Une mise sous forme prénexe préservant les résultats intermédiaires pour les formules booléennes quantifiées

Benoit da Mota, Igor Stéphan, Pascal Nicolas

Une mise sous forme prénexe préservant les résultats intermédiaires pour les formules booléennes quantifiées

Benoit Da Mota, Igor Stéphan, Pascal Nicolas

LERIA, Université d'Angers
2, boulevard Lavoisier, 49045, Angers Cedex 01
{damota,stephan,pn}@info.univ-angers.fr

Résumé : La plupart des procédures pour résoudre le problème de validité des formules booléennes quantifiées prennent en entrée seulement des formules sous forme normale conjonctive. Mais, il est rarement naturel d'exprimer un problème directement sous cette forme et il est plus courant d'utiliser des variables existentielles pour représenter des résultats intermédiaires. Or, lors de la mise sous forme prénexe, l'équivalence est exprimée en fonction d'autres opérateurs logiques et les variables intermédiares sont multipliées. Dans ce travail, nous mettons en évidence des équivalences logiques qui permettent aux résultats intermédiaires de traverser les équivalences. Les résultats expérimentaux montrent qu'utiliser ces équivalences logiques avant de transformer la formule sous forme prénexe améliore le temps de résolution par les différentes procédures.

Mots-clés : Formules booléennes quantifiées, validité, forme normale conjonctive, vérification formelle de circuits logiques.

1 Introduction

Le problème de validité pour les formules booléennes quantifiées (QBF) est une généralisation du problème de satisfiabilité pour les formules booléennes. Tandis que décider de la satisfiabilité des formules booléennes est NP-complet, décider de la validité des QBF est PSPACE-complet. C'est le prix à payer pour une représentation plus concise pour de très nombreuses classes de formules. Une multitude d'importants problèmes de décision parmi des champs très divers ont des transformations polynomiales vers le problème de validité des QBF.

La plupart des procédures actuelles pour décider des QBF nécessitent d'avoir en entrée une formule sous forme normale conjonctive (Giunchiglia *et al.*, 2004; Biere, 2004; Benedetti, 2005). Or, les problèmes sont rarement exprimés directement sous cette forme. Il est plus naturel de les représenter en utilisant la richesse du langage et donc à l'aide d'opérateurs plus expressifs (l'implication, l'équivalence et le ou exclusif) et avec des quantificateurs à l'intérieur des formules. La mise sous forme normale conjonctive nécessite que la formule soit sous forme prénexe, c'est-à-dire avec tous les

quantificateurs devant la formule. L'ensemble des transformations précédant la résolution est décrit dans la figure 1.

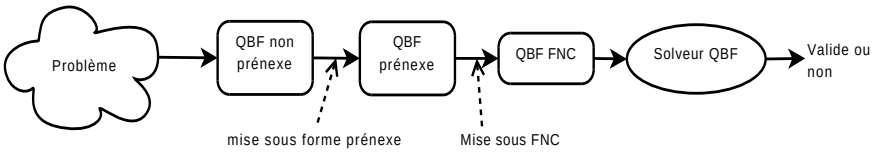

FIG. 1 – Étapes menant à la décision d'un problème via les QBF.

La mise sous forme normale conjonctive a été largement étudiée (Plaisted & Greenbaum, 1986; de la Tour, 1992; Egly, 1996), car cette forme normale est utilisée pour la satisfiabilité des formules booléennes (non quantifiées). Il n'existe pas une unique forme prénexe associée à une QBF et selon la stratégie choisie pour l'ordre d'extraction des quantificateurs, le temps de résolution peut être fortement influencé (Egly *et al.*, 2003). Mais aucune règle n'est fournie pour extraire les quantificateurs de formules booléennes quantifiées contenant des équivalences et des ou exclusifs. Le seul choix possible est d'exprimer ces opérateurs en fonction des opérateurs pour lesquels nous connaissons des règles. Nous verrons que ce choix a des conséquences sur la taille de la formule et sur le nombre de variables. Nous nous intéressons dans la mise sous forme prénexe à l'étape qui consiste à ré-écrire une formule en une forme logiquement équivalente sans équivalence ni ou exclusif. Nous nous intéressons tout particulièrement au traitement des variables existentielles qui représentent un résultat intermédiaire et qui sont introduites pour faciliter l'écriture ou factoriser des sous-formules.

Pour la partie expérimentale, nous utilisons des QBF codant la vérification formelle de circuits logiques et plus particulièrement l'additionneur n-bits, qui illustre en pratique l'intérêt de notre proposition.

