Licence d'informatique – première année
Année 2021-2022

Logique et raisonnement
Polycopié du cours

Sommaire

1 Logique propositionnelle

1.1 Introduction

La logique est à la croisée de plusieurs disciplines comme les mathématiques, l'informatique et la philosophie. Elle a des objectifs très variés tels que la résolution de problèmes mathématiques, la modélisation du raisonnement, l'étude des domaines de l'informatique théorique, la formalisation et la conception de langages informatiques.

Voici quelques définitions :

Définition 1.1 *Une* proposition *est un énoncé qui peut soit être vrai soit être faux, mais qui ne peut pas être les deux à la fois.*

Contre-exemples : « Tout nombre réel négatif n'est pas un carré » n'est pas une proposition car cette assertion est vraie dans **R** et fausse dans **C**.
« La proposition que je suis en train d'énoncer est fausse » n'est pas une proposition, car si elle est vraie, alors elle est fausse et inversement.

Définition 1.2 *Un* théorème *est un énoncé toujours vrai.*

Définition 1.3 *Une hypothèse est une proposition que l'on suppose vraie pour démontrer une autre proposition (sous cette hypothèse). Si on se trouve dans l'hémisphère nord (hypothèse) alors l'eau de l'évier s'écoule dans le sens des aiguilles d'une montre.*

Constitution d'une logique
Une logique est composée d'au moins trois parties complémentaires

1. **une syntaxe** qui fournit une façon de construire l'ensemble des formules
2. **une sémantique** qui donne une description de ce que ces formules signifient
3. **un ou plusieurs systèmes de preuve** qui nous permet de calculer la signification des formules en construisant des preuves

Cette section aborde succinctement la syntaxe et la sémantique de la logique propositionnelle, une des logiques les plus simples, mais aussi une des moins expressives. Il existe de nombreux systèmes de preuve, on définira ici quelques règles de réécriture et on en donnera une utilisation pour prouver la validité d'une formule. On verra également comment utiliser la méthode des tableaux pour montrer qu'une formule F est satisfaisable.

1.2 Syntaxe

Soit $V = \{p, q, r, \cdots\}$ un ensemble dénombrable d'éléments appelés variables propositionnelles. On définit $\mathcal{F}_0$ un ensemble de formules du calcul propositionnel sur l'ensemble V de variables propositionnelles récursivement (ou inductivement) avec le schéma d'induction suivante :

i) **base, initialisation :** Toute variable propositionnelle appartient à $\mathcal{F}_0$;

ii) **induction :** Si P et Q appartiennent à $\mathcal{F}_0$, alors (P$\wedge$Q), (P$\vee$Q), (P$\rightarrow$Q) et ($\neg$P) appartiennent à $\mathcal{F}_0$.

$\wedge$, $\vee$, $\rightarrow$ et $\neg$ sont appelés des *connecteurs logiques*. Par la suite et si cela n'entraîne pas d'ambiguïté, nous écrirons simplement *formule* à la place de *formule du calcul propositionnelle*. D'autre part, nous utiliserons les lettres minuscules pour désigner des variables propositionnelles et les lettres majuscules $A, B, C, \cdots$ pour désigner des formules.

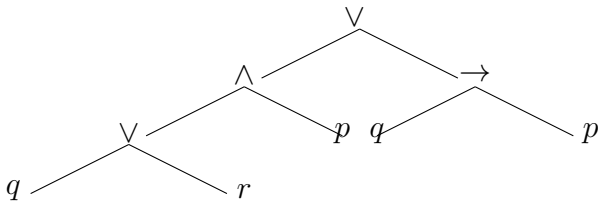
Remarque 1.1 *Notons que d'autres constructions d'ensembles de formules du calcul propositionnel sont possibles. Nous pouvons en effet faire varier le schéma d'induction ci-dessus en considérant, par exemple, d'autres connecteurs logiques comme* $\leftrightarrow$ *et* $\oplus$.

Remarque 1.2 *On retrouve le même schéma de construction pour l'ensemble des formules du calcul algébrique avec les opérateurs* $+, -, /, *$.

Arbre représentant une formule

Toute formule peut être représentée par un arbre. Cette représentation est unique.

Par exemple, la formule $((q \vee r) \wedge p) \vee (q \rightarrow p)$ est représentée par l'arbre suivant



1.3 Sémantique

On appelle *valuation* ou *assignement* l'assignation d'une valeur booléenne à chaque variable propositionnelle. Une valuation est donc une application v de V dans $\{0, 1\}$. A chaque valuation correspond une valeur de la formule (une évaluation de cette formule) qui est déterminée récursivement :

- Si la formule F est une variable propositionnelle p, alors $eval_v(F) = v(p)$.
- Sinon si F est de la forme $\neg A$, $A \wedge B$, $A \vee B$ ou $A \rightarrow B$ et on utilise les définitions suivantes des connecteurs logiques pour calculer $eval_v(F)$.

1. $\top$ et $\bot$ deux connecteurs définis sans variable propositionnelle :

$\top$	1
$\bot$	0

2. **et** (notée $\wedge$ ou parfois $\cdot$ et appelée conjonction ou produit), dont la table de vérité est :

$\wedge$	0	1
0	0	0
1	0	1

soit encore

A	B	A$\wedge$B
0	0	0
1	0	0
0	1	0
1	1	1

3. **ou** (notée $\vee$, ou parfois $+$ et appelée disjonction ou somme), dont la table de vérité est :

$\vee$	0	1
0	0	1
1	1	1

soit encore

A	B	A$\vee$B
0	0	0
1	0	1
0	1	1
1	1	1

4. et l'opération unaire **non**, notée $\neg$ et appelée négation, pour laquelle on utilisera aussi la notation $\overline{\text{A}}$ (négation de A).
5. **L'implication** dont la table de vérité de l'opérateur $\rightarrow$ est la suivante :

$\rightarrow$	0	1
0	1	1
1	0	1

soit encore

A	B	A$\rightarrow$B
0	0	1
1	0	0
0	1	1
1	1	1

On définit également les deux opérateurs suivants.

6. **La double implication** dont la table de vérité de l'opérateur $\leftrightarrow$ est la suivante :

$\leftrightarrow$	0	1
0	1	0
1	0	1

soit encore

A	B	A$\leftrightarrow$B
0	0	1
1	0	0
0	1	0
1	1	1

7. **ou bien** (notée $\oplus$ et appelée disjonction exclusive ou somme exclusive) dont la table de vérité est la suivante :

$\oplus$	0	1
0	0	1
1	1	0

soit encore

A	B	A$\oplus$B
0	0	0
1	0	1
0	1	1
1	1	0

Vocabulaire
Une formule F est une **tautologie** (ou encore une formule valide) si $eval_v(F)$ vaut 1 quelque soit la valuation v (la table de vérité prend toujours la valeur 1).

