QBF}$ à matrice $\mathcal{OBDD}_<$ avec préfixe compatible

procédure SOBDD
Données : Une $\mathcal{QBF}$ $Q_1x_1 \ldots Q_nx_n.\phi$ où $\phi \in \mathcal{OBDD}_<$ est réduit et
$Q_1x_1 \ldots Q_nx_n$ est compatible avec $<$
Résultat : Vrai si $Q_1x_1 \ldots Q_nx_n.\phi$ est valide, Faux sinon
début
    pour *i* **de** *n* **à** *1* **faire**
        si $Q_i = \exists$ **alors**
            pour chaque *nœud* N *étiqueté par* x_i **faire**
                $N \leftarrow 1$;
        sinon
            pour chaque *nœud* N *étiqueté par* x_i **faire**
                $N \leftarrow 0$;
        REDUCE(ϕ) ;
    retourner ϕ ;
fin

La formule $\mathcal{OBDD}_<$ associée à ϕ est :



L'algorithme 5.2 procède comme suit, nous permettant de conclure que Σ est valide.



■

La restriction de QBF au fragment $\mathcal{MODS}$ est également traitable :

Proposition 5.3
QMODS est dans **P**.

Preuve 5.4
À nouveau, la preuve est constructive et repose sur l'existence de l'algorithme SMODS décrit ci-après et en temps polynomial. Cet algorithme procède à l'élimination des quantifications, qui opère comme une loi interne dans le fragment QMODS. Dans cet algorithme, toute formule $\mathcal{MODS}$ est considérée, sans perte de généralité, comme un ensemble de termes et tout terme comme un ensemble de littéraux. La formule $\mathcal{MODS}$ résultante est soit l'ensemble vide (i.e., la disjonction vide équivalente à Faux ($\{\} \equiv$ Faux)), soit un ensemble contenant la conjonction vide équivalente à Vrai ($\{\{\}\} \equiv$ Vrai).

Expliquons comment les quantifications peuvent être éliminées et considérons d'abord le cas des quantificateurs existentiels ; nous exploitons les deux équivalences : (1) pour tout couple de $\mathcal{QBF}$ α et β et toute variable $x \in PS$, nous avons $\exists x.(\alpha \vee \beta) \equiv (\exists x.\alpha) \vee (\exists x.\beta)$ (point 6. de la proposition 1.1), et (2) $\exists x.\gamma$

est équivalent au terme $\gamma \setminus \{x, \neg x\}$ obtenu en retirant x et $\neg x$ de γ lorsque γ est un terme satisfiable (vu comme l'ensemble de ses littéraux) (ce qui est une conséquence directe de la définition du conditionnement et du point 2. de la proposition 1.1). Clairement, si γ est un terme canonique (codant un modèle) sur $X \cup \{x\}$, alors $\gamma \setminus \{x, \neg x\}$ est un terme canonique sur X. Ainsi, retirer toute occurrence de x et $\neg x$ dans une formule $\mathcal{MODS}$ ϕ conduit à une formule $\mathcal{MODS}$ équivalente à $\exists x.\phi$.

Focalisons-nous à présent sur le cas des quantificateurs universels. Soit ϕ une formule $\mathcal{MODS}$, vue comme un ensemble de termes canoniques sur $X \cup \{x\}$, vus comme des ensembles de littéraux. Quel que soit $x \in PS$, les éléments γ de ϕ peuvent être partitionnés en deux ensembles (interprétés de façon disjonctive) : S est l'ensemble de tous les termes γ de ϕ tel que $switch(\gamma, x)$ appartient également à ϕ, où $switch(\gamma, x)$ est le terme canonique sur $X \cup \{x\}$ qui coïncide avec γ pour toute variable de X mais qui contient $\neg x$ lorsque γ contient le littéral x, tandis que $switch(\gamma, x)$ contient x quand γ contient $\neg x$. Il est évident que γ appartient à S si et seulement si $switch(\gamma, x)$ appartient à S. Enfin, S' est le complément de S dans ϕ. Par construction, nous avons $\forall x.\phi \equiv \forall x.(S \vee S')$; à présent, observons que $S = \{\gamma_1, \ldots, \gamma_k\}$ (interprété disjonctivement) est indépendant de x (Lang *et al.* 2003) dans le sens où il est équivalent à la disjonction $\psi = (\gamma_1 \setminus \{x, \neg x\}) \vee \ldots \vee (\gamma_k \setminus \{x, \neg x\})$ qui est une formule $\mathcal{MODS}$ sur X (pour tout γ_i ($i \in 1 \ldots k$), nous avons $\gamma_i \vee switch(\gamma_i, x) \equiv \gamma_i \setminus \{x, \neg x\}$). Clairement, ψ peut être calculé en temps polynomial en la taille de ϕ. Puisque ψ est indépendant de x, nous avons $\forall x.\phi \equiv \psi \vee (\forall x.S')$ (conséquence du point 9. de la proposition 1.1). Enfin, il suffit de montrer que $\forall x.S'$ est contradictoire (et donc que $\forall x.\phi \equiv \psi$). Soit γ un terme canonique vu comme une interprétation sur X ; par définition, γ satisfait $\forall x.S'$ si et seulement si l'extension de γ obtenue en affectant x à 0 et l'extension de γ obtenue en affectant x à 1 sont des modèles de S' ; mais c'est impossible selon la définition de S' (si l'une de ces extensions appartient à S', la seconde ne peut pas appartenir à S' également puisqu'elle est obtenue en inversant x dans la première). Par conséquent, $\forall x.S'$ est contradictoire, donc $\forall x.\phi \equiv \psi$. □

L'élimination de quantifications pour le fragment QMODS est donnée dans l'algorithme 5.3.

Algorithme 5.3 : Algorithme en temps polynomial pour la validité des formules $n\mathcal{QBF}$ à matrice $\mathcal{MODS}$

procédure SMODS
Données : Une $n\mathcal{QBF}$ $Q_1x_1 \ldots Q_nx_n.\phi$ où $\phi \in \mathcal{MODS}$
Résultat : Vrai si $Q_1x_1 \ldots Q_nx_n.\phi$ est valide, Faux sinon
début
  pour *i* **de** *n* **à** *1* **faire**
    si $x_i \in \forall$ **alors**
      $\phi' \leftarrow \{\gamma \in \phi \mid switch(\gamma, x_i) \in \phi\}$;
    sinon
      $\phi' \leftarrow \phi$;
    $\phi \leftarrow \{\gamma \setminus \{x_i, \neg x_i\} \mid \gamma \in \phi'\}$;
  si $\phi = \{\}$ **alors**
    retourner Faux ;
  sinon
    retourner Vrai ;
fin

Cet algorithme SMODS s'exécute en temps $\mathcal{O}(n \times |\phi|^2)$. Notons que cet algorithme pourrait être facilement amélioré, en retournant Faux dès que la clause vide est obtenue mais notre objectif ici est plus de présenter des algorithmes dont le fonctionnement est facile à comprendre que des algorithmes optimisés.

Illustrons la manière dont SMODS fonctionne sur deux exemples simples.

Exemple 5.8 (Exécution de SMODS)
Considérons la formule suivante $\phi = ((a \wedge b) \vee (a \wedge \neg b))$.

1. $\forall b \exists a.\{\{a, b\}, \{a, \neg b\}\} \rightarrow \forall b.\{\{b\}, \{\neg b\}\} \rightarrow \{\{\}\} \equiv \mathsf{Vrai}$
2. $\exists b \forall a.\{\{a, b\}, \{a, \neg b\}\} \rightarrow \exists b.\{\} \rightarrow \{\} \equiv \mathsf{Faux}$

■

Intéressons-nous à présent à deux fragments propositionnels importants en IA : les impliqués premiers et le fragment dual des impliquants premiers (cf. e.g., (Marquis 2000) pour une vue d'ensemble de leurs applications en abduction, en raisonnement hypothétique, en raisonnement sous hypothèse de monde clos et dans d'autres domaines d'IA).

Les définitions originales des notions d'impliqués/impliquants premiers sont dues à W.V.O. Quine (Quine 1955).

Définition 5.3 (Impliqué, Impliqué premier)
Soient $V \subseteq PS$ et Σ une formule de $PROP_V$.

- *Une clause π de $PROP_V$ (à l'équivalence logique près) est un impliqué de Σ si et seulement si $\Sigma \models \pi$.*
- *Une clause π de $PROP_V$ (à l'équivalence logique près) est un impliqué premier de Σ si et seulement si :*
  - *π est un impliqué de Σ, et*
  - *pour chaque impliqué π' de Σ, si $\pi' \models \pi$, alors $\pi \models \pi'$.*

Exemple 5.9 (Impliqué premier)
Soit $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. L'ensemble des impliqués premiers de Σ est

$$\{(\neg a \vee b), (\neg b \vee c), (\neg a \vee c)\}.$$

■

Les impliqués premiers caractérisent les clauses qui sont les conséquences logiques les plus fortes de la formule considérée. Toute clause fondamentale, conséquence logique de Σ, est donc sous-sommée par un impliqué premier.

Définition 5.4 (Formules d'impliqués premiers)
Une **formule d'impliqués premiers** *(ou formule de Blake) de $PROP_{PS}$ est une formule $\mathcal{CNF}$ Σ où chaque impliqué premier de Σ (à l'équivalence logique près) apparaît comme un conjoint. $\mathcal{PI}$ est le langage de toutes les formules d'impliqués premiers (un sous-ensemble propre de $\mathcal{CNF}$).*

Exemple 5.10 (Formules d'impliqués premiers)
La formule suivante est une formule d'impliqués premiers.

$$(a \vee b) \wedge (\neg b \vee c \vee d) \wedge (a \vee c \vee d).$$

■

L'ensemble des formules d'impliqués premiers est une classe polynomiale pour SAT car (1) une formule $\mathcal{CNF}$ Σ est d'impliqués premiers si et seulement si aucune de ses clauses n'est impliquée par une autre clause de Σ et toute résolvante entre deux clauses de Σ est impliquée par une clause de Σ (cela montre que le problème de décider si une formule propositionnelle est d'impliqués premiers peut être décidé en temps polynomial), et (2) une formule d'impliqués premiers Σ est satisfiable si et seulement si elle ne se réduit pas à la clause vide.

Malheureusement, $\mathcal{PI}$ n'est pas un fragment traitable pour QBF (sous les hypothèses standard de la théorie de la complexité).

Proposition 5.4
QPI est **PSPACE**-complet.

Preuve 5.5
L'appartenance vient directement du fait que QCNF est dans **PSPACE**, et toute formule $\mathcal{PI}$ est également une formule $\mathcal{CNF}$.

Pour la difficulté, nous présentons une réduction polynomiale de QCNF vers QPI. Soit ϕ une formule $\mathcal{CNF}$ sur $\{x_1, \ldots, x_n\}$, vue comme l'ensemble S de ses clauses. Nous tirons avantage de la propriété suivante, qui résulte directement de la correction des algorithmes de calcul d'impliqués premiers fondés sur le principe de résolution (comme celui de Tison (Tison 1967)) : un ensemble S de clauses contient tous ses impliqués premiers si et seulement si lorsque deux clauses de S ont une résolvante δ, il existe une clause $\pi \in S$ telle que $\pi \models \delta$.

Supposons sans perte de généralité que S est totalement ordonné par rapport à un ordre arbitraire (mais fixe) $<$. Soient δ_i et δ_j deux clauses de S avec $i < j$; lorsque δ_i et δ_j ont une résolvante, ajouter une nouvelle variable $y_{i,j}$, remplacer δ_i par $\delta_i \vee y_{i,j}$ et δ_j par $\delta_j \vee \neg y_{i,j}$; faire cela de façon systématique pour tout couple de clauses (δ_i, δ_j) de S revient à générer en temps polynomial une formule $\mathcal{CNF}$ ψ sur une vocabulaire étendu $\{x_1, \ldots, x_n\} \cup Y$ où $\mathcal{O}(n^2)$ nouvelles variables $y_{i,j}$ sont introduites. Par construction, toute résolvante binaire des clauses de ψ est tautologique, et ainsi impliquée par une clause de ψ. De ce fait, ψ contient tous ses impliqués premiers, et une formule d'impliqués premiers θ équivalente à ψ peut être construite en temps polynomial en $|\psi|$ en retirant successivement toutes les clauses de ψ qui sont impliquées par au moins une autre clause de ψ.

À présent, pour tout couple de $\mathcal{QBF}$ α et β et toute variable $x \in PS$, nous avons $\forall x.(\alpha \wedge \beta) \equiv (\forall x.\alpha) \wedge (\forall x.\beta)$ (point 5. de la proposition 1.1) ; de plus, pour toute clause non tautologique δ (vue comme l'ensemble de ses littéraux) et toute variable $x \in PS$, $\forall x.\delta$ est équivalent à la clause $\delta \setminus \{x, \neg x\}$ (c'est une conséquence directe de la définition du conditionnement et du point 1. de la proposition 1.1). Par conséquent, nous avons $\phi \equiv \forall Y.\theta$. Enfin, le méta-théorème de substitution pour les $\mathcal{QBF}$ montre que $QS_1 \ldots QS_k.\phi$ est équivalent à $QS_1 \ldots QS_k \forall Y.\theta$. □

L'exemple suivant illustre la réduction donnée dans la preuve précédente :

Exemple 5.11 (Preuve 5.5)
Soit $\phi = (a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg c \vee d)$ une formule $\mathcal{CNF}$. Nous associons en temps polynomial ϕ à la formule $\mathcal{PI}$ suivante $\theta = (a \vee b \vee y_{1,2}) \wedge (\neg b \vee c \vee d \vee \neg y_{1,2} \vee y_{2,3}) \wedge (\neg c \vee d \vee \neg y_{2,3})$. Nous avons $\phi \equiv \forall y_{1,2}, y_{2,3}.\theta$. ■

En se basant sur la réduction donnée dans la preuve précédente, nous pouvons également montrer que QBF pour les formules $\mathcal{PI}$ atteint tous les niveaux de la hiérarchie polynomiale quand des restrictions sont posées sur le préfixe de l'entrée : si aucune alternance de quantificateurs n'apparaît, le problème est dans **P**, si le préfixe est de la forme $\exists S_1 \; \forall S_2$, le problème est **NP**-complet, si le préfixe est de la forme $\forall S_1 \; \exists S_2 \; \forall S_3$, le problème est Π_2^p-complet, et ainsi de suite.

