PROJET TUTORÉ :
Formes normales en logique propositionnelle

Martin LENSCHAT, Pierre MERCURIALI, Stéphane TIV
Encadrant : Miguel COUCEIRO

2015

Master Sciences de la Cognition et Applications – M1

Équipe Orpailleur – Loria
Année universitaire 2014–2015
Formes normales en logique propositionnelle
Étudiants : Martin Lenschat, Pierre Mercuriali, Stéphane Tiv
Tuteur : Miguel Couceiro

Table des matières

Liste des livrables

— Un programme Java effectuant les opérations décrites dans la section 1

Notations et abréviations utilisées

— CNF : Conjunctive Normal Form (forme normale conjonctive)
— DNF : Disjunctive Normal Form (forme normale disjonctive)
— PNF : Polynomial Normal Form (forme normale polynomiale)
— MNF : Median Normal Form (forme normale médiane)
— N^* : l'ensemble des entiers naturels, privé de 0
— $\mathbb{B}$: l'ensemble $\{0, 1\}$

1 Présentation du projet

Le projet s'inscrit dans le cadre de la logique propositionnelle et de la théorie des fonctions booléennes ; plus précisément, dans la recherche d'une meilleure compréhension des fonctions booléennes et de leurs représentations, par l'étude, par exemple, de l'efficacité de celles-ci.

Au cours de celui-ci, nous avons essayé de répondre à deux problématiques principales concernant la représentation des fonctions booléennes, notamment :

— Comment obtenir, d'une table de vérité décrivant explicitement une fonction booléenne, une représentation implicite de cette même fonction sous forme normale ?

— Comment traduire une fonction booléenne d'une forme normale à une autre ?

Nous nous sommes restreints aux formes normales les plus connues (forme normale disjonctive, forme normale conjonctive, forme polynomiale), ainsi qu'une forme un peu moins étudiée (forme médiane) mais dont les expressions sont asymptotiquement moins complexes que celles des autres formes ([1]). Celle -ci repose sur l'opérateur médiane à trois entrées, qui retourne la valeur la plus fréquente apparaissant parmi les entrées ; par exemple,

$$\text{med}(1,1,0) = 1$$
$$\text{med}(0,1,0) = 0$$

Nous avons développé un programme pour résoudre ces deux problématiques. Celui-ci se présente sous la forme d'une fenêtre dans laquelle l'utilisateur peut rentrer une fonction dans une forme normale (ou en représentation explicite par table de vérité ou treillis booléen), choisir la forme de sortie désirée, puis, en sortie, obtenir ladite forme. Les représentations en sortie sont à la fois textuelles et visuelles, une fonction booléenne sous forme normale pouvant être représentée sous la forme d'un arbre.

Le but final de ce programme est d'implémenter les traductions entre les différentes formes normales, tout comme leur génération et représentation à partir de tables de vérité (afin de servir d'outil de recherche dans les domaines sus-mentionnés, en testant, par exemple, la complexité des formes normales obtenues).

2 Préliminaires

Au cours de cette partie, nous allons introduire et définir des concepts relatifs aux fonctions booléennes et à leurs représentations, ainsi qu'aux

opérateurs logiques utilisés.

2.1 Calcul booléen, fonctions booléennes

2.1.1 Définitions

La notion de fonction booléenne repose au cœur du projet.

Définition 1 *Soit $n \in \mathbb{N}^*$. Une* fonction booléenne *f est une fonction*

$$f : \mathbb{B}^n \to \mathbb{B}$$

*L'entier n est appelé l'*arité *de la fonction f.*

Une fonction $f : \mathbb{B}^n \to \mathbb{B}$ peut être définie explicitement grâce à une *table de vérité* : chaque point de $\mathbb{B}^n$ est inscrit dans un tableau, ainsi que la valeur de la fonction en ce point.

x	y	$f(x,y)$
0	0	0
0	1	1
1	0	1
1	1	1

FIGURE 1 – Exemple de table de vérité

Le pendant d'une telle description explicite d'une fonction booléenne f en est une description implicite, beaucoup plus compacte. Elle peut ainsi être représentée par une formule du calcul propositionnel, en utilisant les opérateurs logiques habituels ($\wedge, \vee, \oplus \cdots$).

Exemple 1 *La disjonction $\vee$ peut être évaluée par une fonction booléenne*

$$f : (x, y) \mapsto x \vee y$$

qui est donc donnée par la fonction de la figure 1.

Définition 2 *Projections*
Soit $n \in \mathbb{N}^*$. Pour tout $1 \leq i \leq n$, la i-ème projection *sur $\mathbb{B}^n$ est la fonction*

$$(a_1, \cdots, a_i, \cdots, a_n) \mapsto a_i$$

Les projections sont notées $x_i^{(n)}$ ou x_i, et sont aussi appelées variables.

Définition 3 *Ordre sur les vecteurs booléens*
Soit $(x_1, \cdots, x_n), (y_1, \cdots, y_n) \in \mathbb{B}^n$. On dit que $(x_1, \cdots, x_n)$ est inférieur (resp. supérieur) à $(y_1, \cdots, y_n)$, et on écrit $(x_1, \cdots, x_n) \leq (y_1, \cdots, y_n)$ (resp. $(x_1, \cdots, x_n) \geq (y_1, \cdots, y_n)$) si

$$\forall i \in \{1, \cdots, n\}, x_i \leq y_i \text{ (resp. } x_i \geq y_i)$$

Définition 4 *Fonctions monotones*
Soit $f : \mathbb{B}^n \to \mathbb{B}$ *une fonction booléenne. On dit que* f *est* monotone *si*

$$\forall (x_1, \cdots, x_n), (y_1, \cdots, y_n) \in \mathbb{B}^n,$$

$$(x_1, \cdots, x_n) \leq (y_1, \cdots, y_n) \Rightarrow f(x_1, \cdots, x_n) \leq f(y_1, \cdots, y_n)$$

ou

$$\forall (x_1, \cdots, x_n), (y_1, \cdots, y_n) \in \mathbb{B}^n,$$

$$(x_1, \cdots, x_n) \geq (y_1, \cdots, y_n) \Rightarrow f(x_1, \cdots, x_n) \geq f(y_1, \cdots, y_n)$$

Cette notion sera utile lors de l'introduction de certains algorithmes de conversion se basant sur le cas particulier des fonctions booléennes monotones.

Définition 5 *Composition de fonctions*
Soit f *une fonction booléenne d'arité* n*, et* $g_1, \cdots, g_n$ *des fonctions booléennes d'arité* m*. Alors, leur* composition $f(g_1, \cdots, g_n)$ *est une fonction booléenne d'arité* m*, définie par :*

$$f(g_1, \cdots, g_n)(a_1, \cdots, a_m) = f(g_1(a_1, \cdots, a_m), \cdots, g_n(a_1, \cdots, a_m))$$

2.1.2 Exemples d'application des fonctions booléennes

Par la suite, nous encodons les valeurs Vrai et Faux en Logique propositionnelle par les chiffres 0 et 1.

La plupart de ces exemples sont adaptés de ceux présentés dans [3].

Comme nous l'avons vu, une fonction booléenne peut être définie en termes de logique propositionnelle. Réciproquement, la valeur sémantique d'une phrase en logique propositionnelle peut être obtenue grâce aux fonctions booléennes. Considérons, par exemple, la phrase suivante :

> Si la Terre est avalée par un trou noir, je n'irai pas en cours.