Une formule F est **satisfaisable** si $eval_v(F)$ vaut 1 pour au moins une valuation v (la table de vérité comporte au moins un 1). Dans le cas contraire, la formule est dite **insatisfaisable** ou **contradictoire**, on dit encore que c'est une **antilogie**. Pour toute formule A, nous avons l'équivalence entre A est contradictoire et $\neg A$ est valide.

Définition 1.4 *Deux formules A et B sont dites* logiquement équivalentes*, ou encore équivalentes, lorsque la formule $A \leftrightarrow B$ est valide. On écrit alors $A \equiv B$.*

Exemples

1. $A \rightarrow B \equiv \neg A \vee B$ et $A \rightarrow B \equiv \neg B \rightarrow \neg A$ (contraposition).
2. $A \leftrightarrow B \equiv A \rightarrow B \wedge \ B \rightarrow A$.

3. $A \oplus B \equiv (A \wedge \neg B) \vee (\neg A \wedge B)$.
4. A est une tautologie si et seulement si $A \equiv \top$.
5. A est une antilogie si et seulement si $A \equiv \bot$.

Quelques propriétés des connecteurs
Pour toutes formules A et B, nous avons :

1. les trois connecteurs $\wedge$, $\vee$ et $\oplus$ sont commutatifs : $A \wedge B \equiv B \wedge A$.
2. les trois connecteurs $\wedge$, $\vee$ et $\oplus$ sont associatifs : $(A \wedge B) \wedge C = A \wedge (B \wedge C)$.
3. $\top$ est l'élément neutre de $\wedge$: $\top \wedge A \equiv A \wedge \top \equiv A$.
4. $\bot$ est l'élément neutre de $\vee$ et $\oplus$: $\bot \vee A \equiv A \vee \bot \equiv A$.
5. $\wedge$ et $\vee$ sont idempotentes : $A \wedge A \equiv A$ et $A \vee A \equiv A$.
6. $\bot$ est l'élément absorbant de $\wedge$: $\bot \wedge A \equiv \bot$
7. $\top$ est l'élément absorbant de $\vee$: $\top \vee A \equiv \top$.

Définition 1.5 *Ensemble complet de connecteurs*
Un ensemble de connecteurs est dit complet *lorsque toute formule est équivalente à une formule n'utilisant que les connecteurs de cet ensemble.*

1.4 Règles de réécriture

Définition 1.6 *Si A et B sont deux formules logiquement équivalentes, on notera $A \rightsquigarrow B$ la* règle de réécriture *qui remplace dans une formule quelconque une occurrence de A par B. A chaque règle $A \rightsquigarrow B$ correspond évidemment la règle inverse $B \rightsquigarrow A$.*

Quelques règles de réécriture
Les propriétés des connecteurs logiques énoncés ci-dessus permettent de définir de nombreuses règles de réécriture, comme A $\wedge$ B $\rightsquigarrow$ B $\wedge$ A, ...

Soient A, B et C trois formules quelconques. Elles vérifient les règles de réécriture suivantes (à retenir) :

1. A $\rightarrow$ B $\rightsquigarrow$ $\neg$ A$\vee$B suppression de l'implication
2. $\neg\neg$ A $\rightsquigarrow$ A suppression de la double négation
3. Lois de De Morgan
   (a) $\neg$ (A$\wedge$B) $\rightsquigarrow$ $\neg$ A$\vee\neg$B
   (b) $\neg$ (A$\vee$B) $\rightsquigarrow$ $\neg$ A$\wedge\neg$ B
4. Distributivité
   (a) distributivité de $\vee$ par rapport à $\wedge$: A$\vee$(B$\wedge$C) $\rightsquigarrow$ (A$\vee$B)$\wedge$(A$\vee$C)
   (b) distributivité de $\wedge$ par rapport à $\vee$: A$\wedge$(B$\vee$C) $\rightsquigarrow$ (A$\wedge$B)$\vee$(A$\wedge$C)
   (c) distributivité de $\wedge$ par rapport à $\oplus$: A$\wedge$(B$\oplus$C) $\rightsquigarrow$ (A$\wedge$B)$\oplus$(A$\wedge$C)

Exemple d'application Les règles 1, 2 et 3(a) (resp. 1, 2 et 3(b)) permettent de montrer que $(\neg, \vee)$ (resp. $(\neg, \wedge)$) forme un ensemble complet de connecteurs.

Remarque 1.3 *Pour résoudre certains problèmes de décision, comme celui de la validité, l'utilisation judicieuse de règles de réécriture s'avère souvent plus efficace que les tableaux de vérité car un tableau de n variables propositionnelles comporte 2^n lignes.*

1.5 Formes normales conjonctives et disjonctives

Il existe plusieurs façons d'écrire une même formule logique. Par exemple, les formules $p \to q$, $\neg p \vee q$ et $\neg(p \wedge \neg q)$ sont logiquement équivalentes. Il peut être intéressant pour la manipulation des formules d'utiliser une forme particulière.

Définitions 1.1

- Un littéral est une variable propositionnelle ou la négation d'une variable propositionnelle.
- Une formule est en forme normale conjonctive (FNC) si elle est la conjonction de disjonctions de littéraux.
- Une formule est en forme normale disjonctive (FND) si elle est disjonction de conjonctions de littéraux.

Exemples $F = (\neg p \vee q) \wedge \big((p \vee q) \vee \neg r\big)$ est en FNC et $G = \big((p \wedge q) \vee (\neg p \wedge \neg r)\big) \vee (\neg q \wedge r)$ est en FND.

Théorème 1.1 *Pour toute formule A du calcul propositionnel, il existe une formule B en FNC et une formule C en FND logiquement équivalentes à A.*

Nous verrons à la section suivante que si nous connaissons la table de vérité de A alors nous pouvons en déduire une formule B en FNC et une formule C en FND particulières. Supposons ici que la table de vérité n'est pas connue. L'utilisation de règles de réécriture sur A pour arriver à B ou à C peut s'avérer beaucoup plus efficace (en temps, mais aussi en mémoire) que de calculer la table de vérité.

Idée de la preuve : On traitera uniquement la mise sous FNC, la mise sous FND étant similaire. L'algorithme suivant permet de trouver une formule B en FNC. B n'est pas unique, il y a généralement plusieurs formes possibles.

1. On élimine de A tous les connecteurs autres que $\vee, \wedge, \neg$. Par exemple, il suffit d'utiliser la règle 1 pour éliminer le symbole $\to$.
2. On applique autant que possible les règles 2, 3(a) et 3(b).
3. On applique autant que possible la règle 4(a).