À présent, qu'en est-il des formules $\mathcal{QBF}$ $\mathcal{PI}$ lorsque le quantificateur le plus interne du préfixe est existentiel ? Contrairement aux formules $\mathcal{OBDD}_<$ (dans le cas général), la négation d'une formule $\mathcal{PI}$ ne peut être calculée en temps polynomial (ni même en *espace polynomial*) en une formule $\mathcal{PI}$, donc l'argument utilisé pour la classe $\mathcal{OBDD}_<$ne peut être réutilisé ici. Néanmoins, il est aisé de montrer qu'ajouter des quantifications existentielles au niveau le plus interne ne conduit pas à changer de niveau dans la hiérarchie polynomiale :

Proposition 5.5
La réduction de QPI aux formules avec des préfixes de la forme $\forall S_1 \exists S_2$ est dans **P**.

Preuve 5.6
La preuve repose sur le fait qu'il existe un algorithme en temps polynomial pour éliminer les quantifications existentielles dans une formule de QPI et qui opère comme une loi interne au fragment QPI. Plus précisément, soit $PI(\phi)$ l'ensemble des impliqués premiers d'une formule propositionnelle ϕ (nous ne conservons qu'un représentant par classe d'équivalence) ; considérons le lemme suivant (une conséquence directe de la proposition 55 dans (Marquis 2000)) :

Lemme 5.1
Soient ϕ une formule de $PROP_{PS}$ et Y un ensemble de variables de PS. Nous avons $PI(\exists Y.\phi) = \{\pi \in PI(\phi) \mid Var(\pi) \cap Y = \emptyset\}$.

Ce lemme montre comment une formule $\mathcal{PI}$ ψ sur X, qui est équivalente à $\exists Y.\phi$, peut être dérivée en temps polynomial en $|\phi|$ quand $\phi \in \mathcal{PI}$.

À nouveau, nous exploitons les propriétés suivantes : (1) pour tout couple de $\mathcal{QBF}$ α et β et toute variable $x \in PS$, nous avons $\forall x.(\alpha \wedge \beta) \equiv (\forall x.\alpha) \wedge (\forall x.\beta)$ et (2) pour toute clause non tautologique δ (vue comme l'ensemble de ses littéraux) et toute variable $x \in PS$, $\forall x.\delta$ est équivalent à la clause $\delta \setminus \{x, \neg x\}$. Une conséquence directe de ces propriétés est que $\forall X.\psi$ est valide si et seulement si ψ ne contient que des clauses tautologiques, ce qui peut être facilement vérifié en temps polynomial en $|\psi|$. □

L'exemple suivant illustre le lemme donné dans la preuve précédente :

Exemple 5.12 (Lemme 5.1)
$(c \vee d)$ est une formule $\mathcal{PI}$ équivalente à $\exists a.((a \vee b) \wedge (\neg b \vee c \vee d) \wedge (\neg a \vee c \vee d))$. ■

En corollaire à la proposition précédente, nous obtenons que si le préfixe de la formule d'entrée est de la forme $\exists S_1 \, \forall S_2 \, \exists S_3$, QPI est **NP**-complet, si le préfixe est de la forme $\forall S_1 \, \exists S_2 \, \forall S_3 \, \exists S_4$, le problème est Π_2^p-complet, et ainsi de suite.

La classe duale suivante est également intéressante.

Définition 5.5 (Impliquant, Impliquant premier)
Soient $V \subseteq PS$ et Σ une formule de $PROP_V$.

- *Un terme π de $PROP_V$ (à l'équivalence logique près) est un impliquant de Σ si et seulement si $\pi \models \Sigma$.*
- *Un terme π de $PROP_V$ (à l'équivalence logique près) est un impliquant premier de Σ si et seulement si :*
  - *π est un impliquant de Σ, et*
  - *pour chaque impliquant π' de Σ, si $\pi \models \pi'$, alors $\pi' \models \pi$.*

Exemple 5.13 (Impliquant premier)
Soit $\Sigma = (a \Rightarrow b) \wedge (b \Rightarrow c)$. L'ensemble des impliquants premiers de Σ est

$$\{(\neg a \wedge \neg b), (\neg a \wedge c), (b \wedge c)\}.$$

■

Définition 5.6 (Formules d'impliquants premiers)
Une **formule d'impliquants premiers** *de $PROP_{PS}$ est une formule $\mathcal{DNF}$ Σ où chaque impliquant premier de Σ (à l'équivalence logique près) apparaît comme un disjoint. $\mathcal{IP}$ est le langage de toutes les formules d'impliquants premiers (un sous-ensemble propre de $\mathcal{DNF}$).*

Les formules d'impliquants premiers sont duales aux formules d'impliqués premiers dans le sens où tout impliquant premier d'une formule ϕ est (à l'équivalence logique près) la négation d'un impliqué premier de $\neg\phi$ (voir e.g. la proposition 8 dans (Marquis 2000). En exploitant cette dualité, nous obtenons également que :

Proposition 5.6
QIP est **PSPACE**-complet.

Preuve 5.7
C'est une conséquence immédiate de la proposition 5.4 étant donné que **PSPACE** est fermé sous la complémentation et le fait que la négation d'une $\mathcal{QBF}$ prénexe avec une matrice $\mathcal{PI}$ est une $\mathcal{QBF}$ prénexe avec une matrice $\mathcal{IP}$. □

Proposition 5.7
La restriction de QIP aux formules avec des préfixes de la forme $\exists S_1\ \forall S_2$ est dans **P**.

Preuve 5.8
C'est une conséquence directe de la proposition 5.5 (par dualité). □

En exploitant la dualité, il est également aisé de prouver que les classes Σ_i^p et Π_i^p de la hiérarchie polynomiale qui n'ont pas été « atteintes » par les restrictions de QPI sont « atteintes » par les restrictions de QIP : si aucune alternance de quantificateurs n'apparaît, QIP est dans **P**, si le préfixe est de la forme $\forall S_1\ \exists S_2$ ou $\forall S_1\ \exists S_2\ \forall S_3$, le problème est **co-NP**-complet, si le préfixe est de la forme $\exists S_1\ \forall S_2\ \exists S_3$ ou $\exists S_1\ \forall S_2\ \exists S_3\ \forall S_4$, le problème est Σ_2^p-complet, et ainsi de suite.

5.3 Compilabilité

Il est intéressant de noter les liens existants entre le problème de recherche de restrictions traitables de QBF basées sur les matrices et le problème de compilabilité de QBF lorsque les matrices sont fixes et que les préfixes peuvent varier. La compilabilité dans **P** de QBF se formalise comme l'appartenance à la classe de compilabilité **compP** du problème de compilation COMP-QBF associé à QBF où chaque instance est divisée en deux parties – la partie fixe, la matrice, et la partie variable, le préfixe (Liberatore 2001; Cadoli *et al.* 2002a).

Définition 5.7 (COMP-QBF**)**
COMP-QBF *est le langage de paires $\{\langle \Sigma, P\rangle \mid P.\Sigma$ est une formule fermée, polie, prénexe de $QPROP_{PS}\}$.*

Définition 5.8 (compP (Cadoli *et al.* 2002a))
Un langage de paires L appartient à **compP** *si et seulement si il existe une fonction à sortie polynomiale*[16] *f et un langage de paires $L' \in$* **P** *tel que $\langle x, y\rangle \in L$ si et seulement si $\langle f(x), y\rangle \in L'$.*

L'importance en pratique d'un tel problème de compilabilité vient du fait que, au lieu de considérer la matrice fixe, on peut plus généralement considérer le cas où elle peut être construite en temps polynomial à partir d'entrées plus simples. Comme de nombreuses classes cibles $\mathcal{C}$ pour la compilation de « connaissances » permettent le conditionnement, la conjonction bornée et la disjonction bornée en temps polynomial (Darwiche & Marquis 2002), divers problèmes d'inférence peuvent être codés (puis résolus) comme des $\mathcal{QBF}$ dont les matrices $\mathcal{C}$ peuvent être calculées en temps polynomial depuis une formule compilée $\Sigma \in \mathcal{C}$ (la base de « connaissances ») et une requête clausale γ.

La question suivante exprime l'appartenance de COMP-QBF à **compP** : peut-on trouver un fragment propositionnel complet $\mathcal{C}$ pour lequel il existe un polynôme $p(.)$ tel que toute formule propositionnelle α a une formule équivalente $\beta \in \mathcal{C}$ satisfaisant $|\beta| \leq p(|\alpha|)$ et pour lequel il existe un algorithme polynomial pour décider si une instance de QBF munie de la matrice β est valide ? Clairement, $\mathcal{C} = \mathcal{MODS}$ n'est pas une réponse satisfaisante puisque toute formule propositionnelle n'a pas un nombre polynomial de modèles. On montre que l'existence d'un fragment $\mathcal{C}$ répondant à la compilabilité de QBF est peu vraisemblable :

Donnons tout d'abord un bref rappel sur les classes de complexité non uniformes :

Définition 5.9 (Machine de Turing à avis)
Une **machine de Turing à avis** *est une machine de Turing à laquelle on associe un « oracle à avis » A, qui peut être n'importe quelle fonction (pas nécessairement une fonction récursive). Pour un mot d'entrée m, un ruban supplémentaire est chargé avec $A(|m|)$ et à partir de là, le calcul procède normalement, en se basant sur les deux entrées, m et $A(|m|)$.*

Définition 5.10 (Avis polynomial)
Une machine de Turing à avis est à **avis polynomial** *si son oracle à avis A satisfait $|A(n)| \leq p(n)$ pour un polynôme p et tout entier n non négatif.*

Définition 5.11 ($\mathcal{C}$/poly)
Si $\mathcal{C}$ est une classe de langages définie en terme de machines de Turing à ressources limitées, alors **$\mathcal{C}$/poly** *est la classe de langages définie par les machines de Turing avec les mêmes limites de ressources mais augmentées par un avis polynomial.*

Proposition 5.8
COMP-QBF n'est pas dans **compP**, à moins que **NP** $\subseteq$ **P/poly**. Le résultat reste vrai lorsqu'une seule alternance de quantificateurs apparaît dans le préfixe des formules considérées[17].

Preuve 5.9
Nous montrons que le problème **NP**-complet 3CNF-SAT peut être résolu en temps (déterministe) polynomial en utilisant une machine de Turing à avis polynomial si COMP-QBF appartient à **compP**. Pour chaque entier n, soit ϕ_n la formule $\mathcal{CNF}$ (vue comme un ensemble de clauses) contenant toutes les clauses de la forme $y_i \vee \delta_i$ où δ_i est une clause comprenant au plus 3 littéraux construite à partir de l'ensemble X_n de variables $x_1, \ldots, x_n$ et chaque y_i est un nouveau symbole (un pour chaque clause δ_i). Clairement, ϕ_n ne

16. f est à sortie polynomiale si et seulement s'il existe un polynôme $p(.)$ tel que $\forall x, |f(x)| \leq p(|x|)$.

17. Contrairement à la restriction de COMP-QBF où aucune alternance n'apparaît dans le préfixe qui est manifestement dans **compP** car il n'existe que deux préfixes possibles pour chaque matrice dans ce cas (l'un n'est composé que de quantifications existentielles et l'autre que de quantifications universelles).

dépend que de n. De plus, le nombre p_n de clauses dans ϕ_n est tel que $p_n \in \mathcal{O}(n^3)$, ainsi la taille de ϕ_n est polynomiale en n.

Soit ϕ_n une formule $\mathcal{CNF}$ quelconque sur X_n. Par construction, chaque clause δ_i de ϕ_n correspond à une clause unique $y_i \vee \delta_i$ de ϕ_n. Soit ψ_n le complément de $\{y_i \vee \delta_i \mid \delta_i \in \phi_n\}$ dans ϕ_n. Il est aisé de construire en temps polynomial en la taille de ϕ_n l'ensemble Y_{ϕ_n} de variables y_i tel que les clauses correspondantes δ_i appartiennent à ϕ_n. Nous savons aussi que (1) pour tout couple de $\mathcal{QBF}$ α et β et toute variable $y \in PS$, nous avons $\forall y.(\alpha \wedge \beta) \equiv (\forall y.\alpha) \wedge (\forall y.\beta)$ (point 5. de la proposition 1.1) ; (2) pour toute clause δ (vue comme l'ensemble de ses littéraux) et toute variable $y \in PS$, si δ ne contient pas à la fois y et $\neg y$, alors $\forall y.\delta$ est équivalent à la clause $\delta \setminus \{y, \neg y\}$ (comme une conséquence directe de la définition du conditionnement et le point 1. de la proposition 1.1) ; et (3) si la variable y n'apparaît pas dans la formule α, nous avons $\forall y.\alpha \equiv \alpha$ (une conséquence facile du point 4. de la proposition 1.1). De ces trois propriétés, nous obtenons directement que $\forall Y_{\phi_n}.\phi_n \equiv \phi_n \wedge \psi_n$. Puisque toute clause $y_i \vee \delta_i$ de ψ_n contient le littéral y_i qui est pur dans $\phi_n \wedge \psi_n$ (i.e., son littéral complémentaire n'apparaît pas dans cette formule), alors $\phi_n \wedge \psi_n$ est satisfiable si et seulement si ϕ_n est satisfiable. Ainsi, ϕ_n est satisfiable si et seulement si $\forall Y_{\phi_n}.\phi_n$ est satisfiable si et seulement si $\exists X_n \forall Y_{\phi_n}.\phi_n$ est valide. Supposons enfin qu'il existe un fragment complet $\mathcal{C}$ pour lequel la restriction de QBF aux formules dont la matrice est dans $\mathcal{C}$ est traitable et pour lequel il existe un polynôme $p(.)$ tel que toute formule propositionnelle α a un équivalent $\beta \in \mathcal{C}$ satisfaisant $\beta \leq p(|\alpha|)$. Alors 3-CNF-SAT peut être résolu en temps polynomial par une machine de Turing déterministe avec avis polynomial A comme suit : pour chaque entrée ϕ_n, $A(n)$ est la formule $\phi_{n,\mathcal{C}}$ de $\mathcal{C}$ qui est équivalente à ϕ_n ; ensuite, la satisfiabilité de ϕ_n est décidée en temps polynomial comme la validité de $\exists X_n \, \forall Y_{\phi_n}.\phi_{n,\mathcal{C}}$. □

Notons que **NP** $\subseteq$ **P/poly** impliquerait que la hiérarchie polynomiale s'effondre au second niveau (cf. chapitre 2 et (Karp & Lipton 1980)), ce qui est considéré comme peu vraisemblable.

5.4 Résumé

Dans ce chapitre, nous avons présenté de nouveaux résultats de traitabilité et d'intraitabilité pour les problèmes de validité de $\mathcal{QBF}$ dont les matrices appartiennent à une classe cible pour la compilation de « connaissances ». Nous pouvons ainsi compléter le panorama de complexité pour QBF comme suit (cf. tableau 5.1).