En notant P et Q les sous propositions

P : La Terre est avalée par un trou noir
Q : Je n'irai pas en cours

La phrase complète admet une représentation syntaxique en logique propositionnelle :

$$\neg P \vee Q$$

Ainsi qu'une représentation sémantique, par une fonction booléenne :

P	Q	$f(P,Q)$
0	0	1
0	1	1
1	0	0
1	1	1

FIGURE 2 – Représentation sémantique de $\neg P \vee Q$

L'intérêt de manipuler des fonctions sous forme implicite apparaît, même avec un petit nombre de variables : c'est un gain de place considérable. Par exemple, avec si peu que 6 variables, une description explicite d'une fonction sur $\mathbb{B}^6$ occuperait $2^6 = 64$ lignes !

La branche de *l'électronique numérique* s'occupant des portes logiques fait grande utilisation des fonctions booléennes et des théories associées dans la conception et l'étude de circuits. Deux états encodés par 0 et 1 peuvent correspondre à différents états (tension haute et tension basse, circuit ouvert et circuit fermé...) réels. Les portes logiques font office de fonctions agissant sur des entrées modélisées. La composition de ces portes logiques permet de constituer des circuits plus complexes ; la théorie des fonctions booléennes permet de simuler le circuit, de le simplifier, et même de prévoir son comportement !

2.2 Les treillis Booléens

Un *treillis* (aussi nommé en anglais *lattice*) est, dans le contexte de l'algèbre booléenne, une représentation d'un ensemble complet, distributif et borné d'éléments fini dans $\{0,1\}^n$, n étant le nombre de variables de la fonction booléenne considérée.

Cette représentation graphique d'une table de vérité fait correspondre chaque nœud du treillis à l'une des lignes de la table, ceux-ci étant caractérisés par les états de leurs variables (0 ou 1) ainsi que par leurs liaisons. Ces liaisons sont présentes lorsque, pour un nœud a et un nœud b donnés, l'état des variables de a et l'état des variables de b ne diffèrent que pour une seule et même variable. Cela étant, chaque nœud possède donc n variables et n liaisons.

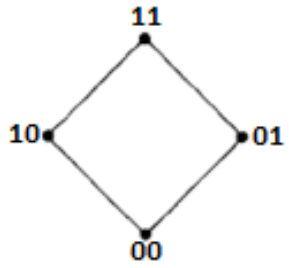


FIGURE 3 – Treillis à deux variables

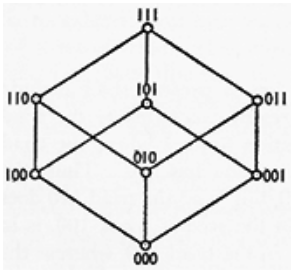


FIGURE 4 – Treillis à trois variables

Dans le cadre de notre programme, nous offrons la possibilité à l'utilisateur de se servir d'un *treillis* afin de renseigner les valeurs de vérité de sa fonction booléenne. Ainsi chaque nœud du *treillis* représente une ligne de la table de vérité, sa valeur en sortie valant 0 ou 1 en fonction de la couleur du nœud (blanc ou noir).

Exemple 2 *Treillis et table de vérité de la fonction* $f(x, y) = x \oplus y$ *:*

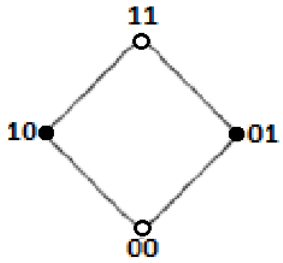


FIGURE 5 – Treillis de f

x	y	$med(x, y, z)$
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 6 – Table de vérité de f

2.3 L'opérateur Médiane

Le connecteur de médiane *med* est un cas particulier de la fonction *majority* et prend un ensemble ternaire d'arguments dans $\{0, 1\}$. Il est satisfait si deux ou trois de ses arguments sont vrais, selon la table de vérité suivante :

x	y	z	$med(x, y, z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Elle possède plusieurs propriétés remarquables :

Propriété 1 *Commutativité des éléments :*
$med(x_1, x_2, x_3) = med(x_{\sigma(1)}, x_{\sigma(2)}, x_{\sigma(3)})$ *pour toute permutation* σ *de* $\{1, 2, 3\}$

Propriété 2 $med(x, y, y) = y$

Propriété 3 $med(x, y, 0) = x \wedge y$

Propriété 4 $med(x, y, 1) = x \vee y$

Propriété 5 $med(x, \overline{x}, y) = y$

2.4 Notions de Complexité

La notion de complexité algorithmique est un problème majeur dans de nombreux domaines scientifiques, notamment en mathématiques et en informatique. Celle-ci est usuellement définie en terme de :

- temps, la durée (ou puissance de calcul) nécessaire pour effectuer les opérations ;

• espace, la mémoire nécessaire pour effectuer les opérations.

Dans le cas de formules booléennes, ce que nous entendons par complexité correspond à la taille de celle-ci, la quantité d'opérateurs et de variables nécessaires afin de représenter une fonction Booléenne. De plus une preuve ([1]) a montré que la MNF a une complexité asymptotiquement moins élevée que les autres formes normales normalement usitées.

Prenons cette table de vérité en exemple :

x	y	z	$f(x,y,z)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

FIGURE 7 – Exemple de table de vérité

Écrite sous les différentes formes normales, cela donne :

- MNF : $med(x, y, z)$
- CNF : $(x \vee y) \wedge (x \vee z) \wedge (y \vee z)$
- DNF : $(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$
- PNF : $(x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \oplus (x \wedge y \wedge z)$

Leur complexité respective est donc de :

- MNF : 4
- CNF : 11
- DNF : 11
- PNF : 17

Il faut également noter que la complexité de la CNF, pour un nombre de variables n donné, a une complexité de moins en moins élevé selon que le nombre de lignes égales à 1 dans la table de vérité devient élevée. Cela peut s'expliquer par la façon dont est construite la CNF.

De la même façon, la DNF a une complexité de plus en plus élevé selon que le nombre de lignes égale à 0 dans la table de vérité devient élevée.

3 Formes normales

Toute expression booléenne peut prendre une valeur de vérité vraie ou fausse. On peut voir une expression comme une fonction booléenne qui prend

un certain nombre de paramètres vrais ou faux et qui retourne un résultat de valeur vraie ou fausse. Cette fonction peut être décrite par une table de vérité qui développe toutes les valeurs possibles pour tous ses paramètres.

3.1 Motivation

Le nombre d'expressions booléennes qui expriment la même fonction est infini. Pour exprimer les expressions d'une même fonction, il est important de trouver une moyen permettant d'identifier ces expressions les unes par rapport aux autres, et d'obtenir une forme commune pour des expressions différentes.

En utilisant la décomposition de Shannon, la forme normale disjonctive (DNF) a été trouvée. La forme normale conjonctive (CNF) peut en être dérivée par les Lois de Morgan. Puis l'opérateur XOR (ou exclusif) a été introduit pour définir une forme normale avec une structure de polynôme de conjonctions (Reed-Muller et/où Zhegalkin polynomials). La particularité de ces formes normales est une représentation plus compacte d'une fonction booléenne donnée.

Plus tard la forme normale médiane (Median Normal Form) a été introduite et qui en comparaison avec les trois précédentes, est une représentation bien plus efficace ([1]).

Définition 6 *Un* littéral *est une expression de la forme x ou $\overline{x}$ où x est une variable booléenne.*

Définition 7 *Une* conjonction élémentaire *(ou* terme*) est une expression de la forme*

$$C = \bigwedge_{i \in A} x_i \wedge \bigwedge_{j \in B} \overline{x}_j, \text{ où } A \cap B = \emptyset$$

Où A et B sont des sous-ensembles disjoints d'indices

Définition 8 *Une* disjonction élémentaire*(ou* clause*) est une expression de la forme*

$$D = \bigvee_{i \in A} x_i \vee \bigvee_{j \in B} \overline{x}_j, \text{où} A \cap B = \emptyset$$

Où A et B sont des sous-ensembles disjoints d'indices ([3]).