2 Préliminaires

L'ensemble des valeurs booléennes **v** et **f** est noté $BOOL$. L'ensemble des variables (propositionnelles ou booléennes) est noté $\mathcal{V}$. Les symboles $\top$ et $\bot$ sont les constantes booléennes. Le symbole $\wedge$ est utilisé pour la conjonction, $\vee$ pour la disjonction, $\neg$ pour la négation, $\rightarrow$ pour l'implication, $\leftrightarrow$ pour l'équivalence et $\oplus$ pour le ou exclusif. L'ensemble des opérateurs $\{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$ est noté $\mathcal{O}$. Un littéral est une variable ou la négation de celle-ci. L'ensemble des littéraux est noté $\mathcal{L}$. L'ensemble $PROP$ des formules propositionnelles est défini inductivement ainsi : tout symbole (constante ou variable) propositionnel est élément de $PROP$; si F est élément de $PROP$ alors $\neg F$ est élément de $PROP$; si F et G sont éléments de $PROP$ et $\circ$ est élément de $\mathcal{O}$ alors $(F \circ G)$ est élément de $PROP$. Le symbole $\exists$ est utilisé pour la quantification existentielle et $\forall$ pour la quantification universelle (q est utilisé pour noter un quantificateur quelconque). L'ensemble QBF des formules booléennes quantifiées est défini inductivement ainsi : si F est un élément de $PROP$ alors c'est un élément de QBF ; si F est un élément de QBF et x est une variable alors $(\exists x\ F)$ et $(\forall x\ F)$ sont des éléments

de QBF. Par convention, des quantificateurs différents lient des variables différentes. L'ensemble des variables d'une formule F est noté $\mathcal{V}(F)$. Une variable x est libre si et seulement si elle n'apparait pas sous la portée d'un quantificateur $\exists x$ ou $\forall x$. L'ensemble des variables libres d'une formule F est noté $\mathcal{L}(F)$. Une substitution est une fonction de l'ensemble des variables dans l'ensemble des formules (quantifiées ou non). Nous définissons la substitution de x par F dans G, notée $G[x \leftarrow F]$, comme étant la formule obtenue de G en remplaçant toutes les occurrences de la variable x par la formule F. Un lieur est une chaîne de caractère $q_1 x_1 \dots q_n x_n$ avec $x_1, \dots, x_n$ des variables distinctes et $q_1 \dots q_n$ des quantificateurs. La fonction q de l'ensemble des variables d'un lieur Q vers $\{\exists, \forall\}$ associe à une variable son quantificateur dans le lieur. Une QBF QF est en forme prénexe si F est une formule booléenne, appelée matrice, et sous forme normale conjonctive (FNC) si F est une formule booléenne en forme normale conjonctive (i.e. une conjonction de disjonctions de littéraux).

La sémantique des symboles booléens est définie de manière habituelle. Une valuation est une fonction de l'ensemble des variables dans $BOOL$; elle satisfait une formule si appliquée à celle-ci elle vaut **v** sinon elle la falsifie. La satisfaction propositionnelle est notée $\models$ et l'équivalence logique $\equiv$. La sémantique des quantificateurs est la suivante : pour toute variable x et QBF F,

$$(\exists x\ F) = (F[x \leftarrow \top] \vee F[x \leftarrow \bot])$$
$$(\forall x\ F) = (F[x \leftarrow \top] \wedge F[x \leftarrow \bot])$$

Une QBF F est valide si $F \equiv \top$.

Enfin, rappelons que le problème (SAT) consistant à décider si une formule booléenne est satisfiable ou non est le problème canonique de la classe NP-complet. De son côté, le problème (QBF) consistant à décider si une formule booléenne quantifiée est valide ou non est le problème canonique de la classe PSPACE-complet.

3 Mise sous forme prénexe et équivalences

3.1 Motivations

Dans (Egly *et al.*, 2003), les auteurs définissent des stratégies pour la mise sous forme prénexe selon l'ordre d'extraction des quantificateurs et montrent expérimentalement que cela peut avoir une forte influence sur le temps de résolution nécessaire aux différentes procédures. Cependant, il n'existe pas de règle pour extraire des quantificateurs de l'équivalence ou du ou exclusif.

D'une manière générale, la mise sous forme prénexe se décompose en trois étapes :

1. suppression des équivalences et des ou exclusifs,
2. renommage des variables afin que des quantificateurs différents lient des variables différentes,
3. extraction des quantificateurs.

L'étape 1 se résume à remplacer toutes les occurrences de l'équivalence et du ou exclusif grâce aux équivalences logiques suivantes :

1) $(F \leftrightarrow G) \equiv ((G \rightarrow F) \wedge (F \rightarrow G))$ 2) $(F \oplus G) \equiv ((G \vee F) \wedge (\neg F \vee \neg G))$

Nous remarquons que les formules F et G sont recopiées deux fois. Nous rappelons les équivalences logiques utilisées dans (Egly *et al.*, 2003) qui permettent de déplacer les

quantificateurs et nous complétons avec celles pour l'implication (règles 13 à 16) qui sont facilement déduites des précédentes. Soit F, G et H des QBF et x une variable booléenne ($x \notin \mathcal{L}(H)$), alors :