Fragment	Complexité
$PROP_{PS}$ (cas général)	**PSPACE**-c
$\mathcal{CNF}$	**PSPACE**-c
$\mathcal{DNF}$	**PSPACE**-c
d-$\mathcal{DNNF}$	**PSPACE**-c
sd-$\mathcal{DNNF}$	**PSPACE**-c
$\mathcal{DNNF}$	**PSPACE**-c
$\mathcal{FBDD}$	**PSPACE**-c
$\mathcal{OBDD}_<$	**PSPACE**-c
$\mathcal{OBDD}_<$ (préfixe compatible)	$\in$ **P**
$\mathcal{PI}$	**PSPACE**-c
$\mathcal{IP}$	**PSPACE**-c
$\mathcal{MODS}$	$\in$ **P**

TAB. 5.1 – *Résultats de complexité pour* QBF.

Dans (Darwiche & Marquis 2002), Darwiche et Marquis ont étudié l'efficacité spatiale de nombreux fragments propositionnels complets, dont ceux considérés dans ce papier. Un fragment donné $\mathcal{C}_1$ est considéré au moins aussi concis qu'un second fragment $\mathcal{C}_2$ lorsqu'il existe un polynôme $p(.)$ tel que pour toute formule $\alpha \in \mathcal{C}_2$, il existe une formule équivalente $\beta \in \mathcal{C}_1$ telle que $|\beta| \leq p(|\alpha|)$. Nos résultats montrent que QBF est un problème difficile, même limité aux instances dont les matrices appartiennent aux fragments qui ne sont pas efficaces du point de vue spatial (i.e., le fragment $\mathcal{OBDD}_<$, le fragment $\mathcal{PI}$ et le fragment $\mathcal{IP}$). La traitabilité n'est atteinte sans restriction que pour le fragment $\mathcal{MODS}$ qui fait partie des moins efficaces du point de vue spatial. Sous l'hypothèse de compatibilité pour l'ordre, la traitabilité est également assurée pour le fragment plus concis des $\mathcal{OBDD}_<$; ce fragment apparaît comme le meilleur candidat parmi les classes considérées dans cet article pour lesquelles la traitabilité est garantie ; malheureusement, le choix de l'ordre $<$ a un impact majeur sur la taille des formules $\mathcal{OBDD}_<$ (Bryant 1986).

6

Résolution pratique de QBF

Sommaire

« La théorie, c'est quand on sait tout et que rien ne fonctionne. La pratique, c'est quand tout fonctionne et que personne ne sait pourquoi. Ici, nous avons réuni théorie et pratique : rien ne fonctionne ... Et personne ne sait pourquoi ! »
Albert Einstein[18]

6.1 Introduction

Comme nous l'avons expliqué à plusieurs reprises dans les parties précédentes, QBF est un problème calculatoirement difficile, aussi bien en théorie qu'en pratique. Les méthodes de résolution de QBF fondées sur la notion de restriction comptent parmi celles permettant d'augmenter l'ensemble d'instances traitables en pratique. L'idée clé est de reconnaître les instances pour lesquelles des algorithmes spécifiques peuvent fournir une solution beaucoup plus efficacement que les prouveurs QBF généraux. Comme stipulé dans la partie II, plusieurs restrictions traitables de QBF ont déjà été identifiées : $\mathcal{QKF}$, $\mathcal{QHF}$, reverse-$\mathcal{QHF}$, ren$\mathcal{QHF}$.

18. Einstein (Albert), physicien allemand, né à Ulm (1879–1955). Bien qu'ayant quitté délibérément l'école avant terme et échoué au concours d'entrée de l'Ecole Polytechnique, Albert Einstein est un génie de la science. Dès 1905, lorsqu'il publie le résultat de ses recherches, il surprend tous les physiciens de l'époque qui lui demandent alors d'enseigner à l'Université de Zurich. Il enchaîne les découvertes et les distinctions : la théorie de la relativité et son fameux $E = mc^2$, le prix Nobel de physique (1921), un poste de directeur à l'Institut des Sciences Avancées de Princeton, la physique quantique... Juif, il quitte l'Allemagne pour les Etats-Unis dès 1932, sentant l'influence grandissante des nazis. Il ne cesse alors d'agir en pacifiste militant et convaincu et demande au président des Etats-Unis de faire accélérer les recherches nucléaires pour contrer les avancées de l'Allemagne nazie dans ce domaine. Mais c'est avec douleur qu'il apprend l'utilisation de la bombe atomique à Hiroshima et à Nagasaki et s'engage alors dans une lutte contre la folie guerrière.

Le principal objectif de ce chapitre est de présenter **Qbfl**, notre prouveur QBF de type Q-DPLL, s'appuyant sur une nouvelle heuristique de branchement. Cette nouvelle heuristique a pour but de favoriser la génération de formules ren$\mathcal{QHF}$, pour lesquelles des algorithmes efficaces existent. Ainsi, l'algorithme de résolution pour les ren$\mathcal{QHF}$ décrit au paragraphe 4.4 est exploité au sein du prouveur **Qbfl**.

6.2 Le prouveur Qbfl

Qbfl[19] est un prouveur QBF qui utilise le prouveur SAT Limmat (version 1.3) comme un oracle SAT pour tester la vérité et la fausseté triviales. Il est basé sur une procédure Q-DPLL standard avec un branchement et/ou dépendant du groupe de quantificateurs (Cadoli *et al.* 2002b), et intègre la détection des faussetés et vérités triviales (Cadoli *et al.* 2002b). Il est adaptatif dans le sens où la période des tests de trivialité évolue avec son taux de succès. La propagation de littéraux unitaires et purs pour QBF (Cadoli *et al.* 2002b) est également intégrée. **Qbfl** réalise une détection de ren$\mathcal{QHF}$ à chaque point de choix et résout les ren$\mathcal{QHF}$ reconnues à l'aide de l'algorithme proposé par Kleine-Büning *et al.* (Kleine-Büning *et al.* 1995).

Algorithme 6.1 : Algorithme utilisé pour le prouveur **Qbfl**

fonction QBFL
Données : Une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ
Résultat : Vrai si Σ est valide, Faux sinon.
début
    $\Sigma \leftarrow$ SIMPLIFIER2(Σ) ;
    si $\Sigma = \{\}$ **alors**
        retourner Vrai ;
    sinon si $\{\} \in \Sigma$ **alors**
        retourner Faux ;
    $\epsilon \leftarrow$ RHCLOS(Σ) ;
    si $\epsilon \neq \emptyset$ **alors**
        $\Sigma' \leftarrow$ RENOMMER(Σ, ϵ) ;
        retourner Q-UNIT-RES(Σ') ;
    $\lambda \leftarrow$ CHOIX_LITTÉRAL ;
    si $\lambda \in \exists$ **alors**
        retourner QBFL($\Sigma_{\lambda\leftarrow 1}$) $\vee$ QBFL($\Sigma_{\lambda\leftarrow 0}$) ;
    sinon retourner QBFL($\Sigma_{\lambda\leftarrow 1}$) $\wedge$ QBFL($\Sigma_{\lambda\leftarrow 0}$) ;
fin

Une version itérative de l'algorithme 6.1, basée sur celle proposée dans (Giunchiglia *et al.* 2001b), est implantée dans notre prouveur. Dans cet algorithme, λ est un littéral et ϵ un ensemble de littéraux (éventuellement vide).

La fonction SIMPLIFIER2 est telle que la fonction SIMPLIFIER définie au paragraphe 3.1.2 mais prend en compte le caractère adaptatif de notre prouveur. En effet, si la propagation de littéraux unitaires et purs est réalisée à chaque point de choix, la période d'évaluation de la détection de la vérité triviale et de la fausseté triviale sur $\exists$ évolue au cours de l'exécution : si l'évaluation de la vérité triviale (resp. de la fausseté triviale sur $\exists$) échoue, c'est-à-dire si la sous-formule courante n'est pas trivialement vraie (resp. fausse sur $\exists$), alors la période d'évaluation de la vérité triviale (resp. de la fausseté triviale sur $\exists$), initialement égale

19. **Qbfl** est un prouveur développé en C (plus de 5000 lignes de code). Il est disponible à l'URL
`http://www.cril.univ-artois.fr/~letombe/qbfl`

à 1, est multipliée par 10, jusqu'à ce que cette période atteigne son maximum fixé à 100000. Dans le cas contraire, c'est-à-dire si la sous-formule est trivialement vraie (resp. fausse sur $\exists$), alors cette période est divisée par 10, jusqu'à ce qu'elle redevienne égale à 1. Clairement, au départ, la trivialité est évaluée à chaque point de branchement. En cas d'échec, elle ne sera évaluée que tous les 10 points de branchement, etc.

Qbfl accepte l'ensemble des heuristiques présentées au paragraphe 3.3. De plus, par un choix d'"option, il peut également utiliser l'heuristique Δ présentée au paragraphe suivant.

Qbfl résout les ren$\mathcal{QHF}$ à l'aide de la Q-unit-résolution (fonction Q-UNIT-RES) de Kleine-Büning *et al.* (Kleine-Büning *et al.* 1995) décrite au paragraphe 4.3, munie du renommage de variables selon la méthode décrite ci-après. RHCLOS est décrite au paragraphe 4.4 (algorithme 4.3) et retourne un Horn renommage quand celui-ci est possible, $\emptyset$ sinon. La fonction RENOMMER prend pour paramètres une ren$\mathcal{QHF}$ Σ et un ensemble de littéraux représentant un Horn renommage ϵ (comme décrit au paragraphe 4.4) et retourne une $\mathcal{QHF}$.

6.3 Une heuristique de branchement dirigée vers les ren$\mathcal{QHF}$

L'idée de base de notre approche est, à chaque point de choix rencontré lors de l'exécution de Q-DPLL, de déterminer (si elle existe) une interprétation partielle S_i de cardinal minimal construite sur les variables associées à la quantification Q_iX_i la plus externe de la formule courante $\Sigma_i = Q_iX_i \ldots, Q_nX_n.\phi_i$ associée au point de choix telle que $\Sigma_{i_{S_i}}$ soit une formule ren$\mathcal{QHF}$. Si une telle interprétation S_i (identifiée à un ensemble de littéraux) existe, alors la validité de $\Sigma_{i_{S_i}}$ peut être décidée en temps polynomial (cf. paragraphe 4.4). Or, la validité de $\Sigma_{i_{S_i}}$ nous renseigne sur la validité de Σ : soient $\Sigma_i = Q_iX_i \ldots Q_hX_h.\phi$ une $\mathcal{CNF}$-$h\mathcal{QBF}$ et S_i une interprétation partielle sur X_i identifiée à un ensemble de littéraux, si X_i est un ensemble de littéraux quantifiés existentiellement, alors si $\Sigma_{i_{S_i}}$ est valide, alors Σ l'est aussi ; si X_i est un ensemble de littéraux quantifiés universellement, alors si $\Sigma_{i_{S_i}}$ n'est pas valide, alors Σ ne l'est pas non plus.

Exemple 6.1
Considérons la $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ de matrice suivante :

$$\begin{bmatrix} (a \vee b \vee c) & \wedge \\ (\neg a \vee \neg b \vee \neg c) & \wedge \\ (\neg a \vee d) & \wedge \\ (\neg b \vee \neg c \vee d) & \end{bmatrix}.$$

Clairement, Σ n'est pas Horn renommable (ne considérer que les deux premières clauses). Or le fait d'affecter a à Faux nous permet d'obtenir une matrice Horn renommable et d'utiliser un algorithme polynomial pour résoudre la ren$\mathcal{QHF}$ associée, de matrice $\begin{bmatrix} (b \vee c) & \wedge \\ (\neg b \vee \neg c \vee d) & \end{bmatrix}$ (en l'occurrence, renommer c et d et appliquer l'algorithme de Q-unit-résolution de Kleine-Büning *et al.* (Kleine-Büning *et al.* 1995)). ■

Malheureusement, le problème consistant à déterminer si une telle interprétation S_i existe est calculatoirement difficile :

Proposition 6.1
Soit le problème HRC suivant :

- **Entrée :** une formule $\mathcal{CNF}$ Σ et un ensemble $X \subseteq Var(\Sigma)$;
- **Question :** existe-t-il $S \subseteq L_X$ tel que Σ_S est une formule Horn renommable $\mathcal{CNF}$?

HRC est **NP**-complet.

Preuve 6.1
L'appartenance est triviale : il suffit de deviner $S \subseteq L_X$ et de vérifier en temps polynomial que Σ_S est Horn renommable $\mathcal{CNF}$. Pour la difficulté, nous présentons une réduction depuis SAT pour une formule $\mathcal{CNF}$ vers HRC.

Soit $\varphi = \gamma_1 \wedge \cdots \wedge \gamma_n$ une formule $\mathcal{CNF}$. On associe à φ la $\mathcal{CNF}$ φ' telle que

$$\varphi' = \bigwedge_{i=1}^{n} (\gamma_i \vee new_1 \vee new_2 \vee new_3) \wedge (\gamma_i \vee \neg new_1 \vee \neg new_2 \vee \neg new_3)$$

avec

$$\{new_1, new_2, new_3\} \cap Var(\varphi) = \emptyset$$

et

$$X = Var(\varphi).$$

Si φ est satisfiable, alors il existe $S \subseteq L_X$ tel que S satisfait φ. Donc φ_S est l'ensemble vide de clauses. Or, cet ensemble correspond à une formule Horn renommable.

Si φ n'est pas satisfiable, alors quel que soit $S \subseteq L_X$, au moins une des clauses γ_i de φ n'est pas satisfaite par S. Alors φ_S contient par construction les deux clauses $new_1 \vee new_2 \vee new_3$ et $\neg new_1 \vee \neg new_2 \vee \neg new_3$ qui font que φ_S n'est pas satisfiable. □

Soit $\Sigma = \exists X.\Psi$ où Ψ est une $\mathcal{CNF}$-$\mathcal{QBF}$. La réduction donnée dans la preuve de la proposition 6.1 montre que déterminer s'il existe $S \subseteq L_X$ telle que Σ_S est ren$\mathcal{QHF}$ est un problème **NP**-difficile.

Pour cette raison, on ne considère qu'une approche heuristique, qui vise, par calcul successif des littéraux qui la composent, à construire une interprétation partielle S_i rendant $\Sigma_{i_{S_i}}$ ren$\mathcal{QHF}$. Cette technique n'assure aucune garantie de minimalité quant au cardinal du premier S_i obtenu (s'il en existe un) dans l'arbre de recherche. En revanche, il est clair que si une interprétation partielle S_i rendant $\Sigma_{i_{S_i}}$ ren$\mathcal{QHF}$ existe, elle sera trouvée (car si $\Sigma_{i_{S_i}}$ est ren$\mathcal{QHF}$ et $S_i \subseteq S'_i$, alors $\Sigma_{i_{S'_i}}$ est aussi ren$\mathcal{QHF}$), mais éventuellement après avoir considéré au préalable toutes les autres interprétations partielles construites sur X_i.