3.2 La forme normale disjonctive (DNF)

Une DNF est une disjonction d'une ou de plusieurs conjonctions d'un ou de plusieurs littéraux. Pour cette forme, les seuls opérateurs sont la conjonction, la disjonction et la négation. De plus l'opérateur non ne doit être utilisé que sur les littéraux.

Définition 9 *Une forme normale disjonctive est une expression de la forme*

$$\bigvee_{k=1}^{m} C_k = \bigvee_{k=1}^{m} \Big(\bigwedge_{i \in A_k} x_i \bigwedge_{j \in B_k} \overline{x}_j \Big),$$

où chaque $C_k (k = 1, 2, \ldots, m)$ est une conjonction élémentaire ; on dit que chaque conjonction C_k est un terme de la DNF ([3]).

Algorithme de conversion à partir d'une table de vérité
Soit τ une table de vérité

— Ecrire la table de vérité τ
— Pour chaque ligne ayant comme valeur de vérité 1, écrire la conjonction des littéraux, $Ci = (l_1 \wedge \ldots \wedge l_r)$ de la façon suivante : si les variables sont $p_1, \ldots, p_r$ et la variable p_i vaut 1, le litteral l_i est p_i. Sinon si la variable a pour valeur 0, le litteral l_i est $\overline{p}_i$
— Prendre la disjonction $(C_1 \vee \ldots \vee C_k)$ de toutes les clauses $C_i = (l_1 \wedge \ldots \wedge l_r)$ correspondant aux lignes de la table qui ont pour valeur 1.

Exemple 3 *Soit la table de vérité suivante*

x	y	z	$f(x,y,z)$
0	*0*	*0*	*1*
0	*0*	*1*	*0*
0	*1*	*0*	*0*
0	*1*	*1*	*1*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*0*
1	*1*	*1*	*1*

Pour chaque ligne ayant pour valeur 1, on écrit la conjonction des littéraux selon les règles décrites. On obtient :

x	y	z	$f(x,y,z)$	
0	*0*	*0*	*1*	$\overline{x}\ \overline{y}\ \overline{z}$
0	*0*	*1*	*0*	
0	*1*	*0*	*0*	
0	*1*	*1*	*1*	$\overline{x}\ y\ z$
1	*0*	*0*	*0*	
1	*0*	*1*	*1*	$x\ \overline{y}\ z$
1	*1*	*0*	*0*	
1	*1*	*1*	*1*	$x\ y\ z$

On prend alors la disjonction de toutes ces lignes pour obtenir la forme normale disjonctive

$$f(x,y,z) = (\overline{x} \wedge \overline{y} \wedge \overline{z}) \vee (\overline{x} \wedge y \wedge z) \vee (x \wedge \overline{y} \wedge z) \vee (x \wedge y \wedge z)$$

3.3 La forme normale conjonctive (CNF)

Une CNF est une conjonction d'une ou de plusieurs disjonctions d'un ou de plusieurs littéraux. Pour cette forme, les seuls opérateurs sont également la conjonction, la disjonction et la négation. De même l'opérateur non ne doit être utilisé que sur les littéraux.

Définition 10 *Une forme normale conjonctive est une expression de la forme*

$$\bigwedge_{k=1}^{m} D_k = \bigwedge_{k=1}^{m} \Big(\bigvee_{i \in A_k} x_i \vee \bigvee_{j \in B_k} \overline{x}_j \Big),$$

Où chaque $D_k (k = 1, 2, \ldots, m)$ est une disjonction élémentaire ; on dit que chaque conjonction D_k est une clause de la CNF ([3]).

Algorithme de conversion à partir d'une table de vérité
Soit τ une table de vérité

- Ecrire la table de vérité τ
- Pour chaque ligne ayant comme valeur de vérité 0, écrire la disjunction des littéraux, $D_i = (l_i \vee \ldots \vee l_r)$ de la façon suivante : si les variables sont $p_1, \ldots, p_r$ et la variable p_i vaut 0, le litteral l_i est p_i. Sinon si la variable a pour valeur 1, le litteral l_i est $\overline{p}_i$
- Prendre la conjonction $(C_1 \wedge \ldots \wedge C_k)$ de toutes les clauses $D_i = (l_1 \vee \ldots \vee l_r)$ correspondant aux lignes de la table qui ont pour valeur 0.

Exemple 4 *Soit la table de vérité suivante*

x	y	z	$f(x, y, z)$
0	*0*	*0*	*0*
0	*0*	*1*	*1*
0	*1*	*0*	*1*
0	*1*	*1*	*0*
1	*0*	*0*	*1*
1	*0*	*1*	*0*
1	*1*	*0*	*1*
1	*1*	*1*	*0*

Pour chaque ligne ayant pour valeur 0, on écrit la disjonction des littéraux selon les règles décrites. On obtient :

x	y	z	$f(x,y,z)$	
0	*0*	*0*	*0*	$x \vee y \vee z$
0	*0*	*1*	*1*	
0	*1*	*0*	*1*	
0	*1*	*1*	*0*	$x \vee \overline{y} \vee \overline{z}$
1	*0*	*0*	*1*	
1	*0*	*1*	*0*	$\overline{x} \vee y \vee \overline{z}$
1	*1*	*0*	*1*	
1	*1*	*1*	*0*	$\overline{x} \vee \overline{y} \vee \overline{z}$

On prend alors la conjonction de toutes ces lignes pour obtenir la forme normale conjonctive

$$f(x,y,z) = (x \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

3.4 La forme normale polynomiale (PNF, Reed-Muller, Zhegalkin polynomial ou Forme Normale Algebrique)

Définition 11 *Une forme polynomiale est une expression de la forme :*

$$f = \sum_{I \subset \{1,\ldots,n\}} \alpha_i \prod_{i \in I} x_i + cst,\ I \neq \emptyset$$

où les I sont les sous-ensembles de l'ensemble {1,...,n}, les α_i sont des coefficients booléens ($\in \{0,1\}$),les x_i des variables booléennes et les sommes sont des sommes modulo 2.

ou encore de la forme :

$$\begin{aligned} f(x_0, x_1, \ldots, x_{n-1}) =& a_0 x_0 \oplus a_1 x_1 \oplus \ldots \oplus a_{0,1} x_0 x_1 \oplus \ldots \oplus \\ & a_{0,n-1} x_0 x_1 \oplus \ldots \oplus a_{0,1,\ldots,n-1} x_0 x_1 \ldots x_{n-1} \end{aligned}$$

où $a_0, a_1, \ldots, a_{0,1,\ldots,n-1}$ sont des coefficients binaires et $x_0, x_1, \ldots, x_{n-1}$ sont des variables booléennes.

Les seuls opérateurs sous cette forme sont le $\oplus$ et la conjonction(produit) avec des constantes(0 et 1), il n'y a pas de négation.