3) $\exists x(\neg F) \equiv \neg(\forall x\ F)$.
4) $\forall x(\neg F) \equiv \neg(\exists x\ F)$.
5) $\forall x\ F \equiv F$, si $x \notin \mathcal{L}(F)$.
6) $\exists x\ F \equiv F$, si $x \notin \mathcal{L}(F)$.
7) $\forall x(F \wedge H) \equiv (\forall x\ F) \wedge H$,
8) $\forall x(F \vee H) \equiv (\forall x\ F) \vee H$,
9) $\exists x(F \wedge H) \equiv (\exists x\ F) \wedge H$,
10) $\exists x(F \vee H) \equiv (\exists x\ F) \vee H$,
11) $\forall x(F \wedge G) \equiv (\forall x\ F) \wedge (\forall x\ G)$.
12) $\exists x(F \vee G) \equiv (\exists x\ F) \vee (\exists x\ G)$.
13) $\forall x(F \rightarrow H) \equiv (\exists x\ F) \rightarrow H$,
14) $\exists x(F \rightarrow H) \equiv (\forall x\ F) \rightarrow H$,
15) $\forall x(H \rightarrow G) \equiv H \rightarrow (\forall x\ G)$,
16) $\exists x(H \rightarrow G) \equiv H \rightarrow (\exists x\ G)$,

L'extraction des quantificateurs est un processus qui consiste à appliquer ces équivalences de la droite vers la gauche jusqu'à obtenir une QBF sous forme prénexe. La mise sous forme prénexe conserve la validité mais ne garantit ni de garder la taille de la formule ni l'espace de recherche associé. Alors que l'équivalence représente le "si et seulement si" du langage naturel, le ou exclusif représente sa négation. Le pouvoir d'expression de l'équivalence s'étend bien au delà de la simple équivalence logique entre $(F \leftrightarrow G)$ et $((G \rightarrow F) \wedge (F \rightarrow G))$: un quantificateur présent dans F et G, de part les équivalences logiques 13 à 16, sera extrait aussi bien sous sa forme universelle que sous sa forme existentielle, et ce quelque soit sa forme initiale. Si cela n'a aucun impact vis-à-vis de la validité, il n'en est pas de même vis-à-vis du calcul. Nous montrons dans les exemples suivants qu'à la fin des trois étapes de la mise sous forme prénexe, l'augmentation de la taille de la formule n'est pas le seul problème, même pour une formule très simple.

Soit a une variable booléenne, ϕ, ϕ_1 et ϕ_2 des formules booléennes quantifiées telles que $\phi = (\phi_1 \leftrightarrow \exists a(\phi_2))$ et $a \notin \mathcal{L}(\phi_1)$. Dans un premier temps il faut exprimer l'équivalence à l'aide de la conjonction et de l'implication : $\phi \stackrel{1}{\equiv} ((\phi_1 \rightarrow \exists a(\phi_2)) \wedge (\exists a(\phi_2) \rightarrow \phi_1))$. Les formules ϕ_1 et ϕ_2 ont été dupliquées. Ensuite, il faut extraire les quantificateurs des implications : $\phi \stackrel{16,13}{\equiv} (\exists\ a(\phi_1 \rightarrow \phi_2) \wedge \forall\ a(\phi_2 \rightarrow \phi_1))$. Il faut renommer une des variables a afin d'extraire les quantificateurs de la conjonction. Soit b une nouvelle variable booléenne et $\phi'_2 = \phi_2[a \leftarrow b]$ alors : $\phi \stackrel{9,7}{\equiv} \exists a\ \forall b((\phi_1 \rightarrow \phi_2) \wedge (\phi'_2 \rightarrow \phi_1))$. L'ordre d'extraction des quantificateurs aurait pu être inversé, nous aurions obtenu : $\phi \stackrel{7,9}{\equiv} \forall b\ \exists a((\phi_1 \rightarrow \phi_2) \wedge (\phi'_2 \rightarrow \phi_1))$. Dans le cas général, si F est une formule booléenne quantifiée, $\forall y\ \exists x(F) \not\equiv \exists x\ \forall y(F)$. Le fait de savoir que les quantificateurs peuvent s'inverser contrairement au cas général est une information importante qui pourrait être exploitée par les procédures de décision pour le problème QBF. Les variables a et b représentent la même variable de la formule non prénexe mais rien dans la formule prénexe ne semble les mettre en relation.

Nous envisageons le cas où un quantificateur doit traverser plusieurs équivalences. Nous montrons dans l'exemple suivant, que le choix de la position des parenthèses peut influer sur la taille de la formule mise sous forme prénexe. Soit a une variable booléenne, ϕ_d, ϕ_g, ϕ_1, ϕ_2 et ϕ_3 des formules booléennes quantifiées telles que $\phi_g = ((\phi_1 \leftrightarrow \phi_2) \leftrightarrow (\exists a\ \phi_3))$, $\phi_d = (\phi_1 \leftrightarrow (\phi_2 \leftrightarrow (\exists a\ \phi_3)))$ avec $a \notin \mathcal{L}(\phi_1)$ et $a \notin \mathcal{L}(\phi_2)$ alors $\phi_d \equiv \phi_g$ par associativité de l'équivalence logique. Nous mettons ϕ_g sous forme prénexe :