Pour prouver que cet algorithme convient, il faut vérifier les trois points suivants.

a) L'algorithme se termine.
b) La formule obtenue est en FNC.
c) Elle est équivalente à A.

Le dernier point est évident, car on remplace à chaque fois une formule par une formule équivalente. Pour une démonstration rigoureuse, les deux autres points nécessitent une induction sur les arbres représentant les formules intermédiaires.

Remarque Contrairement à la mise sous forme normale canonique à partir de la table de vérité (voir ci-dessous), la solution n'est pas forcément unique aussi bien pour la FND que pour la FNC.

Définition 1.7 *Une clause est une disjonction de littéraux. Une formule sous FNC est donc une conjonction de clauses.*

FNC et validité La mise sous FNC peut servir à montrer qu'une formule est valide. Pour cela, on utilise le théorème suivant :

Théorème 1.2 *Une formule en FNC est valide si et seulement si toute clause contient deux littéraux contradictoires, autrement dit chaque clause est de la forme*

$$l_1 \vee \ldots p \vee \ldots \vee \neg p \vee \ldots \vee l_n.$$

Codage d'une valuation On peut représenter une valuation par un mot binaire. Pour cela, on écrit les variables propositionnelles dans un ordre fixé à l'avance ; le mot binaire correspondant à une valuation est la suite des valeurs des variables propositionnelles, prises dans cet ordre. À toute formule du calcul propositionnel, on peut alors faire correspondre une *fonction booléenne.*

Formes normales conjonctives canoniques et formes normales disjonctives canoniques
Nous pouvons obtenir à partir de la table de vérité de A une FND particulière que nous appelerons FND canonique. Nous pouvons également obtenir une FNC canonique.

Définition 1.8 *On appelle monôme une conjonction non nulle de littéraux dans lesquels on ne trouve pas à la fois une variable et sa négation.*

Le degré d'un mônome est le nombre des littéraux dont il est le produit.
Exemples $p \wedge \neg q \wedge r \wedge \neg t$ est un monôme de degré 4 et $\neg p \wedge r \wedge s$ est un monôme de degré 3.

On considère une formule F fixée. Nous allons définir les mintermes et les maxtermes de F en fonction des variables propositionnelles contenues dans F.

Définition 1.9 *Un minterme est une conjonction de littéraux où chaque variable propositionnelle apparaît une seule fois.*

Définition 1.10 *Un maxterme est une disjonction de littéraux où chaque variable propositionnelle apparaît une seule fois.*

Supposons que F contienne n variables propositionnelles. Un minterme est alors un monôme de degré n et chaque ligne de la table de vérité correspond à un de ces mintermes.

Soient $p_1, \ldots, p_n$ les variables propositionnelles de F. Un minterme est de la forme $l_1 \wedge l_2 \wedge \ldots \wedge l_n$ où, pour tout $i \in \{1, \ldots, n\}$, soit $l_i = p_i$, soit $\neg p_i$.

La valuation v correspondante vérifie
— $v(p_i) = 1$ lorsque $l_i = p_i$
— $v(p_i) = 0$ lorsque $l_i = \neg p_i$

Nous avons ainsi 2^n mintermes et 2^n maxtermes pour une formule contenant n variables propositionnelles.
Chaque maxterme peut être construit en prenant la négation d'un minterme.
Exemple Supposons que notre formule F contienne les trois variables propositionnelles p, q et r. Les $2^3 = 8$ mintermes et les 8 maxtermes sont :

ligne	p	q	r	minterme	maxterme
1	0	0	0	$\neg p \wedge \neg q \wedge \neg r$	$p \vee q \vee r$
2	1	0	0	$p \wedge \neg q \wedge \neg r$	$\neg p \vee q \vee r$
3	0	1	0	$\neg p \wedge q \wedge \neg r$	$p \vee \neg q \vee r$
4	1	1	0	$p \wedge q \wedge \neg r$	$\neg p \vee \neg q \vee r$
5	0	0	1	$\neg p \wedge \neg q \wedge r$	$p \vee q \vee \neg r$
6	1	0	1	$p \wedge \neg q \wedge r$	$\neg p \vee q \vee \neg r$
7	0	1	1	$\neg p \wedge q \wedge r$	$p \vee \neg q \vee \neg r$
8	1	1	1	$p \wedge q \wedge r$	$\neg p \vee \neg q \vee \neg r$

Supposons que F ait la table de vérité suivante :

ligne	p	q	r	F	minterme	maxterme
1	0	0	0	1	$\neg p \wedge \neg q \wedge \neg r$	
2	1	0	0	0		$\neg p \vee q \vee r$
3	0	1	0	0		$p \vee \neg q \vee r$
4	1	1	0	0		$\neg p \vee \neg q \vee r$
5	0	0	1	1	$\neg p \wedge \neg q \wedge r$	
6	1	0	1	1	$p \wedge \neg q \wedge r$	
7	0	1	1	0		$p \vee \neg q \vee \neg r$
8	1	1	1	0		$\neg p \vee \neg q \vee \neg r$

Pour avoir la formule C en FND canonique, on forme la disjonction des mintermes correspondant aux valuations v pour lesquelles $eval_v(F) = 1$. On obtient ainsi une formule logiquement équivalente à F car la formule est vraie uniquement pour ces valuations.

$$C = \overset{1}{(\neg p \wedge \neg q \wedge \neg r)} \vee \overset{5}{(\neg p \wedge \neg q \wedge r)} \vee \overset{6}{(p \wedge \neg q \wedge r)}$$

Pour avoir la formule D en FNC canonique, on forme la conjonction des maxtermes correspondant aux valuations v pour lesquelles $eval_v(F) = 0$. On obtient ainsi une formule logiquement équivalente à F car la formule est la formule est fausse uniquement pour ces valuations.

$$B = \overset{2}{(\neg p \vee q \vee r)} \wedge \overset{3}{(p \vee \neg q \vee r)} \wedge \overset{4}{(\neg p \vee \neg q \vee r)} \wedge \overset{7}{(p \vee \neg q \vee \neg r)} \wedge \overset{8}{(\neg p \vee \neg q \vee \neg r)}$$

Contrairement aux FND et FNC, les FND et FNC canoniques sont uniques.

Remarque 1.4 *B peut également s'obtenir facilement en prenant la négation de la FNC canonique de $\neg F$.*

1.6 Problème SAT

Rappelons qu'une formule F sous FNC est une conjonction de clauses $C_1, \ldots, C_k$.
Le problème SAT s'énonce ainsi : étant donnée une formule F sous FNC, F est-elle satisfaisable ? c'est-à-dire existe-t-il un assignement qui satisfasse toutes les clauses C_i, $i = 1 \ldots k$?