Le principe de notre heuristique (Coste-Marquis *et al.* 2005a; Letombe 2005) est de brancher en priorité sur des variables pour lesquelles la propagation conduit à une formule Horn renommable, ou à une formule « presque » Horn renommable. Nous utilisons une adaptation de l'algorithme proposé par Hébrard (cf. paragraphe 4.4) pour détecter les formules Horn renommables. Cet algorithme construit un ensemble de littéraux à renommer afin d'obtenir une formule Horn. Si un tel ensemble existe, il est possible d'utiliser l'algorithme en temps polynomial de Kleine-Büning (Kleine-Büning *et al.* 1995) afin de résoudre l'instance. Sinon, nous utilisons les ensembles $\text{CLOS}_\Sigma(l)$ calculés dans l'algorithme de détection de Horn renommabilité pour mesurer heuristiquement à quelle distance la formule courante Σ est d'une ren$\mathcal{QHF}$: plus la mesure calculée est grande, plus Σ est considérée comme proche d'une ren$\mathcal{QHF}$. Nous appelons cette mesure *distance de contradiction*.

Définition 6.1 (Distance de contradiction)
La « **distance** »[20] **à une contradiction de Horn renommabilité** δ_l^Σ *pour un littéral* l *dans une* $\mathcal{QHF}$ Σ *est définie comme suit :*

$$\delta_l^\Sigma = \begin{cases} 0 \; si \; \nexists v \mid l \Rightarrow_\Sigma v \\ 1 \; si \; l \Rightarrow_\Sigma t \; et \; l \Rightarrow_\Sigma t^c \\ 1 + min(\{\delta_v^\Sigma \mid l \Rightarrow_\Sigma v\}) \; sinon. \end{cases}$$

20. On notera l'abus de langage. Ce n'est pas une distance au sens mathématique.

Afin de calculer cette distance, il est nécessaire de légèrement modifier l'algorithme proposé par Hébrard pour calculer $\textsc{Clos}_\Sigma(l)$: cet algorithme réalise une recherche en profondeur d'abord (DFS) alors que le nôtre utilise une recherche en largeur d'abord (BFS). Cette adaptation est nécessaire si l'on veut calculer la distance de contradiction. L'adaptation réalisée est présentée dans l'algorithme 6.2. La partie droite de l'algorithme 6.2 décrit le calcul de distance de contradiction par recherche en largeur. Ce dernier algorithme est de complexité linéaire en temps ($\mathcal{O}(n+m)$, avec n le nombre de littéraux et m le nombre de clauses, si l'on considère que les clauses ont une longueur bornée).

Algorithme 6.2 : Algorithme de Horn renommage de Hébrard (à gauche) et calcul de la distance de contradiction (à droite)

fonction RHCLOS
Données : une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ
Résultat : $\emptyset$ si Σ n'est pas Horn renommable,
un Horn renommage R pour Σ sinon.
début
  $R \leftarrow \emptyset$;
  pour chaque $p \in Var(\Sigma)$ **faire**
    si $p \notin R$ **et** $\neg p \notin R$ **alors**
      si $\forall q \in \textsc{Clos}_\Sigma(p) \setminus R$,
      $q^c \notin \textsc{Clos}_\Sigma(p) \setminus R$ **alors**
        # $\textsc{Clos}_\Sigma(p) \setminus R$ est cohérent
        $R \leftarrow R \cup (\textsc{Clos}_\Sigma(p) \setminus R)$;
      sinon si $\forall q \in \textsc{Clos}_\Sigma(\neg p) \setminus R$,
      $q^c \notin \textsc{Clos}_\Sigma(\neg p) \setminus R$ **alors**
        # $\textsc{Clos}_\Sigma(\neg p) \setminus R$ est cohérent
        $R \leftarrow R \cup (\textsc{Clos}_\Sigma(\neg p) \setminus R)$;
      sinon retourner $\emptyset$;
fin

fonction δ
Données : une $\mathcal{CNF}$-$h\mathcal{QBF}$ Σ, un littéral l
Résultat : la distance de contradiction δ_l^Σ de l
dans Σ.
début
  $Closl \leftarrow \{l\}$;
  $Distance \leftarrow 0$;
  ENFILER$(Filel, l)$;
  tant que non FILEVIDE$(Filel)$ **faire**
    $m \leftarrow$ DEFILER$(Filel)$;
    $Engendre \leftarrow \{t^c \mid m \Rightarrow_\Sigma t\}$;
    pour chaque $e \in Engendre$ **faire**
      si $e^c \in Closl$ **alors**
        retourner $Distance + 1$;
      sinon si $e \in Closl$ **alors**
        $Closl \leftarrow Closl \cup \{e\}$;
        ENFILER$(Filel, e)$;
    $Distance \leftarrow Distance + 1$;
  retourner $Distance + 1$;
fin

La fonction DEFILER (resp. ENFILER) est une fonction standard de défilement (resp. d'enfilement) qui prend pour paramètre une file et retourne le premier élément de la file à avoir été inséré qui se trouve en tête de file (resp. qui prend pour paramètre une file et un élément qu'elle insère en queue de file).

Exemple 6.2 (Distance de contradiction en utilisant une DFS)
Considérons à nouveau l'exemple 4.6 :

$$\begin{bmatrix} (a \vee b \vee c) & \wedge \\ (a \vee \neg d \vee e) & \wedge \\ (b \vee \neg c \vee \neg d) & \wedge \\ (\neg c \vee \neg e) & \wedge \\ (\neg b \vee c) & \end{bmatrix}.$$

L'algorithme proposé par Hébrard réalise une DFS pour calculer l'image de $\{a\}$ par la fermeture réflexive transitive de $\Rightarrow_\Sigma$:

– Profondeur 0 : $\textsc{Clos}_\Sigma(a) = \{a\}$;
– Profondeur 1 : a est dans les clauses $(a \vee b \vee c)$ et $(a \vee \neg d \vee e)$;

– $(a \vee \neg d \vee e) \rightsquigarrow$ CLOS$_\Sigma(a)$ = $\{a, d, \neg e\}$;
– Profondeur 2 : $\neg e$ est dans la clause $(\neg c \vee \neg e)$;
– $(\neg c \vee \neg e) \rightsquigarrow$ CLOS$_\Sigma(a)$ = $\{a, d, \neg e, c\}$;
– Profondeur 3 : c est dans les clauses $(a \vee b \vee c)$ et $(\neg b \vee c)$;
– $(a \vee b \vee c) \rightsquigarrow$ CLOS$_\Sigma(a)$ = $\{a, d, \neg e, c, \neg a, \neg b\}$;
– $\{a, d, \neg e, c, \neg a, \neg b\} \equiv$ Faux.

■

Avec la DFS, aucune garantie ne nous permet de conclure que la contradiction trouvée est obtenue après un nombre minimal d'étapes.

Exemple 6.3 (Distance de contradiction (en utilisant une BFS))
L'algorithme δ produira sur le même exemple le résultat suivant :

– Niveau 0 : CLOS$_\Sigma(a)$ = $\{a\}$;
– Niveau 1 :
  – a apparaît dans les clauses $(a \vee b \vee c)$ et $(a \vee \neg d \vee e)$;
  – $(a \vee b \vee c) \rightsquigarrow$ CLOS$_\Sigma(a)$ = $\{a, \neg b, \neg c\}$;
  – $(a \vee \neg d \vee e) \rightsquigarrow$ CLOS$_\Sigma(a)$ = $\{a, \neg b, \neg c, d, \neg e\}$;
– Niveau 2 :
  – $\neg b$ apparaît dans la clause $(\neg b \vee c)$;
  – $\neg c$ apparaît dans les clauses $(b \vee \neg c \vee \neg d)$ et $(\neg c \vee \neg e)$;
  – $(\neg c \vee \neg e) \rightsquigarrow$ CLOS$_\Sigma(a)$ = $\{a, \neg b, \neg c, d, \neg e, e\}$;
  – $\{a, \neg b, \neg c, d, \neg e, e\} \equiv$ Faux.

La contradiction apparaît au second niveau. Nous concluons ainsi que a est à une distance de contradiction δ_a^Σ de 2 d'une formule Horn renommable dans Σ. ■

Cette distance est souvent insuffisante pour élire une seule variable. C'est pourquoi nous l'utilisons au sein d'une heuristique souvent plus discriminante à la Jeroslow-Wang (Jeroslow & Wang 1990) avec un réglage largement utilisé dans les prouveurs complets pour des k-SAT aléatoires hérités de POSIT (Freemann 1995). Nous restreignons simplement la sélection de variables dans la portée du quantificateur le plus externe et nous branchons en priorité sur le littéral maximisant la distance de contradiction.

Définition 6.2 (Heuristique Δ)
Soit une $\mathcal{CNF}$-$n\mathcal{QBF}\Sigma = Q_1X_1 \ldots Q_nX_n.\phi$. Notre heuristique Δ de choix de littéral de Σ au sein de **Qbfl** *est comme suit. Pour chaque variable x de X_1 :*

– $\Delta_x^\Sigma = 1024 \times \delta_x^\Sigma \times \delta_{\neg x}^\Sigma + \delta_x^\Sigma + \delta_{\neg x}^\Sigma$;
– *la variable x est choisie si $\forall y \in X_1$, $\Delta_x^\Sigma \leq \Delta_y^\Sigma$;*
– *le littéral x est enfin choisi en priorité si $\delta_x^\Sigma > \delta_{\neg x}^\Sigma$, x^c sinon.*

Cette heuristique et celle de Jeroslow et Wang (Jeroslow & Wang 1990) ont été utilisées lors de nos expérimentations.

6.4 Résultats expérimentaux

Les résultats empiriques présentés dans ce paragraphe et dans l'annexe ont été obtenus sur des PIV 3GHz et de 512MB de RAM.

Nous comptabilisons les temps CPU des instances résolues et le nombre de benchmarks résolus afin de comparer les comportements des prouveurs.

6.4.1 Benchmarks « polynomiaux »

L'étude de fragments traitables pour QBF est aussi importante d'un point de vue pratique pour le développement de prouveurs. En particulier, il est intéressant de connaître le comportement des prouveurs QBF actuels sur des instances $\mathcal{QBF}$ dont la matrice est constituée de clauses de Horn ou Horn renommables.

Les matrices de ces formules, en tant qu'instances SAT, peuvent être facilement résolues par n'importe quel prouveur SAT actuel : bien que ces prouveurs n'utilisent pas d'algorithmes dédiés à de tels fragments polynomiaux pour SAT, ces instances « faciles en théorie » le sont aussi en pratique pour les prouveurs SAT. A contrario, nous avons observé que les prouveurs QBF actuels sont peu performants sur les instances QBF correspondantes. En effet, nous avons soumis des instances ren$\mathcal{QBF}$ à l'évaluation des prouveurs QBF 2005 ; les résultats obtenus par les prouveurs participant à cette évaluation sont donnés à la table 6.1.

Prouveur	# vrai /81	# faux /82	Total /163
GRL	7	59	66
qbfbdd	42	15	57
QchaffLearn	2	53	55
QMRes	0	0	0
quantor	0	10	10
openqbf	0	20	20
semprop	54	55	109
ssolve	78	77	155
sKizzo 0.4	0	33	33
sKizzo 0.5	0	32	32
walkqsat	0	39	39
yquaffle	40	35	75

TAB. 6.1 – *Tableau comparatif des prouveurs* QBF *de l'évaluation comparative 2005 sur les instances de formules ren$\mathcal{QHF}$ que nous avons soumises.*

Dans cette table, pour chaque prouveur participant à l'évaluation comparative des prouveurs QBF 2005, dans la colonne « # vrai /81 » (resp. « # faux /82 », « Total /163 »), nous donnons les nombre d'instances de formules ren$\mathcal{QHF}$ valides (resp. incohérentes, total) sur les 81 (resp. 82, 163) soumises qu'il a résolues dans le temps imparti.

On peut observer que seuls deux prouveurs sont capables de résoudre plus de 50% des instances considérées dans le temps imparti. De notre point de vue, les performances des prouveurs actuels sur ces instances montrent que les approches actuelles, issues pour la plupart de généralisations de techniques pour SAT, i.e. d'algorithmes de type Q-DPLL, sont largement améliorables. En effet, ces algorithme ne sont pas capables de résoudre systématiquement des instances a priori « faciles ».

En conséquence, nous pensons que le développement de nouveaux algorithmes pour des fragments polynomiaux de QBF peut aussi aider au développement d'algorithmes généraux plus performants.

Afin d'affiner les conclusions, nous avons également effectué des expérimentations de notre côté. Nos premières expérimentations concernent des ensembles de benchmarks de $\mathcal{QHF}$ et ren$\mathcal{QHF}$ engendrés aléatoirement[21], comme suit :

$\mathcal{QHF}$ Pour chaque clause, nous choisissons aléatoirement sa taille entre 1 et 50 selon une distribution uniforme. Ensuite, le littéral positif de la clause est choisi aléatoirement selon une distribution uniforme.

21. Ces générateurs sont disponibles sur `http://www.cril.univ-artois.fr/~letombe/qbfg`.

Nous complétons la clause en choisissant aléatoirement les littéraux négatifs selon une distribution uniforme en supprimant à chaque étape le littéral (et son complémentaire) retenus à l'étape précédente de l'ensemble des possibilités.

ren$\mathcal{QHF}$ Nous commençons par choisir le nombre de variables à renommer. Ensuite, les variables à renommer sont sélectionnées avant d'engendrer les clauses par le même procédé que celui utilisé pour engendrer les $\mathcal{QHF}$ mais en renommant les littéraux sélectionnés.

Nous avons choisi de n'utiliser qu'une alternance de quantificateurs, $\forall X \exists Y$, tel que $|X| > \alpha$ avec $2 < \alpha < 2/3 \times \#V$ avec $\#V$ le nombre de variables de la formule.

Ces benchmarks sont triés par groupes de 1000 (resp. 100) instances de $\mathcal{QHF}$ (resp. ren$\mathcal{QHF}$). Chaque instance a 400 variables et nous augmentons le ratio du nombre de clauses sur le nombre de variables de 3 à 6 par pas de 0,1.

Une limite de temps est fixée à 60 secondes. Notons que cette limite est de 3 ordres de grandeur plus élevée que le temps de résolution de notre prouveur dédié à ce type d'instances. Des résultats similaires ont été obtenus avec une limite de temps de 300 secondes.

Afin de comparer **Qbfl** (version 1.7) avec d'autres prouveurs, nous avons choisi d'utiliser QuBE-Rel[22] (version 1.3) et Semprop[23] (version 010604) pour ces expérimentations : ces deux prouveurs se sont révélés parmi les meilleurs des prouveurs QBF ayant participé aux deux premières évaluations de prouveurs QBF 2003 et 2004 (Le Berre *et al.* 2003; Le Berre *et al.* 2005) et sont librement accessibles.