Exemple 5 *L'expression $f(x,y,z) = x \oplus y \oplus yz$ est en forme normale polynomiale. Il consiste en une somme modulo 2 de conjonction avec des coefficients. On peut l'écrire sous la forme suivante : $f(x,y,z) = 0 \oplus 1x \oplus 1y \oplus 0z \oplus 0(xy) \oplus 0(xz) \oplus 1(yz) \oplus 0(xyz)$*

Algorithme de conversion à partir d'une table de vérité

Soit τ une table de vérité qui décrit une fonction f

La PNF est de la forme :

$$\begin{aligned} f(x_1, \ldots, x_n) = & a_0 \oplus \\ & a_1 x_1 \oplus \ldots a_n x_n \oplus \\ & a_{12} x_1 x_2 \oplus \ldots a_{n-1n} x_{n-1} x_n \oplus \\ & \ldots \\ & a_{1,\ldots,n} x_1 \ldots x_n \end{aligned}$$

Pour trouver les coefficients, il faut considérer le vecteur de la fonction en fonction des variables associés au coefficient. Pour chaque ensemble de variables $(x_1x_2, x_1 \ldots x_n)$, on prend en compte le vecteur d'une sous fonction de f qui a pour paramètres ces variables.

Par exemple pour $x_1 \ldots x_n$ on considère la fonction f en entier, pour x_1x_2 on ne prend en compte que les lignes avec toutes les autres variables à 0, autrement dit une sous fonction de f qui a comme paramètre x_1 et x_2.

On calcule alors pour chacun des vecteur obtenu la parité du poids de Hamming de ce vecteur. Le poids de Hamming d'un vecter booléen correspond au nombre de 1 dans ce vecteur.

Si le nombre de 1 est pair, la valeur du coefficient vaut 0, sinon le coefficient vaut 1.

On remplace alors les coefficients dans l'expression, on obtient ainsi une PNF.

Exemple 6 *Soit τ la table de vértié suivante :*

x	y	z	$f(x,y,z)$
0	*0*	*0*	*0*
0	*0*	*1*	*1*
0	*1*	*0*	*1*
0	*1*	*1*	*0*
1	*0*	*0*	*0*
1	*0*	*1*	*1*
1	*1*	*0*	*0*
1	*1*	*1*	*1*

On écrit la PNF :

$$\begin{aligned} f(x, y, z) = & a_0 \oplus \\ & a_1 x \oplus a_2 y \oplus a_3 z \oplus \\ & a_{12} xy \oplus a_{13} xz \oplus a_{23} yz \oplus \\ & a_{123} xyz \end{aligned}$$

On doit trouver les coefficients a_i.

Soit $c = \{f(0), \ldots, f(7)\}$.

On considère la parité p du poids de Hamming w en fonction du coefficient à trouver.

Si le nombre de 1 est pair, le coefficient vaut 0, sinon il vaut 1.

Pour a_{123} on prend en compte $p(w(c))$.

Pour a_{12} on ne considère un sous ensemble de c qui comprend les valeurs pour x et y, donc les lignes pour $z = 0$.

$a_{123} = p(w(c))$ *avec* $c = \{0, 1, 1, 0, 0, 1, 0, 1\}$ *on* $a_{123} = 0$

$a_{12} = p(w(c'))$ *avec* $c' = \{0, 1, 0, 0\}$ *on a* $a_{12} = 1$

On procède d'une façon similaire pour tous les autres coefficients.

On obtient les coefficients suivants :

$$\begin{aligned} a_{123} &= 0 \\ a_{12} &= 1 \\ a_{13} &= 0 \\ a_{23} &= 0 \\ a_1 &= 0 \\ a_2 &= 1 \\ a_3 &= 1 \\ a_0 &= 0 \end{aligned}$$

On remplace les coefficients pour obtenir l'expression en PNF :

$$f(x, y, z) = xy \oplus y \oplus z$$

3.5 La forme normale médiane (Median Normal Form)

Définition 12 *Une forme normale médiane est une expression de la forme :*

$$f = med(x_1, x_2, x_3)$$

où x_1, x_2 et x_3 sont des constantes(0 ou 1), des variables booléennes ou des formes normales médiane.

La forme normale médiane ne comprend que l'opérateur médiane, la négation et des constantes (0 et 1). La négation ne s'utilise que sur les littéraux.

Proposition 1 *Toute fonction boolénne peut s'écrire sous une forme normale médiane.*

Lemme 1 *Soit $f : \mathbb{B}^n \to \mathbb{B}$ une fonction booléenne. Si f est monotone alors pour tout $i \in n$, et $C \in \{0, 1\}$ $f(x_1, \ldots, C, \ldots, x_n)$ est monotone*

Demonstration 1 *Soit $f : \mathbb{B}^n \to \mathbb{B}$ une fonction booléenne.*

Si f est monotone,

On a pour tout i :

$$f(x_1, \ldots, x_n) = med(f(x_1, \ldots, 0, \ldots, x_n), x_i, f(x_1, \ldots, 1, \ldots, x_n))$$

où 0 et 1 sont respectivement la ième composante du vecteur $x_1, \ldots, 0, l \ldots x_n$ *et du vecteur* $x_1, \ldots, 1, l \ldots x_n$.

D'après le Lemme 1, $f(x_1, \ldots, 0, \ldots, x_n)$ *et* $f(x_1, \ldots, 1, \ldots, x_n)$ *sont monotones.*

On peut donc décomposer récursivement les fonctions de la même façon et obtenir une forme normale médiane.

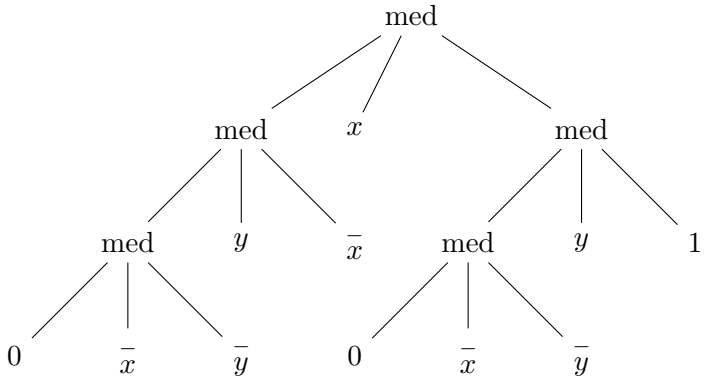
Exemple 7 $med(x_1, x_2, 0) = 1$ *pour* $x_1 = 1$ *et* $x_2 = 1$

L'expression $f(x, y) = med(med(med(0, \overline{x}, \overline{y}), y, \overline{x}), x, med(med(0, \overline{x}, \overline{y}))$ *est une forme normale médiane. Elle ne contient que l'opérateur médiane, ainsi que des négations et des constantes.*

On peut également représenter l'expression d'une forme normale médiane comme un arbre où chaque noeud est un opérateur médiane et chaque feuille est une constante ou un littéral.

FIGURE 8 – Représentation d'une expression sous forme d'arbre

$$f(x, y) = med(med(med(0, \overline{x}, \overline{y}), y, \overline{x}), x, med(med(0, \overline{x}, \overline{y}))$$

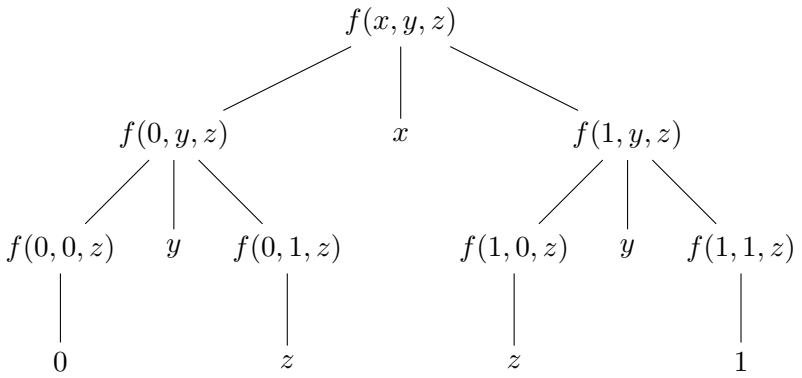


Algorithme de conversion à partir d'une table de vérité pour les fonctions monotones

Algorithm 1 Algorithme MMNF [2]

if $n \geq 2$ **then**
    $\alpha \leftarrow (MMNF(f(x_1, \ldots, x_{n-1}, 0))$
    $\beta \leftarrow (MMNF(f(x_1, \ldots, x_{n-1}, 1))$
    return $\mu\alpha x_n\beta$
else
    if $f = 0$ **then**
        return 0
    else
        if $f = 1$ **then**
            return 1
        else
            return x_1

FIGURE 9 – Construction de la forme normale médiane d'une fonction booléenne monotone



$$f(x, y, z) = med(med(0, 0, z), y, z), x, med(z, y, 1)$$

Algorithme de conversion pour le cas général[COU 11]

Dans le cas ou la fonction f n'est pas monotone, on construit une fonction $g_f : \mathbb{B}^{2n} \to \mathbb{B}$ de telle sorte que f soit recouverte par g_f en substituant des variables niées par d'autres variables, afin d'obtenir une représentation en forme médiane de f.