$$\begin{array}{lll}
\phi_g & \overset{1}{\equiv} & (((\phi_1 \leftrightarrow \phi_2) \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow (\phi_1 \leftrightarrow \phi_2))) \\
\phi_g & \overset{16,13,9,7}{\equiv} & \exists a \forall b\ (((\phi_1 \leftrightarrow \phi_2) \rightarrow \phi_3)\ \wedge (\phi_3[a \leftarrow b] \rightarrow (\phi_1 \leftrightarrow \phi_2)))
\end{array}$$

puis ϕ_d :

$$\begin{array}{lll}
\phi_d & \overset{1}{\equiv} & (\phi_1 \leftrightarrow ((\phi_2 \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow \phi_2))) \\
\phi_d & \overset{1}{\equiv} & ((\phi_1 \rightarrow ((\phi_2 \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow \phi_2))) \wedge \\
 & & (((\phi_2 \rightarrow (\exists a\ \phi_3)) \wedge ((\exists a\ \phi_3) \rightarrow \phi_2)) \rightarrow \phi_1)) \\
\phi_d & \overset{13,14,15,16,9,7}{\equiv} & \exists a \exists c \forall b \forall d((\phi_1 \rightarrow ((\phi_2 \rightarrow \phi_3) \wedge (\phi_3[a \leftarrow b] \rightarrow \phi_2))) \wedge \\
 & & (((\phi_2 \rightarrow \phi_3[a \leftarrow d]) \wedge (\phi_3[a \leftarrow c] \rightarrow \phi_2)) \rightarrow \phi_1))
\end{array}$$

Les formules ϕ_g et ϕ_d, bien qu'équivalentes, n'ont pas du tout la même taille ni le même nombre de variables une fois mises sous forme prénexe. Maintenant, si la formule considérée est $(\phi_1 \leftrightarrow (\phi_2 \leftrightarrow ...(\phi_n \leftrightarrow (\exists a\ \phi_{n+1}))))$, alors 2^n variables seront créées dont la moitié sera quantifiée universellement. Les formules ϕ_n et ϕ_{n+1} seront recopiées 2^n fois, la formule ϕ_{n-1} sera recopiée 2^{n-1} fois, et ainsi de suite, jusqu'à ϕ_1 qui sera recopiée 2 fois. La taille de la formule et le nombre de variable croissent exponentiellement avec le nombre d'équivalences traversées. Si chacune des ϕ_k possède un quantificateur à extraire, il sera impossible d'éviter le pire cas.

3.2 Mise sous forme prénexe et résultats intermédiaires

Il est habituel en programmation et en spécification QBF d'utiliser une variable existentiellement quantifiée pour représenter un résultat intermédiaire soit pour ne pas répéter une sous formule soit pour la lisibilité soit enfin par construction. Formellement, par extension d'un résultat propositionnel classique (Tseitin, 1970), nous nous intéressons au motif $\exists x((x \leftrightarrow F) \wedge G)$, x variable intermédiaire absente de F, qui est équivalent à $G[x \leftarrow F]$. Pour les procédures QBF actuelles, les variables intermédiaires sont traitées comme des variables du problème alors qu'il suffirait pour s'en abstraire de faire un remplacement syntaxique. Nous proposons de garder ces variables intermédiaires car elles peuvent aussi avoir un intérêt opérationnel, tout en évitant les recopies et l'explosion du nombre de variables. Pour ce faire nous exhibons un ensemble d'équivalences logiques permettant d'extraire le quantificateur d'un résultat intermédiaire. L'objectif est de ne créer aucune nouvelle variable et de ne pas recopier les sous formules.

Théorème 1
Soit F et G des formules booléennes quantifiées et x une variable représentant le résultat intermédiaire F, avec $x \notin \mathcal{V}(F)$, alors les équivalences logiques suivantes sont vraies :

1) $(\exists x((F \leftrightarrow x) \wedge G)) \equiv (\forall x((F \leftrightarrow x) \rightarrow G))$
2) $(\exists x((F \leftrightarrow x) \rightarrow G)) \equiv \top$
3) $(\forall x((F \leftrightarrow x) \wedge G)) \equiv \bot$

et pour une formule booléenne quantifiée $H = (Qx((F \leftrightarrow x)\ o\ G))$ avec $Q \in \{\exists, \forall\}$ et $o \in \{\wedge, \vee, \rightarrow\}$ il est possible de mettre H sous la forme d'une des trois équivalences.

Notre proposition s'appuie sur le théorème suivant qui permet d'extraire la définition de la variable intermédiaire lorsqu'elle est dans une sous formule d'une équivalence (ou d'un ou exclusif).

Théorème 2
Soit F, G et H des formules booléennes quantifiées et x une variable booléenne qui représente le résultat intermédiaire F, avec $x \notin \mathcal{L}(H)$, $x \notin \mathcal{V}(F)$ et $o \in \{\oplus, \leftrightarrow\}$, alors $(H\ o\ (\exists x((F \leftrightarrow x) \wedge G))) \equiv (\exists x((F \leftrightarrow x) \wedge (H\ o\ G)))$.