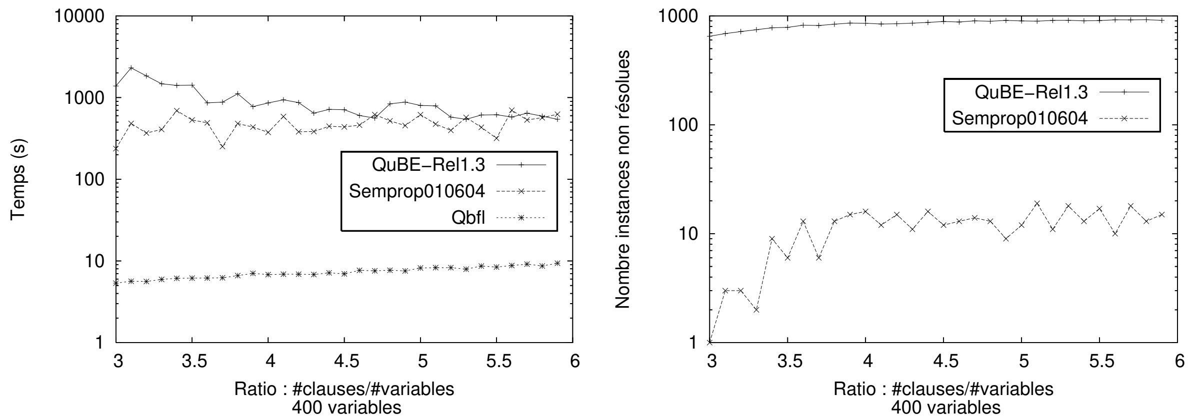


FIG. 6.1 – *Comparaison entre Semprop, QuBE et Qbfl sur des ensembles d'instances* $\mathcal{QHF}$.

La figure 6.1 présente des comparaisons expérimentales sur les $\mathcal{QHF}$. Les temps fournis sont la somme des temps nécessaires au prouveur considéré pour résoudre (au plus) les 1000 instances. **Qbfl** étant conçu pour ce type de problèmes, il résout tous ces benchmarks très facilement. Il est intéressant de noter la différence de comportement des deux autres prouveurs QBF : **Semprop** peut très souvent (dans environ 99% des cas) résoudre les instances rapidement (mais deux ordres de grandeur plus lentement que notre prouveur) mais échoue sur certains d'entre eux. D'un autre côté, **QuBE** ne peut résoudre la plupart de ces benchmarks. La conclusion manifeste de ces expérimentations est que la résolution d'instances de restrictions polynomiales pour QBF est difficile pour les prouveurs QBF actuels.

La figure 6.2 présente des comparaisons expérimentales sur les benchmarks ren$\mathcal{QHF}$. Les temps fournis sont la somme des temps nécessaires au prouveur considéré pour résoudre (au plus) les 100 instances

22. http://www.star.dist.unige.it/~qube
23. http://www4.in.tum.de/~letz/semprop

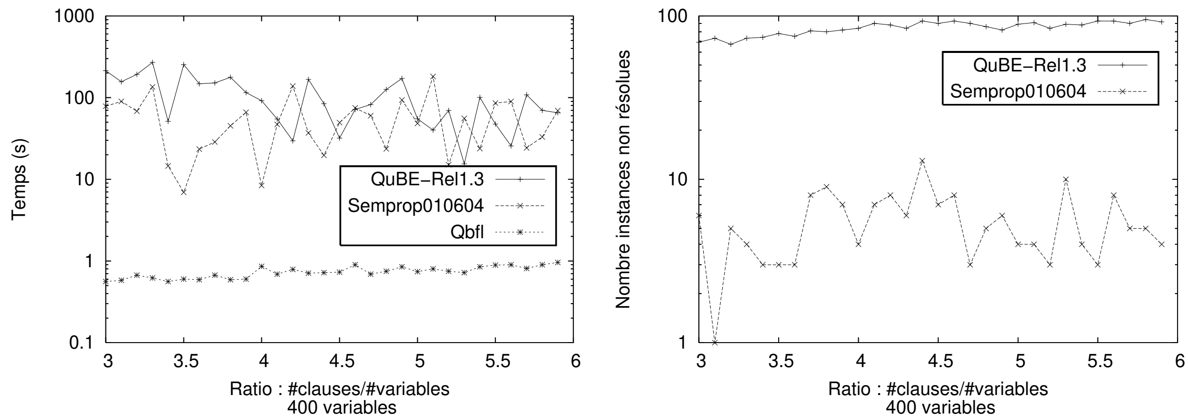


FIG. 6.2 – *Comparaison entre Semprop, QuBE et Qbfl sur des ensembles d'instances ren$\mathcal{QHF}$.*

considérées. Le comportement des prouveurs sur ce type d'instances est relativement similaire à celui obtenu sur les $\mathcal{QHF}$. Cependant, ces benchmarks apparaissent plus difficiles pour **Semprop** qui échoue dans 10% de cas.

Notons que **Qbfl** n'apparaît pas sur le côté droit des graphes dans les figures 6.1 et 6.2 car notre prouveur a résolu tous ces benchmarks.

Observons aussi que le temps d'exécution de **Qbfl** pour résoudre ces formules polynomiales croît lentement et régulièrement.

Les quelques expérimentations réalisées avec Quantor (Biere 2004), qui est un prouveur relativement différent des autres prouveurs QBF, montrent que celui-ci n'arrive pas non plus à résoudre des instances $\mathcal{QHF}$ et ren$\mathcal{QHF}$ en temps raisonnable. Il serait intéressant d'étudier également de façon approfondie le comportement du prouveur qbfbdd développé par Audemard et Saïs (Audemard & Saïs 2004; Audemard & Saïs 2005) sur ces benchmarks car il n'impose pas de contrainte sur l'ordre des variables. Malheureusement, les premiers résultats expérimentaux montrent que même ce prouveur ne peut résoudre facilement ces instances.

Évidemment, on ne peut pas conclure de ces premières expérimentations que notre approche est utile : elles révèlent uniquement que notre prouveur peut détecter et résoudre des formules $\mathcal{QHF}$ et ren$\mathcal{QHF}$ efficacement et que ces formules ne sont pas triviales pour les prouveurs QBF actuels.

6.4.2 Évaluations comparatives 2004 et 2005

6.4.2.1 Benchmarks de l'évaluation QBF 2004

Il est intéressant d'observer le comportement de notre prouveur sur l'ensemble des benchmarks de l'évaluation 2004.

Nous nous focalisons dans un premier temps sur les formules proposées par G. Pan, résultant de la traduction vers QBF d'instances du problème de satisfiabilité pour la logique modale K et présentées à TANCS'98 (Balsiger *et al.* 2000). Ces 378 instances - 9 types de 21 benchmarks valides and 9 non valides - ont été proposées pour l'évaluation de prouveurs QBF 2003 et la plupart d'entre eux ont été classés parmi les instances *difficiles* de l'évaluation, i.e. celles résolues par au plus un prouveur. Depuis 2003, des prouveurs comme Quantor (Biere 2004) se sont améliorés et ces benchmarks ne sont plus classés parmi les instances difficiles.

Les résultats obtenus par **Qbfl** avec une limite de temps de 900 secondes sur ces instances sont présentés dans le tableau 6.2. Des résultats plus détaillés sont donnés en annexe. Pour chaque type d'instance, la

Type d'instance	Jeroslow-Wang		Δ	
	%résolus	%HR	%résolus	%HR
k_branch_n	4.76	25.26	**9.52**	9.29
k_d4_n	4.76	13.25	4.76	5.63
k_dum_n	4.76	11.69	**23.80**	11.51
k_grz_n	0	-	**61.90**	11.94
k_lin_n	9.52	20.00	9.52	24.26
k_path_n	9.52	5.47	**14.28**	12.12
k_ph_n	23.80	2.66	23.80	11.73
k_poly_n	9.52	0	**14.28**	6.66
k_t4p_n	4.76	11.37	4.76	26.62
Total valides	7.93	11.21	**18.51**	13.30
k_branch_p	4.76	24.56	4.76	9.33
k_d4_p	9.52	26.33	**14.28**	25.34
k_dum_p	4.76	11.71	**14.28**	11.40
k_grz_p	0	-	0	-
k_lin_p	9.52	11.88	**19.04**	8.65
k_path_p	14.28	7.43	**19.04**	7.64
k_ph_p	19.04	10.06	19.04	6.89
k_poly_p	4.76	12.91	**9.52**	11.04
k_t4p_p	0	-	**4.76**	26.02
Total faux	7.40	14.98	**11.64**	13.28
Total	7.67	13.09	**15.07**	13.29

TAB. 6.2 – *Pourcentages de benchmarks résolus et de ren$\mathcal{QHF}$ atteintes.*

colonne « %résolus » contient le taux d'instances résolues parmi l'ensemble des instances de ce type et la colonne « %HR » contient les taux de formules Horn renommables rencontrées, i.e. le nombre de fois où le test de Horn renommabilité a réussi divisé par le nombre de fois où il a été appelé. Ces résultats sont présentés pour **Qbfl** muni de l'heuristique de Jeroslow-Wang ainsi que pour l'heuristique Δ.

La principale observation est que beaucoup plus de benchmarks (environ le double) sont résolus par notre prouveur à l'aide de l'heuristique Δ par rapport à l'heuristique de Jeroslow-Wang. Le résultat est significatif, spécialement pour les instances *k_grz_n* : 30 benchmarks sont résolus avec l'heuristique de Jeroslow-Wang alors que 59 sont résolues avec Δ. Malheureusement, il est difficile de savoir si ces bons résultats sont dûs à la détection de formules Horn renommables ou si le changement d'heuristique en est le seul responsable. En effet, nous n'avons aucun moyen de comparer le nombre de formules Horn renommables au cours de la recherche entre les deux heuristiques : réduire l'espace de recherche réduit également le nombre de formules Horn renommables détectées.

Nous avons testé l'heuristique Δ sur d'autres types de formules pour lesquelles le comportement de **Qbfl** n'est que peu altéré. Dans les centaines d'instances proposées par A. Ayari (72), C. Castellini (169), M. Mneimneh et K. A. Sakallah (202), J. Rintanen (67) et C. Scholl et B. Becker (64)[24], nous résolvons un total de 283 instances à l'aide des deux heuristiques (119 valides et 164 non valides). Seules 14 instances différencient les deux heuristiques : 264 instances sont résolues avec Jeroslow-Wang et 250 à l'aide de Δ. En prenant en compte les benchmarks de Pan, **Qbfl** fonctionne au total mieux avec que sans l'heuristique Δ. Les résultats sont résumés dans le tableau 6.3.

6.4.2.2 Résultats de l'évaluation 2005

Ces expérimentations ne seraient pas complètes si elle ne comportaient pas un paragraphe comparatif de **Qbfl** avec les prouveurs QBF actuels sur des instances de formules modélisant divers problèmes. C'est

24. Tous ces benchmarks sont disponibles sur `http://www.qbflib.org`.

Type d'instance	Total	Jeroslow-Wang		Δ	
		%solved	%HR	%solved	%HR
DFlipFlopUnSat	10	100.00	0	100.00	0
SzymanskiPSat	12	**100.00**	9.52	75.00	10.67
ToiletASat	21	**100.00**	14.28	71.42	20.00
ToiletAUnSat	56	69.64	0	**85.71**	0
ToiletCSat	29	**100.00**	10.34	65.51	15.78
ToiletCUnSat	56	82.14	0	**94.64**	0
ToiletGSat	7	100.00	14.28	100.00	14.28
s27	4	**50.00**	94.61	25.00	97.30
s3271	21	100.00	0	100.00	0
ChainSat	12	66.66	99.69	**75.00**	99.72
ImplSat	10	**100.00**	0	70.00	0
LognUnSat	4	50.00	50.00	50.00	50.00
R3cnfSat	13	**100.00**	7.71	92.30	0.02
R3cnfUnSat	7	**100.00**	0.00	71.42	0.01
ToiletSat	5	**100.00**	0.00	40.00	0.00
ToiletUnSat	3	66.66	0	66.66	0
C432	8	25.00	0	25.00	0
C499	8	25.00	0	25.00	0
C880	8	0	-	**12.50**	1.51
comp	8	**87.50**	0.73	50.00	0
term1	8	50.00	0.78	50.00	1.87
z4ml	8	100.00	0	100.00	0
Total	334	**76.94**	8.08	72.74	8.73

TAB. 6.3 – *Pourcentages de benchmarks résolus et de ren$\mathcal{QHF}$ atteintes.*

notre propos dans le paragraphe qui suit.

Le tableau 6.4 fournit les résultats obtenus par les prouveurs QBF participant à l'évaluation comparative 2005.

Dans cette table, pour chaque prouveur participant à l'évaluation comparative des prouveurs QBF 2005, dans la colonne « # vrai » (resp. « # faux », « Total /3177 »), nous donnons les nombre d'instances de formules valides (resp. incohérentes, total) sur les 3177 instances de la base de tests qu'il a résolues dans le temps imparti.

Les deux lignes en caractères gras présentent les résultats de **Qbfl** et ceux du prouveur qui semble être le meilleur pour cette évaluation : SSOLVE. On constate rapidement que **Qbfl** est loin d'être le meilleur prouveur mais qu'il obtient des résultats intéressants sur les instances de formules valides puisqu'il se classe troisième parmi l'ensemble des prouveurs sur ce type d'instances.

6.5 Résumé

Nous avons construit un prouveur QBF de type Q-DPLL qui intègre une heuristique basée sur l'algorithme de détection de formules Horn renommables proposé par Hébrard. Nous avons évalué ce prouveur et l'avons comparé à deux prouveurs existants. Nous avons noté qu'en pratique, les prouveurs QBF actuels ne sont pas capables de résoudre en un temps raisonnable certaines instances de $\mathcal{QHF}$ ou ren$\mathcal{QHF}$.

Prouveur	# vrai	# faux	Total /3177
GRL	843	754	1597
qbfbdd	229	142	371
Qbfl	**949**	**584**	**1533**
QChaffLearn	828	836	1664
QMRes	535	126	661
quantor	683	319	1002
openqbf	786	596	1382
semprop	1008	911	1919
ssolve	**1128**	**913**	**2041**
sKizzo v0.4	736	889	1625
sKizzo v0.5	795	903	1698
WalkQSAT	873	802	1675
yquaffle	742	698	1440

TAB. 6.4 – *Tableau comparatif des prouveurs* QBF *de l'évaluation 2005.*

Conclusion générale

> *« C'est là en effet un des grands et merveilleux caractères des beaux livres que pour l'auteur ils pourraient s'appeler « Conclusions » et pour le lecteur « Incitations ». »*
> Marcel Proust[25]

Dans cette thèse, nous nous sommes intéressés au problème de la validité des formules booléennes quantifiées. Vu la complexité du problème, nous avons considéré des restrictions de celui-ci, portant sur la matrice des instances. Notre objectif principal était double : (1) identifier la complexité de QBF pour des restrictions non considérées jusqu'ici et (2) explorer dans quelle mesure les classes polynomiales pour QBF peuvent être exploitées au sein d'un prouveur général afin d'améliorer son efficacité.