Soit $f : \mathbb{B}^n \to \mathbb{B}$ une fonction boolénne.

On construit g_f de la façon suivante :

Pour tout $a = (a_1, \ldots, a_{2n} \in \mathbb{B}^{2n}$, et soit $b = (a_1, \ldots, a_n)$, $c = (a_{n+1}, \ldots, a_{2n})$,

$$g_f(a) = \begin{cases} 0 & \text{if } w(a) < n, \\ 1 & \text{if } w(a) > n, \\ f(b) & \text{si } b = \overline{c}, \\ 0 & \textit{sinon} \end{cases}$$

w représente le *poids de Hamming* de a, le nombre de 1 dans a et pour tout $b \in \mathbb{B}^n$, $f(b) = g_f(b, \overline{b})$.

Donc de fait : g_f est monotone.

f peut être obtenue de g

$$f = g_f(x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n})$$

Algorithm 2 Algorithme GenMNF [2]

```
1: if f est monotone then
2:     return MMNF(f)
3: else
4:     Construire g_f
5:     w ← MMNF(G_f)
6:     for i = 1 à n do
7:         Remplacer chaque occurrence de x_{n+i} dans α par x̄_i
8:
9:     return w
```

Exemple 8 *soit la table de vérité suivante qui décrit la fonction* $f(x, y)$ *:*

x	y	$f(x,y)$
0	*0*	*1*
0	*1*	*0*
1	*0*	*0*
1	*1*	*1*

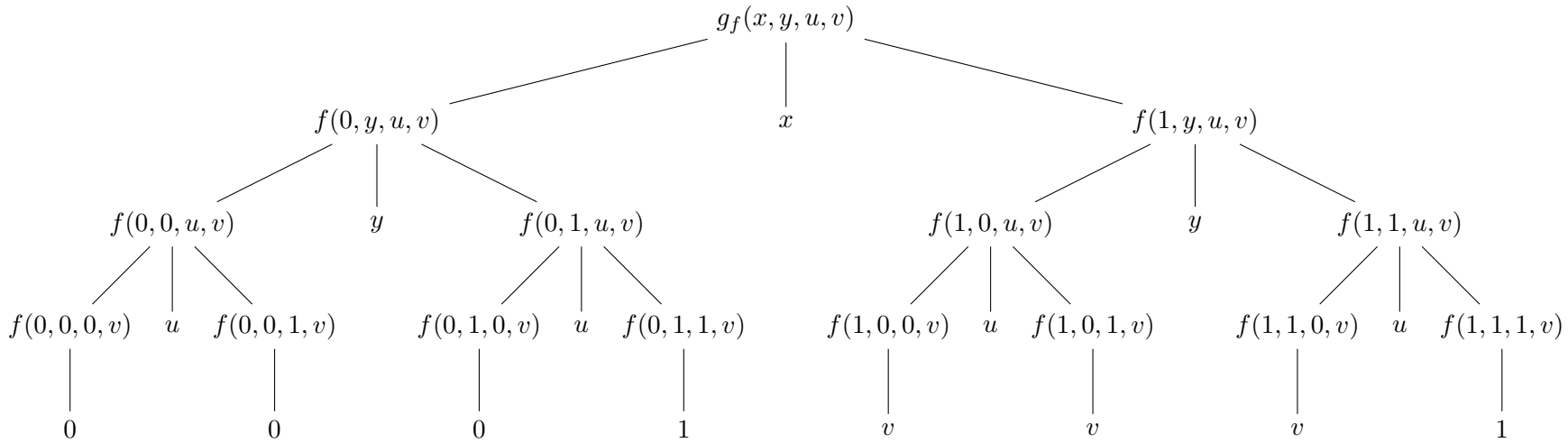
On construit g_f *avec* $a = (x, y, u, v)$, $b = (x, y)$ *et* $c = (u, v)$

$$g_f(x, y, u, v) = \begin{cases} 0 & \textit{si } w(a) < 2, \\ 1 & \textit{si } w(a) > 2, \\ f(b) & \textit{si } b = \overline{c}, \\ 0 & \textit{sinon} \end{cases}$$

Calcul de $f(b)$ *dans le cas où* $b = \overline{c}$

1100	$f(11) = 0$
1001	$f(10) = 1$
0110	$f(01) = 1$
0011	$f(00) = 0$

FIGURE 10 – Construction de la forme normale médiane d'une fonction booléenne non monotone



On obtient l'expression de g_f suivate :

$$g_f(x, y, u, v) = med(med(0, y, u), x, med(v, y, med(v, u, 1)))$$

En remplaçant respectivement les variables u et v par la négation de x et y, on obtient la forme normale médiane de f :

$$f(x, y) = med(med(0, y, \overline{x}), x, med(\overline{y}, \overline{x}, 1))$$

4 Implémentation et démarche

4.1 Présentation générale du programme et fonctionnalités

Le but du programme que nous avons réalisé est donc de permettre la traduction d'équations booléennes dans différentes formes. Pour cela, nous avons décidé de créer un programme JAVA qui donnerait à l'utilisateur la possibilité de remplir une table de vérité ou un treillis avant d'opérer la transformation de l'équation donnée par celle-ci en l'une des formes proposées : CNF, DNF, PNF, ou MNF. Une fois cette traduction opérée, l'utilisateur peut alors observer l'équation produite. Notre programme propose également une visualisation sous forme d'arbre afin de rendre ces équations plus lisibles.

Voici une description des différentes parties de l'interface et leur utilisation (voir 6 pour les captures d'écran) :

- **1** *Table de Vérité* : l'utilisateur peut ici renseigner les valeurs de vérité de chaque ligne dans la dernière colonne de cette table.
- **2** *Nombre de variables* : l'utilisateur peut utiliser cet outil afin de modifier la table de vérité et le treillis afin de représenter une formule booléenne à deux ou trois variables.
- **3** *Conversion de Table de Vérité* : il faut ici renseigner la Forme Normale désirée en sortie avant de valider pour lancer la conversion.
- **4** *Treillis* : comme expliqué en 2.2 une représentation en treillis est offerte à l'utilisateur. Il peut ainsi renseigner la table de vérité qui se met automatiquement à jour lors de chaque clic sur un nœud du treillis.
- **5** *Résultat* : Cette zone est réservé à l'affichage du résultat.
- **6** *Écriture d'équation* : il est également possible d'utiliser ce champ de texte afin de taper une équation à la main avant de la convertir.
- **7** *Ajout d'opérateurs* : ce bouton permet à l'utilisateur d'ajouter des opérateurs booléens à la suite de son équation.
- **8** *Conversion d'équation* : comme pour le bouton *3*, celui-ci convertit l'équation entrée en *6*. En cas d'équation invalide, un message d'erreur vient alors s'afficher en *9*

- **9** *Affichage de l'arbre* : Cette zone est réservée à l'affichage d'un arbre représentant la Forme Normale obtenue. Cet affichage est obtenu en utilisant la librairie JAVA GraphStream ([5]).