Une instance est généralement codée comme un ensemble de clauses $I = \{C_1, \ldots, C_k\}$ qui doivent être toutes satisfaites simultanément par au moins une valuation.

Il existe plusieurs simplifications possibles d'une instance SAT.

1. Nous pouvons supprimer les doublons (un littéral qui apparaît plus d'une fois).
2. Si une clause contient une variable propositionnelle p et sa négation $\neg p$, alors cette clause sera satisfaite quelle que soit la valuation. Nous pouvons donc supprimer cette clause.
3. Soient C_i et C_j deux clauses de I. On note $C_i < C_j$ lorsque tout littéral de C_i apparaît dans C_j, c'est-à-dire $C_i = l_1 \vee \dots l_a$ et $C_j = l_1 \vee \dots l_a \vee \dots l_b$, où a et b sont deux entiers avec $a < b$. Si $C_i < C_j$, alors toute valuation satisfaisant C_i satisfait également C_j. Nous pouvons donc supprimer la clause C_j.

Une instance SAT est valide si et seulement si nous pouvons supprimer toutes les clauses.

Remarque 1.5 *Il existe de nombreux algorithmes qui permettent de montrer qu'une instance SAT est satisfaisable. Le plus célèbre est l'algorithme de Davis-Putman-Logemann–Loveland (DPLL) qui est un algorithme de backtracking.*

1.7 Méthode des tableaux

La méthode des tableaux permet de montrer qu'une formule F est satisfaisable ou non.

A partir de F, on construit un arbre binaire $T(F)$. Les feuilles de l'arbre seront dites *ouvertes* ou *fermées*. La formule sera satisfaisable lorsqu'elle possèdera au moins une feuille ouverte.

Soit F une formule de $\mathcal{F}_0$. Par définition de $\mathcal{F}_0$, si F n'est pas un littéral alors F admet une des décompositions suivantes

1. $F = A \vee B$
2. $F = A \wedge B$
3. $F = A \to B$
4. $F = \neg(A \vee B)$
5. $F = \neg(A \wedge B)$
6. $F = \neg(A \to B)$
7. $F = \neg(\neg A)$

Un ensemble de formule $\{F_1, \dots, F_n\}$ sera noté $F_1, \dots, F_n$.

La racine de l'arbre sera F. Ensuite, pour chaque noeud N de l'arbre, on construit les fils de la manière suivante :

où E est un ensemble de formules. E est l'ensemble vide au départ.

L'algorithme ci-dessus n'est pas déterministe car il peut exister plusieurs façons d'appliquer les deux règles. Mais on peut le rendre déterministe en imposant un ordre dans l'application des règles.

On montre que lorsque l'algorithme se finit, les feuilles de l'arbre sont des ensembles de littéraux.

Une feuille sera appelée *fermée* si elle contient un littéral et sa négation (p et $\neg p$, pour une variable propositionnelle p). Dans le cas contraire, elle sera appelée *ouverte*.

Théorème 1.3 *F est satisfaisable si et seulement si elle possède une feuille ouverte.*

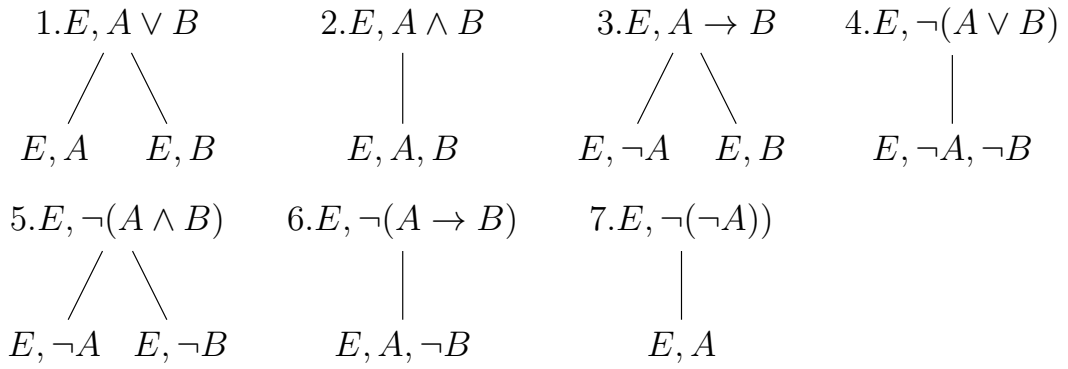


FIGURE 1 – Méthode des tableaux

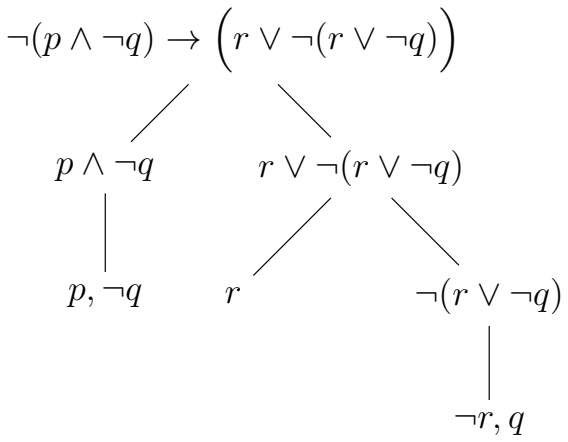


FIGURE 2 – Méthode des tableaux – exemple

Dans l'exemple précédent, les trois feuilles de F sont ouvertes, donc en appliquant le théorème on en déduit que F est satisfaisable.

Dire qu'une formule est insatisfaisable est équivalent à dire que sa négation est valide. Donc la méthode des tableaux est une méthode pour prouver qu'une formule est satisfaisable ou qu'une formule est valide en partant de sa négation pour construire l'arbre (F est valide si $\neg F$ n'est pas satisfaisable, c'est-à-dire si $\neg F$ ne possède pas de feuille ouverte).

2 Logique des prédicats

La logique propositionnelle s'avère trop limitée dans de nombreux cas, en particulier lorsque l'on souhaite pouvoir décomposer la variable propositionnelle. Prenons la proposition « Paul habite à Caen ». En logique des prédicats, on sépare cette proposition en un sujet (*Paul*) et un prédicat (*habite à Caen*). Le sujet (*Paul*) peut être remplacé par une variable x appelée variable individuelle, la proposition devient « x habite à Caen » et il est possible de considérer différents sujets comme valeur de x.

La notion de prédicat remonte à l'antiquité avec des philosophes comme Aristote. Cependant c'est le mathématicien et philosophe Gottlob Frege qui a formalisé la logique des prédicats

du premier ordre à la fin du 19ème siècle.