Concernant le premier point, nous avons montré que QBF restreint aux fragments cibles pour la compilation de « connaissances » étudiés dans (Darwiche & Marquis 2002), qui sont traitables pour SAT, reste **PSPACE**-complet et donc intraitable en pratique. Nous avons également mis en évidence le lien étroit qui existe entre notre étude et le problème de compilabilité de QBF.

Concernant le second point, nous avons décrit une nouvelle heuristique de branchement Δ visant à promouvoir la génération de formules ren$\mathcal{QHF}$ durant le parcours de l'arbre de recherche effectué par une procédure de type Q-DPLL. Les résultats expérimentaux présentés montrent qu'en pratique, hormis notre prouveur **Qbfl**, les prouveurs QBF actuels ne sont pas capables de résoudre facilement les instances issues des classes $\mathcal{QHF}$ et ren$\mathcal{QHF}$; ceci suffit à justifier l'intérêt de l'approche suivie. Les résultats obtenus montrent également que, muni de l'heuristique Δ, **Qbfl** améliore significativement ses performances sur certains types d'instances, mais qu'il conserve tout de même des résultats moyens en général, comparé aux prouveurs QBF actuels.

Perspectives

Ce travail appelle un certain nombre de perspectives. L'une d'entre elles consiste à étendre davantage le panorama de complexité de QBF. En particulier, il serait intéressant de déterminer en quelles mesures la notion de modèles K-booléens de (Kleine-Büning *et al.* 2003), dans le cas où K est une classe de booléens qui peut être encodée en espace polynomial, pourrait être exploitée pour donner naissance à de nouvelles restrictions de QBF qui sont calculatoirement plus faciles (selon les hypothèses usuelles de la théorie de la complexité).

Dans le même ordre d'idées, une perspective est d'élargir notre investigation à l'étude de restrictions de QBF basées sur les quantifications. Une première approche esquissée au paragraphe 5.2.2 a concerné

25. Proust (Marcel), écrivain français, né à Paris (1871–1922). Marcel Proust est élevé dans un milieu bourgeois et cultivé. Mais la découverte de son homosexualité ouvre des failles dans son monde protégé. Il fait des études de droit, puis de lettres et intègre le milieu artistique et mondain de Paris. Il commence une carrière de journaliste-chroniqueur, voyageant en Europe. Mais la mort de sa mère déstabilise encore sa personnalité fragile et inquiète. Son activité littéraire s'intensifie, et c'est dans la solitude qu'il crée l'un des romans occidentaux les plus achevés, « À la recherche du temps perdu ». Marcel Proust reçut en 1919 le prix Goncourt pour « À l'ombre des jeunes filles en fleur », le deuxième volet de la trilogie. Dans l'ensemble de son oeuvre, il questionne les rapports entre temps, mémoire et écriture.

les restrictions sur le nombre d'alternances des quantifications des fragments $\mathcal{PI}$ et $\mathcal{IP}$. À l'instar de ce que nous avons fait pour $\mathcal{PI}$ et $\mathcal{IP}$, il s'agit d'étudier des restrictions portant conjointement sur la nature du préfixe et de la matrice des instances. En effet, d'un point de vue théorique, ne considérer que des restrictions sur le préfixe (i.e., sans aucune exigence sur la matrice) ne semble pas aussi intéressant. Alors que le fait de limiter le nombre d'alternances de quantificateurs dans le préfixe provoque immédiatement une diminution de la complexité de QBF de **PSPACE** à un niveau donné dans la hiérarchie polynomiale, cela ne conduit pas à la traitabilité (sous les hypothèses habituelles de la théorie de la complexité) ; ainsi, dans le cas limite où aucune alternance de quantificateurs n'apparaît dans le préfixe, QBF se réduit à SAT ou à UNSAT (dépendant de la nature des quantifications des variables), et aucun de ces problèmes ne peut vraisemblablement appartenir à **P** si aucune supposition sur la matrice n'est faite. Bien entendu, cela ne signifie pas que la façon dont les quantifications sont traitées n'a pas d'impact sur l'efficacité des prouveurs QBF. Par exemple, une $\mathcal{QBF}$ Σ de la forme $\forall\{x_1, \ldots, x_{n-1}\}\exists\{x_n\}.\phi$ où ϕ est une formule $\mathcal{CNF}$ peut être résolue en temps quadratique (construire une représentation $\mathcal{CNF}$ de $\exists\{x_n\}.\phi$ en utilisant la résolution pour oublier x_n dans ϕ, réduire ensuite toutes les clauses résultantes en y supprimant tout littéral construit à partir de $\{x_1, \ldots, x_{n-1}\}$; Σ est valide si et seulement si la formule $\mathcal{CNF}$ résultante ne contient pas la clause vide). Pourtant, les prouveurs QBF de type Q-DPLL qui traitent les quantificateurs du plus externe au plus imbriqué peuvent requérir un temps exponentiel pour résoudre des formules de ce type. Une étude plus approfondie de l'interaction entre les restrictions basées sur le préfixe et celles basées sur la matrice constitue donc une perspective de recherche de ce travail.

Pour poursuivre le volet théorique de ce travail, nous envisageons aussi d'étudier la possibilité d'étendre les méthodes de résolution polynomiales pour SAT correspondant aux fragments incomplets décrits au paragraphe 4.5, tels que celui des formules ordonnées par exemple, au cas QBF. Ce dernier point ouvre des perspectives très intéressantes. En effet, les formules ordonnées sont plus expressives que les formules Horn. De plus, il existe une extension des formules ordonnées, les formules ordonnées renommables qui sont également plus expressives que les formules Horn renommables.

Si de telles classes se révèlent également polynomiales pour QBF, il serait intéressant d'étudier comment les exploiter au sein d'un prouveur de type Q-DPLL (soit de façon passive, par un test de détection systématique ou non aux points de choix, soit de façon plus active – comme nous l'avons fait pour les formules Horn renommables $\mathcal{CNF}$) en mettant au point des heuristiques visant à leur promotion.

D'un point de vue plus pratique, nous souhaitons aussi améliorer l'efficacité de notre prouveur en y incorporant les techniques utilisées dans les prouveurs les plus efficaces (en particulier le retour arrière intelligent).

Annexe :
Résultats expérimentaux

Les résultats complets concernant l'évaluation de **Qbfl** que nous avons réalisée sur les instances de G. Pan sont présentés dans les tableaux 1 à 4. Pour chaque heuristique étudiée ici (Jeroslow-Wang et Δ), les colonnes indiquent :

Type vrai si le benchmark est valide, faux sinon,

#Branch nombre de choix de variables réalisés par l'heuristique,

#T (resp.#F) nombre de formules Horn renommables vraies (resp. fausses) atteintes durant la résolution,

Temps le temps nécessaire à **Qbfl** pour résoudre l'instance ou « – » si l'instance n'a pas été résolue au bout de 900 secondes.

Instance	Jeroslow-Wang						Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_dum_n-1	vrai	1985397	232096	0	44.51	11.69	vrai	145638	16767	0	5.53	11.51
k_dum_n-2	-	-	-	-	-	-	vrai	327192	37689	0	11.40	11.51
k_dum_n-3	-	-	-	-	-	-	vrai	3518506	405224	0	150.34	11.51
k_dum_n-4	-	-	-	-	-	-	vrai	17139520	1973956	0	693.29	11.51
k_dum_n-5	-	-	-	-	-	-	vrai	10361259	1196794	0	517.77	11.51
k_dum_p-1	faux	1535321	179913	0	27.47	11.71	faux	92308	10599	0	2.82	11.48
k_dum_p-2	-	-	-	-	-	-	faux	1471511	167384	0	50.33	11.37
k_dum_p-3	-	-	-	-	-	-	faux	1471511	167384	0	51.29	11.37

TAB. 1 – *k_dum*

Instance	Jeroslow-Wang						Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_grz_n-1	-	-	-	-	-	-	vrai	45429	5711	0	1.78	12..57
k_grz_n-2	-	-	-	-	-	-	vrai	8454473	984272	0	392.57	11.64
k_grz_n-3	-	-	-	-	-	-	vrai	65785	8192	0	3.32	12.45
k_grz_n-4	-	-	-	-	-	-	vrai	115506	14336	0	6.83	12.41
k_grz_n-5	-	-	-	-	-	-	vrai	165943	20480	0	9.30	12.34
k_grz_n-6	-	-	-	-	-	-	vrai	308794	36862	0	21.87	11.93
k_grz_n-7	-	-	-	-	-	-	vrai	463423	55294	0	37.98	11.93
k_grz_n-8	-	-	-	-	-	-	vrai	502222	59904	0	42.58	11.92
k_grz_n-9	-	-	-	-	-	-	vrai	850324	101375	0	67.40	11.92
k_grz_n-10	-	-	-	-	-	-	vrai	1930866	222768	0	188.81	11.53
k_grz_n-11	-	-	-	-	-	-	vrai	2499281	288286	0	249.08	11.53
k_grz_n-12	-	-	-	-	-	-	vrai	1818058	209662	0	191.76	11.53
k_grz_n-13	-	-	-	-	-	-	vrai	3182241	366910	0	348.03	11.52

TAB. 2 – *k_grz*

Instance	Jeroslow-Wang						Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_lin_n-1	vrai	5	2	0	0.00	40.00	vrai	5	2	0	0.00	40.00
k_lin_n-2	vrai	2445981	0	0	102.22	0.00	vrai	259533	22162	0	18.89	8.53
k_lin_p-1	faux	3720	464	0	0.11	12.47	faux	571	113	0	0.03	19.78
k_lin_p-2	faux	3301816	372984	0	118.26	11.29	faux	147	0	0	0.01	0
k_lin_p-3	-	-	-	-	-	-	faux	1082712	160799	0	135.91	14.85
k_lin_p-5	-	-	-	-	-	-	faux	67228	0	0	11.78	0

TAB. 3 – *k_lin*

Instance	Jeroslow-Wang						Δ					
	Type	#Branch	#T	#F	Temps	%HR	Type	#Branch	#T	#F	Temps	%HR
k_ph_n-1	vrai	0	1	0	0.00	100	vrai	0	1	0	0.00	100
k_ph_n-2	vrai	15	2	0	0.00	13.33	vrai	15	2	0	0.00	13.33
k_ph_n-3	vrai	181	0	0	0.00	0.00	vrai	101	17	0	0.01	16.83
k_ph_n-4	vrai	785	0	0	0.03	0.00	vrai	1647	240	0	0.11	14.57
k_ph_n-5	vrai	13119	0	0	1.12	0.00	vrai	37407	3044	0	4.22	8.13
k_ph_n-6	vrai	290163	0	0	45.15	0.00	vrai	487594	28375	0	82.33	5.81
k_ph_p-1	faux	3	1	0	0.00	33.33	faux	1	0	0	0.00	0.00
k_ph_p-2	faux	18	1	0	0.00	5.55	faux	6	0	0	0.00	0.00
k_ph_p-3	faux	535	2	0	0.01	0.37	faux	403	68	0	0.03	16.87
k_ph_p-4	faux	67297	677	0	1.00	3.56	faux	57154	6117	0	4.82	10.70

TAB. 4 – *k_ph*

Bibliographie

[Arvind & Biswas 1987] ARVIND V. & BISWAS S. (1987). « An $O(n^2)$ algorithm for the satisfiability problem of a subset of propositional sentences in CNF that includes all Horn sentences ». *Information Processing Letters*, **24**(1), 67–69.

[Aspvall 1980] ASPVALL B. (1980). « Recognizing disguised NR(1) instances of the satisfiability problem ». *Journal of Algorithms*, **1**(1), 97–103.

[Aspvall *et al.* 1979] ASPVALL B., PLASS M. & TARJAN R. (1979). « A linear-time algorithm for testing the truth of certain quantified boolean formulas ». *Information Processing Letters*, **8**(3), 121–123. Erratum: Information Processing Letters 14(4): 195 (1982).

[Audemard & Saïs 2004] AUDEMARD G. & SAÏS L. (2004). « SAT based BDD solver for Quantified Boolean Formulas ». In *Proceedings of the 16th International Conference on Tools with Artificial Intelligence (ICTAI'04)*, p. 82–89, Boca Raton, Florida, USA.

[Audemard & Saïs 2005] AUDEMARD G. & SAÏS L. (2005). « A Symbolic Search Based Approach for Quantified Boolean Formulas ». In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, p. 16–30, St. Andrews, United Kingdom.

[Ayari & Basin 2000] AYARI A. & BASIN D. A. (2000). « Bounded model construction for monadic second-order logics ». In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV'00)*, p. 99–112, Chicago, Illinois, USA.

[BAHIA (Groupe) 1995] BAHIA (GROUPE) (1995). « Etude comparative de trois formalismes en calcul propositionnel ». In *Proceedings of the 5èmes Journées du PRC Intelligence Artificielle*, p. 125–157.

[Balsiger *et al.* 2000] BALSIGER P., HEUERDING A. & SCHWENDIMANN S. (2000). « A benchmark method for the propositional modal logics K, KT, S4 ». *Journal of Automated Reasoning*, **24**(3), 297–317.

[Belleannée *et al.* 1995] BELLEANNÉE C., BENHAMOU B., BOIRON P., CASTAING J., DUBOIS ., FOUQUERÉ C., GÉNISSON R., HÉBRARD J. J., JÉGOU P., LUQUET P., MASSERON M., OXUSOFF L., RAUZY A., SAÏS L., SIEGEL P. & VORC'H R. (1995). « Projet inter-PRC « classes polynomiales » – travaux et résultats ». In *Proceedings of the 5èmes Journées du PRC Intelligence Artificielle*, p. 3–28.

[Benedetti 2005] BENEDETTI M. (2005). « sKizzo: a suite to evaluate and certify QBFs ». In *Proceedings of the 20th International Conference on Automated Deduction (CADE'05)*, p. 369–376, Tallinn, Estonia.

[Benoist 2000] BENOIST E. (2000). « *Génération à délai polynomial pour le problème SAT* ». Thèse d'université, Université de Caen.

[Benoist & Hébrard 1999] BENOIST E. & HÉBRARD J. J. (1999). « *Ordered Formulas* ». Rapport interne 14, Les cahiers du GREYC, Université de Caen.

[Benoist & Hébrard 2003] BENOIST E. & HÉBRARD J. J. (2003). « Recognition of Simple Enlarged Horn Formulas and Simple Extended Horn Formulas ». *Annals of Mathematics and Artificial Intelligence*, **37**(3), 251–272.