4.2 Choix d'implémentation et de structure du programme

Dans cette sous-section, nous présentons les choix effectués en termes de structure et de manipulation des données dans le programme.

4.2.1 Choix des structures de données

Dans le programme, la plupart des données à manipuler sont des expressions en logique du premier ordre. Nous avons donc choisi de les représenter sous forme d'arbres : ainsi, nous avons directement accès à la structure d'une expression (chose que nous n'aurions pas si nous avions, par exemple, choisi de ne manipuler que des chaînes de caractères).

Nous avons donc créé un objet *Tree* ; un Tree contient une racine (*root*), qui est un caractère (voire une chaîne de caractères), ainsi qu'une liste de Tree (ses fils). La classe *Tree* contient un certain nombre de méthodes utiles, en plus des constructeurs et des méthodes d'accès aux champs (qui sont *root* et *list_of_children*) :

- void `add_child`(Tree t) : permet d'ajouter un fils à l'arbre t
- int `get_number_of_children`() : renvoie le nombre de fils de l'arbre
- public Boolean `is_leaf`() : renvoie VRAI si l'arbre est une feuille (pas de fils)
- public Boolean `equals`(Tree tree) : renvoie VRAI si l'arbre que l'on teste et tree sont égaux. Cette méthode repose sur un parcours récursif de l'arbre et une comparaison fils-à-fils non ordonnée des sous-arbres. Ainsi,
$$\wedge(a, b) \text{ et } \wedge(b, a)$$
sont égaux, selon cette méthode.
- public String `toString`() : renvoie une chaîne de caractères correspondant à l'expression concernée
- public void `print`() : appel de la méthode `prettyPrint` qui affiche l'expression de manière plus lisible, sous forme d'arbre, en console.

Le code des arbres est contenu dans le fichier `Tree.java`.

Pour stocker les tables de vérité, nous avons opté pour des hashtables : leurs clés sont les vecteurs booléens, et les valeurs sont les valeurs de la fonction en ces points. La convention choisie est que, pour un vecteur 001, par exemple, les valeurs correspondantes des variables soient $x_1 = 0, x_2 = 0, x_3 = 1$; la valeur correspondant de f, la fonction définie par la table, est donc
$$f(0, 0, 1)$$

4.2.2 Architecture des méthodes

Pour effectuer les conversions, nous avons choisi, autant que possible, de diviser les opérations en sous-méthodes effectuant une tâche simple et précise. Voici un exemple de procédure typique d'une méthode de conversion :

Algorithm 3 Exemple de conversion d'une forme normale à une autre

```
1: procedure CONVERSION_MNF_À_DNF(expression_sous_forme_médiane)
2:     expression_intermédiaire ← IDENTITÉ_M_À_D(expression_sous_forme_médiane)
3:     expression_finale ← CONVERSION_DNF(expression_intermédiaire)
4:     return expression_finale
```

Une telle méthode va ainsi d'abord remplacer, dans l'expression de départ, les symboles qui ne doivent pas apparaître dans l'expression finale, grâce à une sous-méthode (appelée, ici, IDENTITÉ_M_À_D) qui repose sur des identités telles que

$$\mathrm{med}(x, y, z) = \vee(\vee(\wedge(x, y), \wedge(x, z)), \wedge(y, z))$$

Puis, l'expression en résultant est mise sous la bonne forme par une autre méthode qui repose principalement sur la distribution d'une opération par rapport à une autre (autrement dit, elle remet les opérations dans le bon ordre pour avoir une forme normale correcte). Par exemple,

$$\vee(x, \wedge(a, b))$$

est réécrite

$$\wedge(\vee(x, a), \vee(x, b))$$

pour avoir une expression en DNF.

Cette sous-méthode fait elle-même appel à d'autres méthodes qui remplissent des rôles différents, comme des simplifications qui découlent des propriétés des opérateurs utilisés :

$$x \vee 0 \to x$$

4.3 Détail des méthodes

Le code des conversions est contenu dans le fichier `Converter.java`. Certaines méthodes étant très similaires dans leur structure, leur description est parfois plus générale, comme précédemment. Par exemple, les algorithmes de conversion vers la CNF ou DNF n'ont de différence fondamentale que l'interversion entre $\vee$ et $\wedge$. Nous commencerons par décrire certaines méthodes les plus générales, puis nous décrirons les sous-méthodes appelées par ces méthodes générales.

4.3.1 Méthodes générales

Ces méthodes sont appelées pour convertir directement une expression dans une autre, ou pour passer d'une table de vérité à une forme normale.

Pour le passage d'une forme normale à une autre, le principe suivant est appliqué :

Méthode 1 *public static Tree* `medianNormalFormToDisjunctiveNormalForm`*(Tree median_normal_form)*

Cette méthode repose sur l'identité

$$med(X, Y, Z) = \vee(\vee(\wedge(X, Y), \wedge(X, Z)), \wedge(Y, Z))$$

où X, Y, Z sont des expressions.

Si l'expression en entrée est une feuille, la méthode renvoie cette feuille. Sinon, les fils X, Y, Z de la médiane (car si l'expression n'est pas une feuille, elle est obligatoirement une médiane) sont convertis par la même fonction, puis stockés dans un nouveau Tree correspondant à la structure de droite dans l'expression ci-haut.

À ce point, l'expression ne contient plus l'opérateur med*, mais n'est pas sous la bonne forme : on appelle donc une méthode de conversion en DNF (décrit dans la sous-section suivante), pour finalement renvoyer le résultat.*

Voici les identités utilisées pour les conversions :

— MNF $\rightarrow$ DNF

$$\text{med}(X, Y, Z) = \vee(\vee(\wedge(X, Y), \wedge(X, Z)), \wedge(Y, Z))$$

— MNF $\rightarrow$ CNF

$$\text{med}(X, Y, Z) = \wedge(\wedge(\vee(X, Y), \vee(X, Z)), \vee(Y, Z))$$

— MNF $\rightarrow$ PNF

$$\text{med}(X, Y, Z) = \oplus(\wedge(X, Y), \oplus(\wedge(Y, Z), \wedge(Z, X)))$$

(la multiplication apparaît, dans le programme, comme une disjonction)

— DNF ou CNF $\rightarrow$ MNF

$$\vee(X, Y) = \text{med}(X, Y, 1) \text{ et } \wedge(X, Y) = \text{med}(X, Y, 0)$$

— PNF $\rightarrow$ MNF

$$\oplus(X, Y) = \vee(\wedge(X, \neg Y), \wedge(\neg X, Y))$$

puis les identités précédentes faisant intervenir $\vee$ et $\wedge$, pour donner l'identité :

$$\oplus(X, Y) = \text{med}(\text{med}(X, \neg Y, 0), \text{med}(\neg X, Y, 0), 1)$$

Pour le passage d'une table de vérité à une autre, les principes théoriques décrits dans la section précédente ont étés implémentés presque directement.

4.3.2 Méthodes de conversions

Certaines de ces méthodes sont directement inspirées de [4].