On définit la logique des prédicats du premier ordre en utilisant les quantificateurs $\exists$ (il existe) et $\forall$ (quel que soit). Soit $G = \{Pierre, Paul, Jacques\}$ un groupe de copains. $\exists x \in G$ « x habite à Paris », signifie qu'une des personnes de G habite à Paris, ce qui correspond à « Pierre habite à Paris ou Paul habite à Paris ou Jacques habite à Paris ».

D'autre part, $\forall x \in G$ « x habite à Paris », signifie que toutes les personnes de G habite à Paris, ce qui correspond à « Pierre habite à Paris et Paul habite à Paris et Jacques habite à Paris ».

Les prédicats font intervenir des relations. Nous allons donner quelques définitions générales concernant celles-ci.

2.1 Relations

Définition générale

Définition 2.1 Soit E un ensemble et $k \in N^*$. Une relation R d'arité k sur E est un sous-ensemble de E^k (produit cartésien de k fois l'ensemble E). On peut également voir R comme une application de E^k vers $\{0, 1\}$. Les deux assertions suivantes sont alors équivalentes :

1. $(e_1, \ldots, e_k) \in R$.
2. $R(e_1, \ldots, e_k) = 1$.

E est souvent appelé le domaine ou l'univers.

Remarque Une relation est un sous-ensemble, comme tout sous-ensemble elle peut être définie en extension (on donne la liste des k-uplets) ou en compréhension (les k-uplets sont caractérisés par une propriété donnée, par exemple, par une équation mathématique).

Exemples Soient S et P les relations ternaires (d'arité 3) sur $\mathbb{Z}$ définies par

$$\begin{aligned} S &= \{(x, y, z) \in \mathbb{Z}^3 \mid x + y = z\} \\ P &= \{(x, y, z) \in \mathbb{Z}^3 \mid x * y = z\} \end{aligned}$$

Nous avons, par exemple, $(2, 3, 5) \in S$ et $(4, 5, 20) \in P$.
On montre que l'intersection des deux relations est $S \cap P = \{(0, 0, 0), (2, 2, 4)\}$.

Définition 2.2 Relation unaire (arité 1)
Une relation unaire U sur un ensemble E est un sous-ensemble de E. Par exemple, $\mathcal{P}air$ – l'ensemble des entiers pairs – et $\mathcal{I}mpair$ – l'ensemble des entiers impairs – sont deux relations unaires sur $\mathbb{Z}$.

Définition 2.3 Relation binaire (arité 2)
Une relation binaire R sur un ensemble E est un sous-ensemble de $E \times E$.

On retrouve très souvent des relations binaires lorsque l'on étudie des ensembles en mathématiques ou en informatique, elles interviennent de manière naturelle. Par exemple, pour un ensemble de personnes P, nous pouvons définir la relation $H(x, y)$ avec $H(x, y) = 1$ lorsque x et y habitent la même ville ou encore la relation $C(x, y)$ avec $C(x, y) = 1$ lorsque x et y se connaissent.

2.2 Logique des prédicats du premier ordre

Les prédicats sont donc définis par des relations dont l'arité dépend du nombre de variables et de constantes. Par exemple « x habite à Paris » peut être défini par une relation unaire $H_P(x)$. Cependant cela nécessite d'avoir une relation unaire pour chaque ville. Il est donc préférable de définir une relation binaire H avec $H(x, Caen)$, où Caen est une constante. Nous pouvons faire varier à la fois la personne et sa ville. Soit $V = \{Caen, Cherbourg, Lisieux\}$ un ensemble de villes. Reprenons le groupe de copains $G = \{Pierre, Paul, Jacques\}$. Nous pouvons définir la formule

$$\forall x \in G \; \exists y \in V \; H(x, y),$$

qui signifie que toute personne de G habite dans une des villes de V.

Nous distinguons deux sortes de formules, les formules qui possèdent au moins une variable libre (non quantifiée) – comme les formules $H(x, Caen)$ et $\exists y \in V \; H(x, y)$ où x est une variable libre – et les formules closes dont toutes les variables sont liées (quantifiées) – comme la formule $H(Pierre, Caen)$ qui ne possède pas de variable et la formule $\exists y \in V \; H(Pierre, y)$ qui a ses deux variables liées. Seules les formules closes prennent une valeur de vérité.

On définit inductivement les formules de la logique des prédicats du premier ordre

1. $R(p_1, \ldots, p_k)$ est une formule, où R est une relation d'arité k et $p_1, \ldots, p_k$ sont des constantes ou des variables.
2. Soient F_1 et F_2 deux formules, $\neg F_1$, $F_1 \vee F_2$ et $F_1 \wedge F_2$ sont des formules. (On peut ajouter d'autres connecteurs comme $\rightarrow$ et $\leftrightarrow$).
3. Soit F une formule où x est une variable libre, $\exists x \; F$ et $\forall x \; F$ sont des formules.

Remarque 2.1 *La quantification $\exists x \; F$ et $\forall x \; F$ s'effectue sans préciser les ensembles sur lesquels porte la variable x. Lorsqu'un seul ensemble est en jeu, il n'est pas nécessaire de le préciser.*

Lorsqu'il y a plusieurs ensembles, ces ensembles sont en fait déterminés par une relation unaire. La formule $\forall x \in G \; \exists y \in V \; H(x, y)$ devrait donc s'écrire $\forall x \exists y \; G(x) \rightarrow (V(y) \wedge H(x, y))$. Dans les exercices, vous pourrez utiliser les deux écritures.

Exemples

1. Il existe un nombre plus petit que tous les autres.

   $$\exists x \forall y \; x \leq y.$$

   Notons que cette formule est vraie sur $\mathbb{N}$ et sur tout ensemble fini, mais elle est fausse sur les domaines $\mathbb{Z}$ et $\mathbb{R}$.

2. Tout quadrilatère qui est un carré est un rectangle, mais un rectangle peut ne pas être un carré.

   $$\Big(\forall x\big(Carre(x) \rightarrow Rectangle(x)\big)\Big) \wedge \Big(\exists x\big(Rectangle(x) \wedge \neg Carre(x)\big)\Big).$$

3. Tous les étudiants se sont mis en binôme pour rendre le devoir.

   $$\forall x \exists y \forall z \; \Big(x \neq y \wedge ensemble(x, y) \wedge \big(ensemble(x, z) \rightarrow z = y\big)\Big).$$

   où ensemble(x, y) est un prédicat qui est vrai lorsque x et y ont rendu le devoir ensemble.