L'objectif est atteint : le quantificateur qui porte sur x est extrait de l'équivalence, aucune variable n'est créée, aucune sous formule n'est dupliquée. Il est possible de choisir comment sera quantifié x à l'aide du théorème 1, rien ne nous oblige à garder le même quantificateur. De même, pour extraire un quantificateur universel, il faut appliquer l'équivalence du théorème 1 sur la partie gauche de l'équivalence.

3.3 Vérification formelle de circuits : l'additionneur n-bits

Lors de la conception de circuits logiques, le but est de construire un circuit qui corresponde au comportement souhaité. C'est à dire que pour chaque jeu d'entrées possible, la sortie attendue est obtenue. Le plus souvent le modèle booléen du circuit diffère grandement du circuit conçu par l'ingénieur pour des raisons pratiques (encombrement, prix, intégration, etc...). La vérification formelle de circuit est un des problèmes qui s'expriment à l'aide de formules de la logique monadique. La construction de modèles bornés (Bounded Model Construction), est une méthode pour résoudre des problèmes de validité de formules de la logique monadique en les transformant en QBF et en cherchant la validité des formules obtenues. Parmi les circuits, la vérification de l'additionneur n-bits est devenu un problème classique dans sa version QBF et sa description complète peut être trouvée dans (Ayari *et al.*, 1999; Ayari & Basin, 2002). La vérification de l'additionneur consiste à vérifier l'équivalence en logique monadique entre la structure du circuit et le comportement souhaité (n est la taille de l'additionneur, A et B sont les deux n-bits à additionner, S le résultat de l'addition, c_i la retenue en entrée et c_o la retenue en sortie) :

$$\phi = \forall n \forall A \forall B \forall S \forall c_i \forall c_o (add_{struc}(n, A, B, S, c_i, c_o) \leftrightarrow add_{comp}(n, A, B, S, c_i, c_o))$$

Les formules add_{struc} et add_{comp} contiennent des variables existentielles, qui représentent des résultats intermédiaires, entre autres les retenues. Grâce aux théorèmes 1 et 2, nous avons le choix d'extraire les variables intermédiaires soit en existentielles, soit en universelles. Dans les deux cas nous pouvons obtenir (en sortant d'abord les quantificateurs universels puis existentiels) une alternance des quantificateurs de type $\forall\exists$ et conserver cette alternance après la mise sous FNC.

Nous donnons un exemple pour l'additionneur n-bits avec $n = 1$. La QBF ϕ_1 code la vérification de l'additionneur pour $n = 1$ avec les fonctions booléennes suivantes : $C_1(x) = (c_i \leftrightarrow x)$, $C_2(x) = (c_o \leftrightarrow x)$, $C_3(x) = (x \leftrightarrow (A_0 \oplus B_0))$, $C_4(x) = (x \leftrightarrow (A_0 \wedge B_0))$, $C_5(x, y, z) = (x \leftrightarrow (y \wedge z))$, $F_1(x, y) = (S_0 \leftrightarrow (x \oplus y))$, $F_2(x, y, z) = (x \leftrightarrow (y \vee z))$, $F_3(x, y) = ((((A_0 \wedge B_0) \vee (A_0 \wedge x)) \vee (B_0 \wedge x)) \leftrightarrow y)$, et $F_4(x) = (((A_0 \leftrightarrow B_0) \leftrightarrow S_0) \leftrightarrow x)$. Les fonctions C_k, $1 \leq k \leq 5$ représentent les motifs extraits de la QBF initiale ; les fonctions C_3, C_4 et C_5 représentent les définitions des variables intermédiaires x_3, x_4 et x_5 ; les fonctions F_k, $1 \leq k \leq 4$, représentent le cœur de la spécification de l'additionneur n-bits.

$\phi_1 = \forall c_i \forall c_o \forall A_0 \forall B_0 \forall S_0$
$[\exists x_1 \exists x_2 (((C_1(x_1) \wedge C_2(x_2)) \wedge$
$(\exists x_3 \exists x_4 \exists x_5 (C_3(x_3) \wedge (F_1(x_3, x_1) \wedge (C_4(x_4) \wedge (C_5(x_5, x_1, x_3) \wedge F_2(x_2, x_5, x_4)))))))))]$
$\leftrightarrow [\exists x_6 \exists x_7 ((C_1(x_6) \wedge C_2(x_7)) \wedge (F_3(x_6, x_7) \wedge F_4(x_6)))]$

La QBF ci-dessous ϕ_i est la QBF initiale ϕ_1 mise sous forme prénexe selon l'algorithme classique (les variables y_k sont issues des variables x_k par recopie des sous-formules de l'équivalence).