Ces méthodes prennent en argument une expression contenant les bons opérateurs pour la forme à convertir (par exemple $\vee$ et $\wedge$ pour la DNF), mais qui n'apparaissent pas dans le bon ordre. Ces méthodes visent à y remédier. Elles contiennent également des appels aux fonctions de simplification (particulièrement utiles pour la PNF, par exemple, puisque les simplifications débarrassent l'expression en entrée des constantes, ne laissant que des variables, et donnant donc une forme valide).

Toutes ces méthodes sont récursives : la condition d'arrêt est la feuille (si l'expression d'entrée est une feuille, il n'y a rien à faire). Si l'expression n'en est pas une, l'algorithme s'appelle lui-même au niveau des fils, qui sont alors de la bonne forme ; suivant la racine de l'expression (i.e. l'opérateur de tête de l'arbre), celle-ci est reconstituée, à partir de ses fils, dans la forme désirée.

Puisque les trois méthodes de conversion `convert_to_CNF`, `convert_to_DNF` et `convert_to_PNF` sont très similaires structurellement (les propriétés de distributivité étant les mêmes pour les trois opérateurs $\vee, \wedge, \oplus$), seule la première sera détaillée.

Méthode 2 *public static Tree* `convert_to_CNF`*(Tree expression)*
Cette méthode convertit une expression quelconque contenant des $\vee$ et des $\wedge$ en CNF.
Si l'expression est une feuille, il n'y a rien à faire et la méthode renvoie l'expression.
Sinon, les fils P et Q sont extraits, et on leur applique la méthode elle-même. Puisque la méthode les a convertis en CNF, ils peuvent être écrits sous la forme :

$$P = \bigwedge_i p_i$$

$$Q = \bigwedge_j q_j$$

Une méthode s'occupe d'extraire une liste des sous-arbres p_i et q_j, qui sont des disjonctions de littéraux (par application de la méthode).

Suivant la forme de l'expression ($P \wedge Q$ ou $P \vee Q$), la méthode renvoie un arbre qui est de la bonne forme :

— *Si l'expression est de la forme $P \wedge Q$, alors l'expression renvoyée est*

$$\bigwedge_j q_j \wedge \bigwedge_i p_i$$

— *Sinon, l'expression renvoyée est la distribution des p_i et des q_i suivant la distribution des $\vee$ et des $\wedge$.*

Les méthodes de simplification décrites plus bas sont également appelées lors de la reconstruction de l'arbre final renvoyé.

Méthode 3 *public static Tree* `convert_to_DNF`*(Tree expression)*
Cette méthode convertit une expression quelconque contenant des $\vee$ et des $\wedge$ en DNF.

Méthode 4 *public static Tree* `convert_to_PNF`*(Tree expression)*
Cette méthode convertit une expression quelconque contenant des $\oplus$ et des $\wedge$ en PNF.

4.3.3 Méthodes de simplification

Ces méthodes reposent sur les propriétés des opérateurs utilisés. Elles prennent en entrée un certain nombre d'arbres, qui sont les arguments de l'opérateur, et renvoient un arbre qui est une simplification de l'expression opérateur(argument 1, argument 2, $\cdots$, argument n). Par exemple,

$$\mathrm{med}(0, 0, X),$$

qui est un arbre à trois feuilles, est simplifié en l'arbre suivant :

$$0$$

Les méthodes de simplification testent l'égalité des arbres portés en argument et, suivant les cas, renvoient d'autres arbres simplifiés.

Méthode 5 *public static Tree* `simplify_XOR`*(Tree t1, Tree t2)*
Cette méthode renvoie la simplification de l'expression $t1 \oplus t2$.

Méthode 6 *public static Tree* `simplify_OR`*(Tree t1, Tree t2)*
Cette méthode renvoie la simplification de l'expression $t1 \vee t2$.

Méthode 7 *public static Tree* `simplify_AND`*(Tree t1, Tree t2)*
Cette méthode renvoie la simplification de l'expression $t1 \wedge t2$ *.*

Méthode 8 *public static Tree* `simplifyDegree1`*(Tree t1, Tree t2, Tree t3)*
Cette méthode renvoie la simplification de l'expression $med(t1, t2, t3)$*.*

4.3.4 Méthodes diverses

Cette section contient des méthodes diverses qui sont appelées par les autres méthodes décrites ci-haut.

Méthode 9 *public static List<Tree>* `get_conjunctions_of_literals_from_DNF`*(Tree dnf)*

Les méthodes de ce type servent à extraire, d'une forme donnée, les sous-arbres d'une tête particulière (ici, des conjonctions).
Ces méthodes sont utiles dans les algorithmes de conversion pour extraire les p_i *et* q_j *mentionnés plus hauts.*

Méthode 10 *public static Tree* `get_nested_disjunctions_from_list`*(List < Tree > list_trees)*
Nous avons choisi d'utiliser un même nombre d'arguments pour n'importe quel opérateur : ils en prennent tous deux (sauf la médiane, qui en prend trois).
Par exemple, au lieu d'avoir des arbres de cette forme :

$$\wedge(a, b, c)$$

Nous stockons les données dans un arbre binaire :

$$\wedge(a, \wedge(b, c))$$

Les méthodes du type `get_nested_` $\cdots$ *servent donc à reconstituer un arbre binaire à partir d'une liste d'arguments :*

$$(a, b, c) \rightarrow \wedge(a, \wedge(b, c))$$

Finalement, certaines méthodes servent à effectuer les tests lors de la construction de g_f décrite plus haut (conversion table de vérité $\rightarrow$ forme médiane), comme une méthode pour calculer le poids de Hamming d'un vecteur booléen, ou une autre qui renvoie la valeur écrite de g_f suivant le vecteur placé en entrée.

5 Perspectives et conclusion

5.1 Perspectives

Cette section regroupe des idées d'améliorations futures du programme.

— Tester extensivement le programme pour vérifier la validité des conversions, ou
— Effectuer la preuve que le programme effectue des conversions correctes ;
— Optimiser les algorithmes et les méthodes du programme pour réduire les coûts en mémoire et en temps des conversions ;
— Améliorer les algorithmes de simplification ;
— Convertir le programme Java en applet, et en proposer un accès en ligne ; alternativement, passer du Java au Javascript directement ;
— Augmenter le nombre de variables des fonctions que l'on peut convertir.

5.2 Conclusion

Notre expérience de ce projet tutoré a été globalement très positive. Il nous a permis d'approcher directement un domaine de recherche et d'y participer concrètement, ainsi que de nous initier à des problèmes d'une complexité supérieure à ce que nous avons pu rencontrer jusqu'à présent.

Cette complexité repose en grande partie sur le fait que nous avons touché à un domaine qui nous était peu familier (mathématiques discrètes). En revanche, la formation en informatique qui nous a été dispensée a été amplement suffisante. Notre tuteur s'est également assuré, avant que nous commencions toute implémentation, que nous ayons les connaissances nécessaires pour manipuler les notions utilisées.

Références

[1] Miguel Couceiro, Stephan Foldes, and Erkko Lehtonen. Composition of post classes and normal forms of boolean functions. *Discrete mathematics*, 306(24) :3223–3243, 2006.

[2] Miguel Couceiro, Erkko Lehtonen, Jean-Luc Marichal, and Tamás Waldhauser. An algorithm for producing median formulas for boolean functions. In *Reed Muller 2011 Workshop*, pages 49–54, 2011.

[3] Yves Crama and Peter L Hammer. *Boolean functions : Theory, algorithms, and applications*. Cambridge University Press, 2011.

[4] Jason Eisner. How to convert a formula to cnf. http://www.cs.jhu.edu/~jason/tutorials/convert-to-CNF.html, dernière consultation : Mai 2015.