3 Démonstration par Récurrence

3.1 Sommes et produits

Définition 3.1 *Soient m et n deux entiers naturels tels que $m \leq n$. Nous noterons $\sum_{k=m}^{k=n} a_k$, $\sum_{k=m}^{n} a_k$ ou $\sum_{m\leq k\leq n} a_k$ la somme $a_m + a_{m+1} + \ldots + a_{n-1} + a_n$.*

Remarques 3.1 *Cette somme comporte $n - m + 1$ termes.*
Si $n < m$ alors cette somme est nulle.
La variable k est muette : $\sum_{k=m}^{n} a_k = \sum_{p=m}^{n} a_p$

Propriété 3.1 *La somme est linéaire*

$$\sum_{k=m}^{n}(\alpha a_k + \beta b_k) = \alpha \sum_{k=m}^{n} a_k + \beta \sum_{k=m}^{n} b_k.$$

Mais en général

$$\sum_{k=m}^{n} a_k b_k \neq (\sum_{k=m}^{n} a_k)(\sum_{k=m}^{n} b_k).$$

Définition 3.2 *On appelle somme télescopique toute somme du type $\sum_{k=m}^{n}(a_{k+1} - a_k)$. Cette somme vérifie $\sum_{k=m}^{n}(a_{k+1} - a_k) = a_{n+1} - a_m$*

Propriété 3.2 *nous avons les deux égalités suivantes (à retenir)*

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}.$$

Soit $a \neq 1$,

$$\sum_{k=0}^{n} a^k = 1 + a + a^2 + \ldots + a^k + \ldots + a^n = \frac{1 - a^{n+1}}{1 - a}.$$

Définition 3.3 *Soient m et n deux entiers naturels tels que $m \leq n$. Nous noterons $\prod_{k=m}^{k=n} a_k$, $\prod_{k=m}^{n} a_k$ ou $\prod_{m\leq k\leq n} a_k$ le produit $a_m \cdot a_{m+1} \cdots a_{n-1} \cdot a_n$.*

Remarques 3.2 *Ce produit comporte $n - m + 1$ facteurs.*
Si $n < m$ alors ce produit vaut 1.

Propriété 3.3

$$\prod_{k=1}^{n} \lambda a_k = (\lambda)^n \prod_{k=1}^{n} a_k \text{ et } \prod_{k=m}^{n} a_k b_k = (\prod_{k=m}^{n} a_k)(\prod_{k=m}^{n} b_k).$$

3.2 Types de raisonnement

Le raisonnement par récurrence est l'une des méthodes mathématiques de démonstration les plus utilisées. Particulièrement efficace (souvent beaucoup plus facile à appliquer que les autres méthodes pour un problème donné), cette méthode ne peut s'appliquer que sous certaines conditions :

- le résultat à démontrer doit être explicité (impossible d'utiliser cette méthode pour chercher en même temps à formuler et à démontrer un résultat) ;
- ce résultat doit porter sur des nombres entiers ou, plus généralement, sur des structures séquentielles ou récursives (suites, mots, arbres, graphes,...).

Rappelons brièvement ce que sont les (autres) grands principes de démonstrations.

Pour démontrer A⇒ B, on peut opter pour :

1. Une démonstration par déduction :
   Faire l'hypothèse que A est vraie et utiliser des propositions déjà connues pour déduire B.
2. Une démonstration par contraposition :
   Démontrer non(B) ⇒ non(A) (car cela équivaut à montrer A ⇒ B, voir le chapitre *logique propositionnelle*).
3. Une démonstration par l'absurde :
   Faire l'hypothèse que A est vraie et B fausse et montrer qu'alors une proposition connue devient fausse.

3.3 Démonstration par récurrence

C'est une méthode pour démontrer des propositions du type :

$$\forall n \in \mathbb{N}\ P(n)$$

ou plus généralement :

$$\forall\, n \geq n_0\ P(n)$$

où P est une propriété du nombre entier générique n (propriété relative à n, à une expression dans laquelle figure n, etc...).

Le principe de démonstration par récurrence est le suivant :

(1) On vérifie $P(0)$ (plus généralement $P(n_0)$)

(2) On suppose que $P(n)$ (l'hypothèse de récurrence) est vraie pour n, n étant supérieur ou égal à n_0) et on montre que sous cette hypothèse, $P(n+1)$ est encore vérifiée.

Exemple Soit à montrer, pour tout $n \geq 1$:

$$P(n) :\ 1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

(1) $\sum_{k=1}^{1} k = 1$ et $\frac{1*(2)}{2} = 1$ donc $P(1)$ est vraie.

(2) On suppose que $n \geq 1$ et $1 + 2 + \cdots + n = \dfrac{n(n+1)}{2}$.

Alors on a :

$$1 + 2 + \cdots + n + (n+1) = \frac{n(n+1)}{2} + (n+1) = \frac{(n+2)(n+1)}{2} = \frac{(n+1)(n+2)}{2}$$

et $P(n+1)$ est vérifiée.

On peut alors conclure d'après le principe de récurrence que $P(n)$ est vraie pour tout $n \geq 1$.

3.4 Variantes de la démonstration par récurrence (exemples)

Variante 1

(1) On vérifie $P(n_0)$;

(2) On suppose que $n \geq n_0$ et que $P(n_0)$, $P(n_0+1)$, $\cdots P(n)$ sont toutes vraies ; on montre que, sous cette hypothèse, $P(n+1)$ est encore vraie.

Cette variante revient en fait à faire une démonstration par récurrence classique, avec comme hypothèse de récurrence le prédicat suivant, défini pour $n \geq n_0$:

$$Q(n) = P(n_0) \wedge P(n_0+1) \wedge \cdots \wedge P(n).$$

En effet, on vérifie bien que :

(1) $Q(n_0)$ est vraie, puisque $P(n_0)$ l'est ;

(2) Si $Q(n)$ est vraie pour $n \geq n_0$, alors $Q(n+1)$ est encore vraie, puisque $Q(n+1)$ équivaut à $Q(n) \wedge P(n+1)$.

Variante 2

(1) On vérifie $P(1)$;

(2) On suppose que $n \geq 1$ et $P(n)$ est vraie ; on montre que, sous cette hypothèse, $P(2n)$ est encore vraie.

(c) On suppose que $n \geq 2$ et $P(n)$ est vraie ; on montre que, sous cette hypothèse, $P(n-1)$ est encore vraie.

Cette variante revient à faire une démonstration par récurrence classique, avec comme hypothèse de récurrence le prédicat suivant, défini pour $n \geq 1$:

$$Q(n) = P(1) \wedge \cdots \wedge P(2^n).$$

Variante 3

(1) On vérifie $P(1)$, $P(2)$, $P(3)$ et $P(4)$;

(2) On suppose que $n \geq 1$ et $P(n)$; on montre que, sous cette hypothèse, $P(n+4)$ est encore vraie.