$\phi_i = \forall c_i \forall c_o \forall A_0 \forall B_0 \forall S_0 \forall y_1 \forall y_2 \forall y_3 \forall y_4 \forall y_5 \forall y_6 \forall y_7 \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 \exists x_6 \exists x_7$
$[C_1(y_1) \wedge C_2(y_2) \wedge C_3(y_3) \wedge (F_1(y_3, y_1) \wedge C_4(y_4) \wedge C_5(y_5, y_1, y_3) \wedge F_2(y_2, y_5, y_4)]$
$\rightarrow [C_1(x_6) \wedge C_2(x_7) \wedge F_3(x_6, x_7) \wedge F_4(x_6)]] \wedge$
$[C_1(y_6) \wedge C_2(y_7) \wedge F_3(y_6, y_7) \wedge F_4(y_6)]$
$\rightarrow [C_1(x_1) \wedge C_2(x_2) \wedge C_3(x_3) \wedge F_1(x_3, x_1) \wedge C_4(x_4) \wedge C_5(x_5, x_1, x_3) \wedge F_2(x_2, x_5, x_4)]]$

L'ordre des variables influe grandement sur l'efficaté des méthodes de résolution (Egly *et al.*, 2003) ; dans la mise sous forme prénexe de ϕ_1 en ϕ_i, l'ordre des variables n'est contraint que par le respect de l'ordre partiel qui nécessite que les quantificateurs universels des variables initiales du problème doivent précéder les quantificateurs existentiels.

La QBF ci-dessous ϕ_t est la QBF initiale ϕ_1 transformée à l'aide de nos équivalences logiques puis mise sous forme prénexe selon l'algorithme classique.

$\phi_t = \forall c_i \forall c_o \forall A_0 \forall B_0 \forall S_0 \exists x_1 \exists x_2 \exists x_3 \exists x_4 \exists x_5 \exists x_6 \exists x_7$
$[C_1(x_1) \wedge C_2(x_2) \wedge C_3(x_3) \wedge C_4(x_4) \wedge C_5(x_5, x_1, x_3) \wedge C_1(x_6) \wedge C_2(x_7)] \wedge$
$[[F_1(x_3, x_1) \wedge F_2(x_2, x_5, x_4)] \leftrightarrow [F_3(x_6, x_7) \wedge F_4(x_6)]]$

La matrice de la QBF obtenue est partagée en deux parties : la définition des variables intermédiaires suivie de l'exploitation de ces variables intermédiaires dans le cœur de l'équivalence.

4 Résultats expérimentaux

Nous avons développé un algorithme en Prolog, qui permet de générer les instances de l'additionneur, de chercher le motif $(H \leftrightarrow (\exists x((F \leftrightarrow x) \wedge G)))$ et de le substituer par $(\exists x((F \leftrightarrow x) \wedge (H \leftrightarrow G)))$, puis d'effectuer la mise sous FNC. La recherche du motif est appliquée jusqu'à l'obtention d'un point fixe. Le tableau ci-dessous montre le nombre de quantificateurs existentiels et universels, pour différentes valeurs de n, de la FNC de l'additionneur n-bits sans et avec notre transformation (NQ_i est le nombre de quantificateurs de la FNC de la QBF initiale, $NQ_i^\forall$ le nombre de quantificateurs universels de la FNC de la QBF initiale, N_t le nombre de quantificateurs de la FNC de la QBF transformée, $N_t^\forall$ le nombre de quantificateurs universels de la FNC de la QBF transformée et NC_t le nombre de clauses de la FNC de la QBF transformée).

n	4	5	6	7	8	9	10	11	12	13
NQ_i	281	346	411	476	541	606	671	736	801	866
$NQ_i^\forall$	36	44	52	60	68	76	84	92	100	108
NQ_t	147	181	215	249	283	317	351	385	419	453
$NQ_t^\forall$	14	17	20	23	26	29	32	35	38	41
NC_t	383	472	561	650	739	828	917	1006	1095	1184
n	14	15	16	17	18	19	20	21	22	23
NQ_i	931	996	1061	1126	1191	1256	1321	1386	1451	1516
$NQ_i^\forall$	116	124	132	140	148	156	164	172	180	188
NQ_t	487	521	555	589	623	657	691	725	759	793
$NQ_t^\forall$	44	47	50	53	56	59	62	65	68	71
NC_t	1273	1362	1451	1540	1629	1718	1807	1896	1985	2074

Les motifs sont extraits dans la version existentielle. La mise sous forme prénexe appliquée ne cherche pas une forme optimale pour l'ordre des quantificateurs (ce qui demanderait une étude supplémentaire). La mise sous FNC est faite par introduction de variables intermédiaires quantifiées existentiellement qui ne fait croître que polynomialement la taille de la formule. Le nombre de clauses de la FNC de la QBF initiale est exactement le double du nombre de clauses de la FNC de la QBF transformée.

Les tests ont été effectués sur un Intel(R) Pentium(R) 4 à 2.80GHz avec 600Mo de mémoire vive disponible. Les résultats sont résumés dans la table 1, les temps sont en secondes, un T indique que la procédure n'a pas pu résoudre l'instance en moins de 3600 secondes, un M indique un dépassement de la mémoire disponible. Les temps d'exécution ne prennent pas en compte le temps de recherche et la substitution du motif. Les colonnes où figure la mention "sans" sont les résultats pour une formule sans recherche de motif. Les colonnes où figure un $\exists$ (resp. $\forall$) sont les résultats pour une formule où les variables intermédiaires sont extraites existentiellement (resp. universellement).