[Besnard *et al.* 2002] BESNARD P., SCHAUB T., TOMPITS H. & WOLTRAN S. (2002). « Paraconsistent Reasoning via Quantified Boolean Formulas, I: Axiomatising Signed Systems ». In *Proceedings of the Journées Européennes sur la Logique en Intelligence Artificielle (JELIA'02)*, p. 320–331, Cosenza, Italy.

[Biere 2004] BIERE A. (2004). « Resove and Expand ». In *Proceedings of the 7^{th} International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, p. 238–246, Vancouver, Colombie Britannique, Canada.

[Biere *et al.* 1999a] BIERE A., CIMATTI A., CLARKE E. M., FUJITA M. & ZHU Y. (1999a). « Symbolic model checking using sat procedures instead of BDDs. ». In *Proceedings of the 36^{th} Design Automation Conference (DAC'99)*, p. 317–320, New Orleans, Louisiana, USA.

[Biere *et al.* 1999b] BIERE A., CIMATTI A., CLARKE E. M. & ZHU Y. (1999b). « Symbolic model checking without BDDs. ». In *Proceedings of the 5^{th} International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'99)*, p. 193–207, Amsterdam, The Netherlands.

[Böhm & Speckenmeyer 1996] BÖHM M. & SPECKENMEYER E. (1996). « A fast parallel SAT–solver — efficient workload balancing ». *Annals of Mathematics and Artificial Intelligence*, **17**(3–4), 381– 400.

[Bordeaux 2003] BORDEAUX L. (2003). « *Résolution de problèmes combinatoires modélisés par des contraintes quantifiées* ». Thèse d'université, Université de Nantes.

[Börner *et al.* 2003] BÖRNER F., BULATOV A., KROKHIN A. & JEAVONS P. (2003). « Quantified constraints: algorithms and complexity ». In *Proceedings of the 17^{th} International Conference for Computer Science Logic (CSL'03)*, p. 58–70, Vienna, Austria.

[Boros *et al.* 1994] BOROS E., HAMMER P. L. & SUN X. (1994). « Recognition of q-Horn formulae in linear time ». *Discrete Applied Mathematics*, **55**(1), 1–13.

[Boufkhad *et al.* 1997] BOUFKHAD Y., GRÉGOIRE E., MARQUIS P., MAZURE B. & SAÏS L. (1997). « Tractable cover compilations ». In *Proceedings of the 15^{th} International Joint Conference on Artificial Intelligence (IJCAI'97)*, p. 122–127, Nagoya, Japan.

[Bryant 1986] BRYANT R. E. (1986). « Graph-based algorithms for boolean function manipulation ». *IEEE Transactions on Computers*, **35**(8), 677–692.

[Cadoli *et al.* 2002a] CADOLI M., DONINI F., LIBERATORE P. & SCHAERF M. (2002a). « Preprocessing of intractable problems ». *Information and Computation*, **176**(2), 89–120.

[Cadoli *et al.* 1998] CADOLI M., GIOVANARDI A. & SCHAERF M. (1998). « An algorithm to Evaluate Quantified Boolean Formulae ». In *Proceedings of The 15^{th} National Conference on Artificial Intelligence (AAAI'98)*, p. 262–267, Madison, Wisconsin, USA.

[Cadoli *et al.* 2002b] CADOLI M., GIOVANARDI A., SCHAERF M. & GIOVANARDI M. (2002b). « An algorithm to evaluate quantified boolean formulae and its experimental evaluation ». *Journal of Automated Reasoning*, **28**(2), 101–142.

[Castell 1997] CASTELL T. (1997). « *Consistance et déduction en logique propositionnelle* ». Thèse d'université, Université Paul Sabatier, Toulouse.

[Castellini *et al.* 2003] CASTELLINI C., GIUNCHIGLIA E. & TACCHELLA E. (2003). « SAT-based planning in complex domains: Concurrency, constraints and nondeterminist ». *Journal of Artificial Intelligence*, **147**(1), 85–117.

[Chandru *et al.* 1990] CHANDRU V., COULLARD C. R., HAMMER P., MONTANEZ M. & SUN X. (1990). « On Renamable Horn and Generalized Horn Functions ». *Annals of Mathematics and Artificial Intelligence*, **1**, 33–47.

[Chen 2004] CHEN H. (2004). « Collapsibility and consistency in quantified constraint satisfaction ». In *Proceedings of The 18th National Conference on Artificial Intelligence (AAAI'02)*, p. 155–160, San Jose, California, USA.

[Chen 2005] CHEN H. (2005). « Quantified constraint satisfaction, maximal constraint languages, and symmetric polymorphisms ». In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS'05)*, p. 315–326, Stuttgart, Germany.

[Cimatti *et al.* 1997] CIMATTI A., GIUNCHIGLIA E., GIUNCHIGLIA F. & TRAVERSO P. (1997). « Planning via model checking: a decision procedure for AR ». In *Proceedings of the 4th European Conference on Planning (ECP'97)*, p. 130–142, Toulouse, France.

[Cook 1971] COOK S. (1971). « The complexity of theorem-proving procedures ». In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC'71)*, p. 151–158, Shaker Heights, Ohio, USA.

[Coste-Marquis *et al.* 2002] COSTE-MARQUIS S., FARGIER H., LAND J., LE BERRE D. & MARQUIS P. (2002). « Résolution de formules booléennes quantifiées : problèmes et algorithmes ». In *Actes du 13e Congrès AFRIF-AFIA Reconnaissance des Formes et Intelligence Artificielle*, p. 289–298, Angers, France.

[Coste-Marquis *et al.* 2005a] COSTE-MARQUIS S., LE BERRE D. & LETOMBE F. (2005a). « A Branching Heuristics for Quantified Renamable Horn Formulas ». In *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, p. 393–399, St. Andrews, United Kingdom.

[Coste-Marquis *et al.* 2005b] COSTE-MARQUIS S., LE BERRE D., LETOMBE F. & MARQUIS P. (2005b). « Propositional Fragments for Knowledge Compilation and Quantified Boolean Formulae ». In *Proceedings of The 20th National Conference on Artificial Intelligence (AAAI'05)*, p. 288–293, Pittsburgh, Pensylvania, USA.

[Coste-Marquis & Marquis 1998] COSTE-MARQUIS S. & MARQUIS P. (1998). « Characterizing consistency-based diagnoses ». In *Proceedings of the 5th International Symposium on Artificial Intelligence and Mathematics (AI & Math'98)*, Fort Lauderdale, Florida, USA. (electronic proceedings).

[Coste-Marquis & Marquis 2001] COSTE-MARQUIS S. & MARQUIS P. (2001). « Knowledge compilation for circumscription and closed world reasoning ». *Journal of Logic and Computation*, **11**(4), 579–607.

[Coste-Marquis & Marquis 2004] COSTE-MARQUIS S. & MARQUIS P. (2004). « On Stratified Belief Base Compilation ». *Annals of Mathematics and Artificial Intelligence*, **42**(4), 399–442.

[Crawford & Auton 1993] CRAWFORD J. & AUTON L. (1993). « Experimental results on the crossover point in satisfiability problems ». In *Proceedings of The 11th National Conference on Artificial Intelligence (AAAI'93)*, p. 21–27, Washington, District of Colombia, USA.

[Crawford & Baker 1994] CRAWFORD J. M. & BAKER A. B. (1994). « Experimental results on the application of satisfiability algorithms to scheduling problems ». In *Proceedings of The 12th National Conference on Artificial Intelligence (AAAI'94)*, p. 1092–1097, Seattle, Washington, USA.

[Darwiche 1999] DARWICHE A. (1999). « Compiling devices into decomposable negation normal form ». In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 284–289, Stockholm, Sweden.

[Darwiche 2001] DARWICHE A. (2001). « Decomposable negation normal form ». *Journal of the Association for Computing Machinery*, **48**(4), 608–647.

[Darwiche & Marquis 2002] DARWICHE A. & MARQUIS P. (2002). « A Knowledge Compilation Map ». *Journal of Artificial Intelligence Research*, **17**, 229–264.

[Darwiche & Marquis 2004] DARWICHE A. & MARQUIS P. (2004). « Compiling propositional weighted bases ». *Journal of Artificial Intelligence*, **157**(1–2), 81–113.

[Davis *et al.* 1962] DAVIS M., LOGEMANN G. & LOVELAND D. (1962). « A machine program for theorem-proving ». *Communications of the ACM*, **5**(7), 394–397.

[Davis & Putnam 1960] DAVIS M. & PUTNAM H. (1960). « A computing procedure for quantification theory ». *Journal of the Association for Computing Machinery*, **7**(3), 201–215.

[Del Val 1994] DEL VAL A. (1994). « Tractable databases: how to make propositional unit resolution complete through compilation ». In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, p. 551–561, Bonn, Germany.

[Del Val 2000] DEL VAL A. (2000). « On 2-SAT and Renamable Horn ». In *Proceedings of The 17th National Conference on Artificial Intelligence (AAAI'00)*, p. 279–284, Austin, Texas, USA.

[Dowling & Gallier 1984] DOWLING W. & GALLIER J. (1984). « Linear time algorithms for testing the satisfiability of propositional Horn formulae ». *Journal of Logic Programming*, **1**(3), 267–284.

[Dubois *et al.* 1996] DUBOIS ., ANDRÉ P., BOUFKHAD Y. & CARLIER J. (1996). « Sat vs. unsat ». In *Second DIMACS implementation challenge: cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, p. 415–436.

[Egly *et al.* 2000] EGLY U., EITER T., TOMPITS H. & WOLTRAN S. (2000). « Solving advanced reasoning tasks using Quantified Boolean Formulas ». In *Proceedings of The 17th National Conference on Artificial Intelligence (AAAI'00)*, p. 417–422, Austin, Texas, USA.

[Egly *et al.* 2003] EGLY U., SEIDL M., TOMPITS H., WOLTRAN S. & ZOLDA M. (2003). « Comparing different prenexing strategies for Quantified Boolean Formulas ». In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, p. 214–228, Portofino, Italy.

[Eiter *et al.* 1995] EITER T., KILPELAINEN T. & MANNILA H. (1995). « Recognizing renamable generalized propositional Horn formulas is NP-complete ». *Discrete Applied Mathematics*, **2**(59), 23–32.

[Even *et al.* 1976] EVEN S., ITAI A. & SHAMIR A. (1976). « On the complexity of timetable and integral multi-commodity flow problems ». *SIAM Journal on Comp*, **5**(4), 691–703.

[Fargier *et al.* 2000] FARGIER H., LANG J. & MARQUIS P. (2000). « Propositional Logic and One-stage Decision Making ». In *Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, p. 445–456, Breckenridge, Colorado, USA.

[Feldmann *et al.* 2000] FELDMANN R., MONIEN B. & SCHAMBERGER S. (2000). « A distributed algorithm to evaluate quantified boolean formula ». In *Proceedings of The 17th National Conference on Artificial Intelligence (AAAI'00)*, p. 285–290, Austin, Texas, USA.

[Freemann 1995] FREEMANN J. W. (1995). « *Improvement to Propositional Satisfiability Search Algorithms* ». PhD thesis, University of Pennsylvania.

[Geffner 2004] GEFFNER H. (2004). « Planning graphs and knowledge compilation ». In *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, p. 662–672, Whistler, Colombie Britannique, Canada.

[Génisson & Rauzy 1995] GÉNISSON R. & RAUZY A. (1995). « *Efficient Horn renaming: yet a Davis and Putnam's procedure* ». Rapport interne, LaBRI, Université Bordeaux I.

[Gent & Rowley 2002] GENT I. P. & ROWLEY A. G. D. (2002). « Solving 2-CNF Quantified Boolean Formulae using Variable Assignment and Propagation ». In *Proceedings of the 5th International Conference on Theory and Applications of Satisfiability Testing (SAT'02)*, p. 17–25, Cincinnati, Ohio, USA.

[Gergov & Meinel 1994] GERGOV J. & MEINEL C. (1994). « Efficient analysis and manipulation of OBDDs can be extended to FBDDs ». *IEEE Transactions on Computers*, **43**(10), 1197–1209.

[Ghallab & Escalada-Imaz 1991] GHALLAB M. & ESCALADA-IMAZ G. (1991). « A linear control algorithm for a class of rule-based systems ». *Journal of Logic Programming*, **11**(2), 117–132.

[GhasemZadeh *et al.* 2004] GHASEMZADEH M., KLOTZ V. & MEINEL C. (2004). « *ZQSAT: a QSAT Solver based on Zero-Suppressed Binary Decision Diagrams* ». Rapport interne, FB-IV Informatik, University of Trier.

[Giunchiglia *et al.* 2001a] GIUNCHIGLIA E., NARIZZANO M. & TACCHELLA A. (2001a). « Backjumping for Quantified Boolean Logic Satisfiability ». In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, p. 275–281, Seattle, Washington, USA.

[Giunchiglia *et al.* 2001b] GIUNCHIGLIA E., NARIZZANO M. & TACCHELLA A. (2001b). « QuBE: A system for deciding Quantified Boolean Formulas Satisfiability ». In *Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR'01)*, p. 364–369, Siena, Italy.

[Giunchiglia *et al.* 2002] GIUNCHIGLIA E., NARIZZANO M. & TACCHELLA A. (2002). « Learning for quantified boolean logic satisfiability ». In *Proceedings of The 18th National Conference on Artificial Intelligence (AAAI'02)*, p. 649–654, Edmonton, Alberta, Canada.

[Giunchiglia *et al.* 2003] GIUNCHIGLIA E., NARIZZANO M. & TACCHELLA A. (2003). « Backjumping for quantified boolean logic satisfiability ». *Artificial Intelligence*, **145**(1–2), 99–120.

[Gu *et al.* 1997] GU J., PURDOM P. W., FRANCO J. & WAH B. W. (1997). « Algorithms for Satisfiability (SAT) Problem: A Survey ». In *Satisfiability (SAT) Problem*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, p. 19–152.

[Haken 1985] HAKEN A. (1985). « The intractability of resolution ». *Theoretical Computer Science*, **39**, 297–308.

[Hébrard 1994] HÉBRARD J. J. (1994). « A Linear Algorithm for Renaming a Set of Clauses as a Horn Set ». *Theoretical Computer Science*, **124**(2), 343–350.

[Itai & Makowsky 1982] ITAI A. & MAKOWSKY J. A. (1982). « *On the complexity of Herbrand's theorem* ». Rapport interne, Department of Computer Science, Israel Institute of Technology.

[Jeroslow & Wang 1990] JEROSLOW R. J. & WANG J. (1990). « Solving propositional satisfiability problems ». *Annals of Mathematics and Artificial Intelligence*, **1**, 167–188.

[Karp 1972] KARP R. M. (1972). Reducibility among combinatorial problems. In *Complexity of Computer Computations*, p. 85–103. Plenum Press.