Cette variante revient à faire une démonstration par récurrence classique, avec comme hypothèse de récurrence le prédicat suivant, défini pour $n \geq 0$:

$$Q(n) = P(4n+1) \wedge P(4n+2) \wedge P(4n+3) \wedge P(4n+4).$$

4 Induction structurale

On peut généraliser le principe des preuves par récurrence à des ensembles autres que $\mathbb{N}$. Il suffit que ces ensembles soient inductifs. Nous allons illustrer le principe d'induction structural sur les ensembles de mots, mais les mêmes constructions sont possibles pour toutes structures séquentielles comme les arbres, les graphes, les formules en logique.

4.1 Ensembles définis inductivement

Les ensembles définis inductivement sont des ensembles pour lesquels nous avons un moyen de les construire en utilisant des éléments de base (appelés également atomes) et des constructeurs.

Soit E un ensemble. On défini inductivement $F \subset E$ en se donnant des règles de construction des éléments de F, règles que l'on sépare en deux types de règles :

i) les règles de bases ou atomes qui indiquent les éléments de base qui sont dans F.

ii) les règles inductives qui donnent un moyen de construire les éléments de F à partir de ceux déjà construits.

Exemple Soit $\mathcal{A} = \{a, b\}$. On définit $E = \mathcal{A}^*$ l'ensemble des mots sur l'alphabet $\mathcal{A}$ qui est défini par le schéma d'induction :

i) Le mot vide (noté ε) appartient à E.

ii) Soit m un mot de E. Les mots ma et mb appartiennent à E.

Ce schéma d'induction permet la construction de tous les mots contenant les symboles a et b, $E = \{\epsilon, a, b, aa, ab, ba, bb, \ldots\}$. Notons que ce shéma d'induction n'est pas unique, nous aurions pu écrire pour le ii)

ii) Soit m un mot de E. Les mots am et bm appartiennent à E.

On définit maintenant $F \subset E$ par le schéma d'induction

i) Le mot a appartient à F.

ii) Soit m un mot de E. mb appartient à F.

F est l'ensemble des mots commençant par a et suivi de b, $F = \{a, ab, abb, abbb, \ldots\}$.

De manière générale, on définit $F \subset E$ inductivement en fixant $B \subset E$, l'ensemble des atomes et en associant à chaque règle inductive une fonction d'arité k $f(x_1, \ldots, x_k)$ renvoyant un élément y de F, où $x_1, \ldots, x_k$ sont des éléments de F.

Soit $\mathcal{F}$ l'ensemble de ces fonctions.

On définit $F \subset E$ avec le schéma d'induction suivant

i) Tout m de B appartient à F.

ii) Soit $f \in \mathcal{F}$ d'arité k, Si $x_1, \ldots, x_k \in F$, alors $f(x_1, \ldots, x_k) \in F$.

Pour le schéma précédent, nous avions une seule fonction $f(m) = mb$.

4.2 Preuve par induction structurelle

Il s'agit d'une généralisation de la preuve par récurrence.

Soit F un ensemble défini inductivement. On veut démontrer qu'une propriété P est vraie pour tout élément de F, autrement dit

$$\forall x \in F \; P(x).$$

Pour cela, on montre que les deux conditions suivantes sont vérifiées :

i) base : $P(x)$ est vraie pour tout $x \in B$

ii) induction : soient $f \in \mathcal{F}$ d'arité k et $x_1, \ldots, x_k \in F$.
Supposons $P(x_1), \ldots, P(x_k)$ vraies, alors $P(f(x_1, \ldots, x_k))$ est vraie.

Exemple Reprenons $\mathcal{A} = \{a, b\}$ et $E = \mathcal{A}^*$. on construit G avec le schéma suivant :
i) le mot vide appartient à G
ii) Si $m \in G$ alors $amb \in G$ et $bma \in G$.
On veut montrer que tout mot de G possède autant de a que de b

On note $n_a(m)$ (resp. $n_b(m)$) le nombre d'occurrences de la lettre a (resp. b) dans le mot m.
Soit $P(m)$ la propriété $n_a(m) = n_b(m)$.

i) base : si $m = \varepsilon$ alors $n_a(m) = n_b(m) = 0$ donc $P(m)$ est vraie.
ii) induction : Soit $m \in D$, supposons $P(m)$ vraie.
Soient $m_1 = amb$ et $m_2 = bma$. $n_a(m_1) = 1 + n_a(m)$ et $n_b(m_1) = 1 + n_b(m)$.
Par hypothèse d'induction $n_a(m) = n_b(m)$, donc $n_a(m_1) = n_b(m_1)$.
Par conséquent $P(m_1)$ est vraie.
De la même manière, on montre que $n_a(m_2) = n_b(m_2)$ et donc $P(m_2)$ est vérifiée.
Nous avons donc montrer par induction que $P(m)$ est vraie pour tout $m \in G$.

5 Fonctions Booléennes sur $\{0,1\}^n$

5.1 Définition et codage d'une fonction booléenne

Les fonctions booléennes sont utilisées dans de nombreux domaines de l'informatique comme la cryptographie, les codes correcteurs, l'intelligence artificielle, la complexité et dans d'autres disciplines comme l'électronique.

Une fonction booléenne à n variables est une fonction prenant la valeur 0 ou 1 lorsque l'on fixe chacune de ses variables à 0 ou 1. On appelle valuation le n-uplet ainsi obtenu.

Il existe beaucoup de similarités avec le calcul propositionnel. Nous avons des valuations, des connecteurs logiques, des tables de vérité, des écritures différentes. Les différences portent plus sur l'utilisation des fonctions booléeennes par rapport à l'utilisation des formules du calcul propositionnel.

Les questions que nous nous posions en calcul propositionnel étaient : la formule est-elle satisfaisable ? Est-elle une tautologie, une antilogie ?
Ici deux critères sont primordiaux : le poids de Hamming et le degré algébrique. À partir de ces deux critères nous pouvons définir un grand nombre de propriétés (nous n'en verrons que quelques unes).

Valuations
Pour une fonction à n variables, les variables seront toujours notées $x_1, \ldots, x_n$. Par exemple, pour $n = 2$, nous avons deux variables x_1 et x_2 et les valuations possibles (les valeurs prises par ces deux variables) sont $(0,0)$, $(0,1)$, $(1,0)$ et $(1,1)$. $(0,1)$ signifie que x_1 prend la valeur 0 et x_2 la valeur 1.

L'ordre des variables est important lors des études effectuées sur ces fonctions, alors qu'il n'y avait pas d'ordre pour les variables propositionnelles.

Connecteurs logiques

En calcul propositionnel, les connecteurs logiques les plus importants sont la négation (noté $\neg$), le **et** (noté $\wedge$) et le **ou** (noté $\vee$). Pour les fonctions booléennes le **et** est noté $\cdot$ et correspond à une multiplication. L'autre connecteur utilisé est le $\oplus$, le **ou exclusif**.