Les tests ont été effectués dans un premier temps avec les réglages par défaut de chaque procédure. Nous utilisons différentes procédures : sKizzo v0.8.2-beta (Benedetti, 2005) qui intègre skolémisation symbolique et appel à une procédure SAT, Quantor 3.0 (Biere, 2004) qui combine Q-résolution (Büning *et al.*, 1995) et expansion, QuBE-BJ1.2 (Giunchiglia *et al.*, 2004) qui étend aux QBF la procédure de Davis-Logemann-Loveland (Davis *et al.*, 1962) et QuBE6.3 une version plus récente qui utilise un préprocesseur (Bubeck & Büning, 2007) intégrant la Q-résolution. QuBE n'a pas réussi à résoudre le problème avec les QBF initiales pour $n > 3$.

Les résultats de la table 1 montrent que les QBF transformées obtiennent presque toujours de meilleurs résultats avec les procédures testées. La procédure sKizzo est la seule pour laquelle les résultats sont moins tranchés. Pour $n < 12$ l'amélioration est sensible, puis les performances se dégradent. Le traitement effectué sur les équivalences handicape sKizzo sur les instances plus grandes. En observant la trace de sKizzo, nous remarquons que la Q-résolution supprime des variables représentant des résultats intermédiaires. QuBE6.3 ayant de moins bons résultats que QuBE-BJ1.2, nous nous posons des questions sur l'impact de la Q-résolution sur ce problème. Nous avons effectué des tests en désactivant la Q-résolution de sKizzo. Que ce soit avec les QBF initiales ou les QBF transformées, désactiver la Q-résolution permet d'obtenir de meilleurs temps. Avec les QBF transformées le gain est significatif. Nous avons aussi essayé d'extraire les résultats intermédiaires sous une forme d'universelle, mais la mise sous FNC introduit des variables existentielles dépendant alors de bien plus de variables universelles. Les résultats sont moins bons que pour le codage original. Pour ce type de problème il faudrait utiliser une procédure pouvant raisonner sur des formules non FNC.

Pour la vérification formelle de circuits logiques, la Q-résolution semble travailler à l'encontre des autres méthodes. Une explication possible est que la résolution sur les variables intermédiaires (du codage ou de la mise sous FNC) enlève des possibilités aux procédures d'élaguer l'espace de recherche. Les mécanismes d'apprentissage retiennent des règles moins générales, le parcours de l'espace de recherche est plus long. Avec les QBF transformées, la Q-résolution a un impact très important sur les grandes instances, à tel point que les QBF initiales conviennent mieux à sKizzo. Les QBF transformées semblent être plus denses, augmentant l'importance des variables intermédiaires et la possibilité de réduire l'espace de recherche en raisonnant sur ces variables.

5 Conclusion

Dans cet article, nous proposons différentes équivalences qui permettent de traiter les résultats intermédiaires contenus dans des équivalences ou des ou exclusifs pour les formules booléennes

ϕ	sKizzo avec Q-résolution		sKizzo sans Q-résolution			QuBE 6.3	QuBE BJ1.2	Quantor	
n	sans	$\exists$	sans	$\exists$	$\forall$	$\exists$	$\exists$	sans	$\exists$
4	0,3	**0,1**	0,5	**0,1**	1,0	2,98	0,26	8,53	0,14
5	0,5	**0,1**	0,8	**0,1**	2,3	177,48	2,16	70,16	0,50
6	1,0	**0,1**	1,3	**0,1**	11,5	T	17,90	M	3,04
7	2,5	0,2	4,1	**0,1**	41,1	T	155,93	M	15,02
8	5,5	0,4	13,3	**0,2**	23,1	T	1199.67	M	53,43
9	9,4	1,1	5,1	**0,3**	139,0	T	T	M	M
10	23,7	6,0	11,5	**0,4**	53,2	T	T	M	M
11	125,5	31,1	24,4	**0,4**	259,2	T	T	M	M
12	170,9	143,7	18,0	**0,5**	128,1	T	T	M	M
13	247,6	756,1	11,9	**0,6**	509,9	T	T	M	M
14	915,6	T	19,7	**0,8**	1710,4	T	T	M	M
15	3501,4	T	15,6	**0,9**	2502.1	T	T	M	M
16	T	T	24,4	**0,9**	T	T	T	M	M
17	T	T	78,0	**1,4**	T	T	T	M	M
18	T	T	68,4	**1,6**	T	T	T	M	M
19	T	T	59,4	**1,1**	T	T	T	M	M
20	T	T	130,9	**1,3**	T	T	T	M	M
21	T	T	78,1	**2,4**	T	T	T	M	M
22	T	T	101,4	**2,2**	T	T	T	M	M
23	T	T	119,9	**1,7**	T	T	T	M	M

TAB. 1 – Résultats en secondes pour différentes procédures avec les QBF initiales et les QBF modifiées.

quantifiées. Ces équivalences permettent de garder la taille de la formule et le nombre de variables, tout en sortant les quantificateurs afin de pouvoir appliquer une mise sous forme prénexe. Il est possible de choisir le type de quantification de la variable intermédiaire et ainsi réduire le nombre d'alternance de quantificateurs. Les QBF peuvent être vu comme un jeu à deux joueurs : le joueur existentiel essaye de rendre la formule valide alors que le joueur universel essaye de l'invalider. Un résultat intermédiaire est une conséquence des choix précédents, il n'y a aucun choix a faire, peu importe quel joueur joue ce coup. Notre proposition va dans ce sens, en permettant de faire jouer un seul des deux joueurs au choix lors d'un résultat intermédiaire dans une équivalence ou sa négation, sans augmenter ni la taille de la formule ni le nombre de variables.