[Karp & Lipton 1980] KARP R. M. & LIPTON R. J. (1980). « Some connections between nonuniform and uniform complexity classes ». In *Proceedings of the 12th Annual ACM Symposium on Theory of Computing (STOC'80)*, p. 302–309, Los Angeles, California, USA.

[Kautz & Selman 1992] KAUTZ H. A. & SELMAN B. (1992). « Planning as satisfiability ». In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, p. 359–363, Vienna, Austria.

[Kim *et al.* 2000] KIM J., WHITTEMORE J., MARQUES-SILVA J. & SAKALLAH K. (2000). « On applying incremental satisfiability to delay fault testing ». In *Proceedings of the 3rd European Conference on Design, Automation and Test in Europe (DATE'00)*, p. 380–384, Paris, France.

[Kleene 1943] KLEENE S. C. (1943). « Recursive predicates and quantifiers ». *Transactions of the American Mathematical Society*, **53**, 41–73.

[Kleine-Büning *et al.* 1995] KLEINE-BÜNING H., KARPINSKI M. & FLÖGEL A. (1995). « Resolution for quantified boolean formulas ». *Information and Computation*, **117**(1), 12–18.

[Kleine-Büning & Lettmann 1999] KLEINE-BÜNING H. & LETTMANN T. (1999). *Propositional Logic: Deduction and Algorithms*. Cambridge University Press.

[Kleine-Büning *et al.* 2003] KLEINE-BÜNING H., SUBRAMANI K. & ZHAO X. (2003). « On boolean models for quantified boolean horn formulas ». In *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, p. 93–104, Portofino, Italy.

[Knuth 1990] KNUTH D. E. (1990). « Nested satisfiability. ». *Acta Informatica*, **28**(1), 1–6.

[Lagasquie-Schiex 1995] LAGASQUIE-SCHIEX M. C. (1995). « *Contribution à l'étude des relations d'inférence non-monotone combinant inférence classique et préférences* ». Thèse d'université, Université Paul Sabatier, Toulouse.

[Lang *et al.* 2003] LANG J., LIBERATORE P. & MARQUIS P. (2003). « Propositional independence - formula-variable independence and forgetting ». *Journal of Artificial Intelligence Research*, **18**, 391–443.

[Larrabee 1992] LARRABEE T. (1992). « Test pattern generation using boolean satisfiability ». *IEEE Transactions on Computer-Aided Design*, p. 4–15.

[Le Berre *et al.* 2005] LE BERRE D., NARRIZANO M., SIMON L. & TACHELLA A. (2005). « The second QBF Solvers Comparative Evaluation ». In *Proceedings of the* 7^{th} *International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, Vancouver, Colombie Britannique, Canada.

[Le Berre *et al.* 2003] LE BERRE D., SIMON L. & TACCHELLA A. (2003). « Challenges in the QBF Arena: the SAT'03 evaluation of QBF solvers ». In *Proceedings of the* 6^{th} *International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, p. 452–467, Portofino, Italy.

[Letombe 2003] LETOMBE F. (2003). « *Algorithmique pour les Formules Booléennes Quantifiées* ». Rapport interne, Centre de Recherche en Informatique de Lens, CNRS FRE 2499.

[Letombe 2004] LETOMBE F. (2004). « *Classes polynomiales pour les Formules Booléennes Quantifiées* ». Rapport interne, Centre de Recherche en Informatique de Lens, CNRS FRE 2499.

[Letombe 2005] LETOMBE F. (2005). « Une heuristique de branchement dirigée vers les formules Horn renommables quantifiées ». In *Proceedings of the* $7^{èmes}$ *Rencontres Jeunes Chercheurs en Intelligence Artificielle (RJCIA'05)*, p. 183–196, Nice, France.

[Letz 2001] LETZ R. (2001). « Advances in Decision Procedures for Quantified Boolean Formulas ». In *Proceedings of the* 1^{st} *International Conference on Quantified Boolean Formulae (QBF'01)*, p. 55–64, Siena, Italy.

[Letz 2002] LETZ R. (2002). « Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas ». In *Proceedings of the* 11^{th} *Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'02)*, p. 160–175, Copenhagen, Denmark.

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

[Li & Anbulagan 1997] LI C. M. & ANBULAGAN (1997). « Heuristics Based on Unit Propagation for Satisfiability Problems ». In *Proceedings of the* 15^{th} *International Joint Conference on Artificial Intelligence (IJCAI'97)*, p. 366–371, Nagoya, Japan.

[Liberatore 2001] LIBERATORE P. (2001). « Monotonic reductions, representative equivalence, and compilation of intractable problems ». *Journal of the Association for Computing Machinery*, **48**(6), 1091–1125.

[Marquis 2000] MARQUIS P. (2000). « *Consequence Finding Algorithms* », In *Handbook on Defeasible Reasoning and Uncertainty Management Systems*, volume 5, chapter 2, p. 41–145. Kluwer Academic Publisher.

[Massacci & Marraro 2000] MASSACCI F. & MARRARO L. (2000). « Logical cryptanalysis as a SAT problem ». *Journal of Automated Reasoning*, **24**(1/2), 165–203.

[Minoux 1988] MINOUX M. (1988). « LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation ». *Information Processing Letter*, **29**(1), 1–12.

[Mostowski 1947] MOSTOWSKI A. (1947). « On definable sets of positive integers ». *Fundamenta Mathematicae*, **34**, 81–112.

[Pan *et al.* 2002] PAN G., SATTLER U. & VARDI M. Y. (2002). « Bdd-based decision procedures for k ». In *Proceedings of the 18th International Conference on Automated Deduction (CADE'02)*, p. 16–30, Copenhagen, Denmark.

[Pan & Vardi 2003] PAN G. & VARDI M. (2003). « Optimizing a BDD-Based Modal Solver ». In *Proceedings of the 19th International Conference on Automated Deduction (CADE'03)*, p. 75–89, Miami Beach, Florida, USA.

[Pan & Vardi 2004] PAN G. & VARDI M. (2004). « Symbolic Decision MakingProcedures for QBF ». In *Proceedings of The 10th International Conference on Principles and Practice of Constraint Programming (CP'04)*, p. 453–467, Toronto, Ontario, Canada.

[Papadimitriou 1994] PAPADIMITRIOU C. H. (1994). *Computational complexity*. Addison–Wesley.

[Prosser 1993] PROSSER P. (1993). « Hybrid algorithms for the constraint satisfaction problem ». *Computational Intelligence*, **9**(3), 268–299.

[Quine 1955] QUINE W. (1955). « A way to simplify truth functions ». *American Mathematical Monthly*, **62**, 627–631.

[Rintanen 1999a] RINTANEN J. (1999a). « Constructing Conditional Plans by a Theorem-Prover ». *Journal of Artificial Intelligence Research*, **10**, 323–352.

[Rintanen 1999b] RINTANEN J. (1999b). « Improvements to the Evaluation of Quantified Boolean Formulae ». In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI'99)*, p. 1192–1197, Stockholm, Sweden.

[Rintanen 2001] RINTANEN J. (2001). « Partial implicit unfolding in the Davis-Putnam procedure for Quantified Boolean Formulae ». In *Proceedings of the 1st International Conference on Quantified Boolean Formulae (QBF'01)*, p. 84–93, Siena, Italy.

[Robinson 1965] ROBINSON J. A. (1965). « A machine-oriented logic based on the resolution principle ». *Journal of the Association for Computing Machinery*, **12**(1), 23–41.

[Schaefer 1978] SCHAEFER T. J. (1978). « The complexity of satisfiability problems ». In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC'78)*, p. 216–226, San Diego, California, USA.

[Schlipf *et al.* 1995] SCHLIPF J., ANNEXSTEIN F., FRANCO J. & SWAMINATHAN R. (1995). « On finding solutions to extended Horn formulas ». *Information Processing Letters*, **54**(3), 133–137.

[Scholl & Becker 2001] SCHOLL C. & BECKER B. (2001). « Checking equivalence for partial implementations ». In *Proceedings of the 38th Design Automation Conference (DAC'01)*, p. 238–243, Las Vegas, Nevada, USA.

[Schrag 1996] SCHRAG R. (1996). « Compilation for critically constrained knowledge bases ». In *Proceedings of The 13th National Conference on Artificial Intelligence (AAAI'96)*, p. 510–515, Portland, Oregon, USA.

[Scutella 1990] SCUTELLA M. G. (1990). « A note on Dowling and Gallier's topdown algorithm for propositional Horn satisfiability ». *Journal of Logic Programming*, **8**(3), 265–273.

[Selman & Kautz 1996] SELMAN B. & KAUTZ H. A. (1996). « Knowledge compilation and theory approximation ». *Journal of the Association for Computing Machinery*, **43**(2), 193–224.

[Sieling & Wegener 1993] SIELING D. & WEGENER I. (1993). « Reduction of OBDDs in linear time ». *Information Processing Letters*, **48**(3), 139–144.

[Stockmeyer 1976] STOCKMEYER L. J. (1976). « The polynomial-time hierarchy ». *Theoretical Computer Science*, **3**(1), 1–22.

[Stockmeyer & Meyer 1973] STOCKMEYER L. J. & MEYER A. R. (1973). « Word problems requiring exponential time ». In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC'73)*, p. 1–9, Austin, Texas, USA.

[Swaminathan & Wagner 1995] SWAMINATHAN R. P. & WAGNER D. K. (1995). « The arborescence-realisation problem ». *Discrete Applied Mathematics*, **59**(3), 267–283.

[Tison 1967] TISON P. (1967). « Generalization of consensus theory and application to the minimization of boolean functions ». *IEEE Transactions on Electronic Computers*, **EC–16**, 446–456.

[Urquhart 1987] URQUHART A. (1987). « Hard examples for resolution ». *Journal of the Association for Computing Machinery*, **34**(1), 209–219.

[Urquhart 1995] URQUHART A. (1995). « The complexity ot propositional proofs ». *The Bulletin of Symbolic Logic*, **1**(4), 425–467.

[Yamasaki & Doshita 1983] YAMASAKI S. & DOSHITA S. (1983). « 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.

[Zanuttini 2002] ZANUTTINI B. (2002). « Approximating propositional knowledge with affine formulas ». In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02)*, p. 287–291, Lyon, France.

[Zhang & Malik 2002] ZHANG L. & MALIK S. (2002). « Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation ». In *Proceedings of The 8th International Conference on Principles and Practice of Constraint Programming (CP'02)*, p. 200–215, Ithaca, New York, USA.

Index

Résumé

Cette thèse est centrée sur le problème QBF de décision de la validité des formules booléennes quantifiées : étant donnée une formule de la forme $\Sigma = \forall y_1 \, \exists x_1 \ldots \forall y_n \, \exists x_n.\phi$ où ϕ est une formule propositionnelle construite sur $\{x_1, y_1, \ldots, x_n, y_n\}$ (la matrice de Σ), il s'agit de déterminer si pour toute affectation d'une valeur de vérité à y_1 dans ϕ, il existe une affectation d'une valeur de vérité à x_1 dans ϕ, ..., pour toute affectation d'une valeur de vérité à y_n dans ϕ, il existe une affectation d'une valeur de vérité à x_n dans ϕ qui rend ϕ valide. Le problème QBF est calculatoirement difficile (**PSPACE**-complet). Il importe donc de mettre en évidence des cas particuliers où sa résolution pratique est envisageable. Dans cette thèse, nous avons considéré des restrictions de QBF portant sur la matrice des instances. Notre objectif principal était double : (1) identifier la complexité de QBF pour des restrictions non considérées jusqu'ici et (2) explorer dans quelle mesure les classes polynomiales pour QBF peuvent être exploitées au sein d'un prouveur QBF général afin d'améliorer son efficacité.

Concernant le premier point, nous avons montré que QBF restreint aux fragments cibles pour la compilation de « connaissances » étudiés dans (Darwiche & Marquis 2002), qui sont traitables pour SAT, reste **PSPACE**-complet et donc intraitable en pratique. Nous avons également mis en évidence le lien étroit qui existe entre notre étude et le problème de compilabilité de QBF.

Concernant le second point, nous avons décrit une nouvelle heuristique de branchement Δ visant à promouvoir la génération de formules Horn renommables quantifiées durant le parcours de l'arbre de recherche effectué par une procédure de type Q-DPLL pour QBF. Les résultats expérimentaux présentés montrent qu'en pratique, hormis notre prouveur **Qbfl**, les prouveurs QBF actuels ne sont pas capables de résoudre facilement les instances Horn quantifiées ou Horn renommables quantifiées ; ceci suffit à justifier l'intérêt de l'approche suivie. Les résultats obtenus montrent également que, muni de l'heuristique Δ, **Qbfl** améliore significativement ses performances sur certains types d'instances, même s'il obtient des résultats moyens en général, comparé aux autres prouveurs QBF actuels.

Mots-clés : formules booléennes quantifiées, logique propositionnelle, compilation de « connaissances », restrictions traitables, complexité, heuristiques de branchement.

Abstract

This thesis is centered on QBF, the validity problem for quantified Boolean formulae: given a formula of the form $\Sigma = \forall y_1 \, \exists x_1 \ldots \forall y_n \, \exists x_n.\phi$ where ϕ is a propositional formula built on $\{x_1, y_1, \ldots, x_n, y_n\}$ (the matrix of Σ), is it the case that for each assignment of a truth value to y_1 in ϕ, there exists an assignment of a truth value to x_1 in ϕ, ..., for each assignment of a truth value to y_n in ϕ, there exists an assignment of a truth value to x_n in ϕ that makes ϕ valid ? Since QBF is computationally hard (**PSPACE**-complete), it is important to point out some specific cases for which the practical solving of QBF could be achieved. In this thesis, we have considered some restrictions of QBF based on the matrices of instances. Our main purpose was (1) to identify the complexity of QBF for some restrictions not considered so far and (2) to explore how to take advantage of polynomial classes for QBF within a general QBF solver in order to increase its efficiency.

As to the first point, we have shown that QBF, when restricted to the target fragments for knowledge compilation studied in (Darwiche & Marquis 2002), remain typically **PSPACE**-complete. We have shown a close connexion between this study and the compilability issue for QBF.

As to the second point, we have presented a new branching heuristics Δ which aims at promoting the generation of quantified renamable Horn formulae into the search-tree developed by a Q-DPLL procedure for QBF. We have obtained experimental results showing that, in practice, state-of-the-art QBF solvers, except our solver **Qbfl**, are unable to solve quantified Horn instances or quantified renamable Horn instances of medium size. This observation is sufficient to show the interest of our approach. Our experiments have also shown the heuristics Δ to improve the efficiency of **Qbfl**, even if this solver does not appear as one of the best QBF solvers at this time.

Keywords: quantified Boolean formulae, propositional logic, knowledge compilation, tractable restrictions, complexity, branching heuristics.