[5] GraphStream. A dynamic graph library. http://graphstream-project.org/, dernière consultation : Avril 2015.

6 Annexe

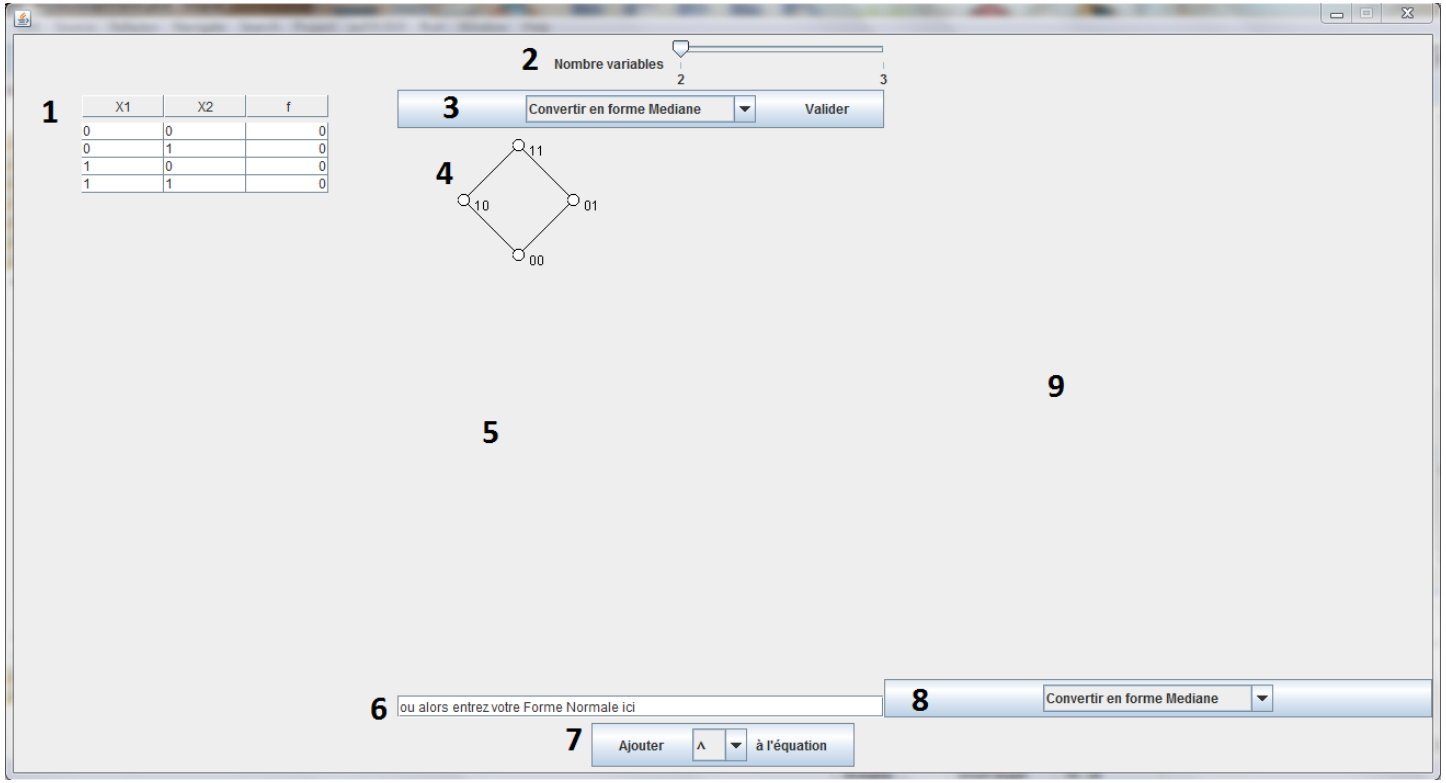


FIGURE 11 – Capture d'écran du programme

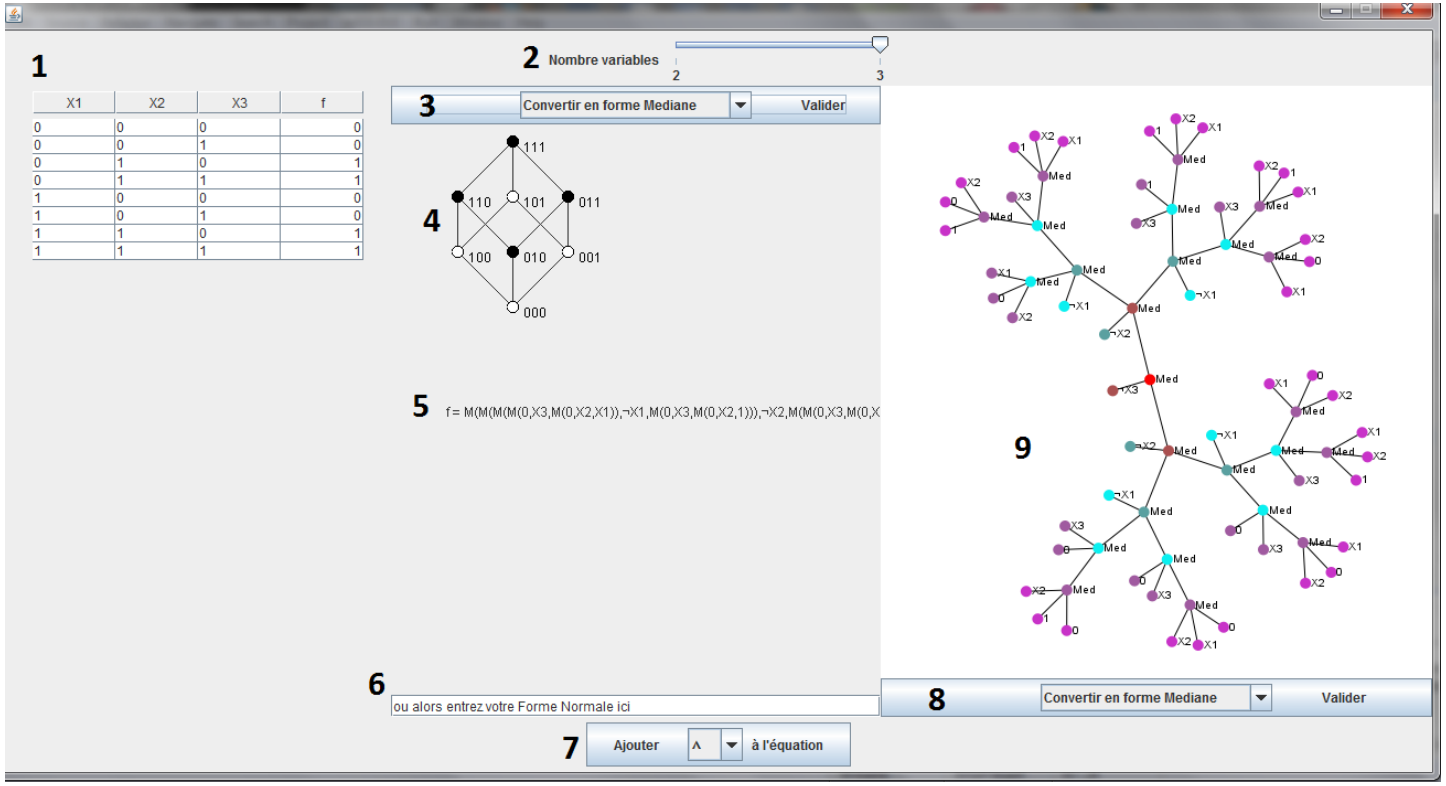


FIGURE 12 – Capture d'écran du programme en cours d'utilisation