Pour notre partie expérimentale, nous avons choisi la vérification formelle de circuits logiques, où une spécification doit être équivalente à la structure du circuit. Nos résultats montrent qu'un codage différent permettrait d'avoir de meilleurs résultats sans sur-coût. Toutefois, il est possible de rechercher les motifs de résultats intermédiaires afin d'effectuer un remplacement. Nos tests montrent que pour le problème choisi, la Q-résolution est néfaste au temps de résolution. Il serait intéressant de voir, si celle-ci pourrait être de nouveau intéressante si elle ne s'appliquait pas sur des résultats intermédiaires.

Pour utiliser nos équivalences, soit il faut coder le problème en les utilisant directement, soit il faut posséder le codage original du problème et effectuer des remplacements. Il serait donc intéressant d'avoir des procédures qui travaillent directement sur la formulation non prénexe. Elles pourraient ensuite en interne mettre sous FNC si nécessaire et classer les variables en deux catégories : les variables du problème et les résultats intermédiaires. Il serait alors possible de

connaître l'impact des heuristiques telle la Q-résolution et de traiter différemment les résultats intermédiaires.

Dans nos travaux futurs, nous nous intéresserons à :

– étendre nos résultats au cas général, afin d'être capable de traiter les quantificateurs dans des équivalences et des ou exclusifs, pas seulement pour les résultats intermédiaires ;
– comparer les nouvelles stratégies possibles pour la mise sous forme prénexe car nos résultats semblent montrer que le choix du type de quantificateur pour les résultats intermédiaires peut influencer grandement le temps de résolution ;
– développer une procédure qui prend en entrée une formule quelconque (non prénexe et non FNC), en effet nos résultats montrent qu'il est difficile de trouver une bonne stratégie pour mettre sous FNC.

Références

AYARI A. & BASIN D. (2002). Qubos : Deciding Quantified Boolean Logic using Propositional Satisfiability Solvers. In *Formal Methods in Computer-Aided Design, Fourth International Conference, FMCAD 2002* : Springer-Verlag.

AYARI A., BASIN D. A. & FRIEDRICH S. (1999). Structural and Behavioral Modeling with Monadic Logics. In *ISMVL*, p. 142–151.

BENEDETTI M. (2005). sKizzo : A Suite to Evaluate and Certify QBFs. In R. NIEUWENHUIS, Ed., *CADE*, volume 3632 of *Lecture Notes in Computer Science*, p. 369–376 : Springer.

BIERE A. (2004). Resolve and Expand. In *7th Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04)*.

BUBECK U. & BÜNING H. K. (2007). Bounded Universal Expansion for Preprocessing QBF. In J. MARQUES-SILVA & K. A. SAKALLAH, Eds., *SAT*, volume 4501 of *Lecture Notes in Computer Science*, p. 244–257 : Springer.

BÜNING H. K., KARPINSKI M. & FLÖGEL A. (1995). Resolution for Quantified Boolean Formulas. *Inf. Comput.*, **117**(1), 12–18.

DAVIS M., LOGEMANN G. & LOVELAND D. W. (1962). A machine program for theorem-proving. *Commun. ACM*, **5**(7), 394–397.

DE LA TOUR T. B. (1992). An Optimality Result for Clause Form Translation. *J. Symb. Comput.*, **14**(4), 283–302.

EGLY U. (1996). On Different Structure-Preserving Translations to Normal Form. *J. Symb. Comput.*, **22**(2), 121–142.

EGLY U., SEIDL M., TOMPITS H., WOLTRAN S. & ZOLDA M. (2003). Comparing Different Prenexing Strategies for Quantified Boolean Formulas. In E. GIUNCHIGLIA & A. TACCHELLA, Eds., *SAT*, volume 2919 of *Lecture Notes in Computer Science*, p. 214–228 : Springer.

GIUNCHIGLIA E., NARIZZANO M. & TACCHELLA A. (2004). QuBE++ : An Efficient QBF Solver. In A. J. HU & A. K. MARTIN, Eds., *FMCAD*, volume 3312 of *Lecture Notes in Computer Science*, p. 201–213 : Springer.

PLAISTED D. A. & GREENBAUM S. (1986). A Structure-Preserving Clause Form Translation. *Journal of Symbolic Computation*, **2**(3), 293–304.

TSEITIN G. S. (1970). On the complexity of derivation in propositional calculus. In A. O. SLISENKO, Ed., *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, p. 115–125. New York : Consultants Bureau.