La négation d'une fonction f est obtenue avec le connecteur logique $\oplus$ et la constante 1, c'est-à-dire en faisant $f \oplus 1$.

De manière plus formelle, on définit $\mathbb{F}_2 = \{0, 1\}$ le corps à deux éléments munis des deux opérations $\oplus$ et $\times$:

$\oplus$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

La notion de corps sera étudiée en deuxième année, pour ce cours de logique, vous n'avez besoin de connaître que ces deux tables.

Écriture et représentation

En calcul propositionnel, l'écriture de la formule est importante, deux formules sont égales si elles ont exactement la même écriture. Par exemple, $(p \wedge \neg q) \wedge r$ et $p \wedge (\neg q \wedge r)$ sont deux formules différentes bien qu'elles aient la même table de vérité. En revanche, comme elles ont la même table de vérité, on peut dire qu'elles sont logiquement équivalentes. Sur les fonctions booléennes, on parle de représentation plutôt que d'écriture et si deux fonctions booléennes ont une représentation différente pour une même table de vérité on dira quand même qu'elles sont égales. Les différentes représentations sont utiles pour définir des propriétés sur ces fonctions. Elles servent également pour les coder efficacement en hardware (avec des circuits booléens).

Table de vérité

Il faut impérativement respecter l'ordre des valuations.

x_1	x_2	x_3	f_3
0	0	0	0
1	0	0	0
0	1	0	1
1	1	0	0
0	0	1	0
1	0	1	1
0	1	1	0
1	1	1	1

Nous avons par exemple $f_3(1, 0, 0) = 0$, $f_3(1, 0, 1) = 1$ et $f_3(0, 0, 1) = 0$.

Soit $n \in \mathbb{N}$ et f_n une fonction booléenne à n variables, f_n est donc une application de $\mathbb{F}_2^n$ vers $\mathbb{F}_2$.

Pour ne pas confondre les valuations et les variables nous noterons différemment celles-ci (c'est un choix que tous ne font pas). Nous noterons toujours x_i (parfois y_i et z_i) les variables et nous choisirons d'autres lettres pour les valuations. Par exemple, $a = (a_1, \ldots, a_n)$ sera une

valuation de $x = (x_1, \ldots, x_n)$. $f_n(a) = f(a_1, \ldots, a_n) = 1$ (resp. 0) signifie que la fonction prend la valeur vraie (resp. fausse) lorsque les variables $x_1, \ldots, x_n$ prennent respectivement les valeurs $a_1, \ldots, a_n$.

Notons $\mathcal{B}F_n$ l'ensemble des fonctions booléennes à n variables. Nous noterons $\overline{0}_n$ (resp. $\overline{1}_n$) la fonction booléenne à n variables prenant toujours la valeur 0 (resp. 1).

Codage d'une fonction

Pour une fonction booléenne à n variables, nous avons 2^n valuations a possibles, f_n peut donc être codée par sa valeur pour ces 2^n valuations.

Il existe plusieurs façons de coder une fonction booléenne par un mot $b_0 \ldots b_{2^n-1}$, la plus commune est la suivante

$$f_n(a_1, \ldots, a_n) = b_k, \text{ où } k = \sum_{i=1}^{n} a_i \, 2^{i-1} \in \{0, \ldots, 2^n - 1\}.$$

$a = (a_1, \ldots, a_n)$ est une représentation binaire de k.

Par exemple, avec $n = 3$, nous avons

— l'indice $k = 3$ correspond à la valuation $a = (1, 1, 0)$:

$$1 \cdot 2^{1-1} + 1 \cdot 2^{2-1} + 0 \cdot 2^{3-1} = 1 + 2 + 0 = 3.$$

— l'indice $k = 4$ correspond à la valuation $a = (0, 0, 1)$

$$0 \cdot 2^{1-1} + 0 \cdot 2^{2-1} + 1 \cdot 2^{3-1} = 0 + 0 + 4 = 4$$

— le plus grand indice vaut $k = 7$, il correspond à la valuation $a = (1, 1, 1)$

$$1 \cdot 2^{1-1} + 1 \cdot 2^{2-1} + 1 \cdot 2^{3-1} = 1 + 2 + 4 = 7$$

Prenons $n = 3$ et considérons f_3 la fonction booléenne à trois variables ayant pour table de vérité

x_1	x_2	x_3	f_3
0	0	0	1
1	0	0	0
0	1	0	1
1	1	0	0
0	0	1	0
1	0	1	1
0	1	1	1
1	1	1	0

On numérote les valuations dans cet ordre et on obtient un mot binaire pour coder f_3 en opérant une rotation de 90 degrés vers la gauche.

f_3	1	0	1	0	0	1	1	0
k	0	1	2	3	4	5	6	7
x_3	0	0	0	0	1	1	1	1
x_2	0	0	1	1	0	0	1	1
x_1	0	1	0	1	0	1	0	1

f_3 est codée par le mot binaire $T(f_3) = 10100110$. La longueur du mot binaire est 2^n, ce qui correspond au nombre de valuations pour n variables.

Notons que $T(f_3)$ permet de coder sans ambiguïté la fonction f_3 uniquement si l'on respecte l'ordre des valuations. Avec d'autres ordres pour ces valuations, nous obtiendrions d'autres mots binaires codant cette même fonction.

Nombre de fonctions booléennes
Pour fixer une fonction booléenne, il suffit de fixer le mot binaire $b_0 \dots b_{2^n-1}$, nous avons deux choix pour chacun des b_i, on montre que cela donne 2^{2^n} fonctions booléennes à n variables. Ce nombre devient très rapidement gigantesque. Pour $n = 8$, nous arrivons à un nombre comparable au nombre d'atomes dans l'univers (celui-ci est estimé à 10^{80}).

n	1	2	3	4	5	6	7	8
$\mathcal{B}F_n$	2	16	256	65536	$4.29\ 10^9$	$1.84\ 10^{19}$	$3.40\ 10^{38}$	$1.15\ 10^{77}$

Si l'on recherche les fonctions booléennes qui vérifient une propriété, il n'est donc pas possible de tester toutes les fonctions booléennes dès que n est suffisamment grand (lorsque n est supérieur ou égal à 8). On ne peut également pas stocker en mémoire toutes les fonctions booléennes de $\mathcal{B}F_n$.

5.2 Mintermes et poids de Hamming

Minterme
Nous avons déjà vu les mintermes dans la partie calcul propositionnel lorsque nous devions déterminer la FND canonique d'une formule F. Un minterme était la conjonction de littéraux où chaque variable propositionnelle contenue dans F apparaît soit positivement, soit négativement.

Ici un minterme est une fonction booléenne particulière, même si cela revient au même.

Pour toute valuation $b = (b_1, \dots, b_n)$, on définit la fonction booléenne M_b qui prend la valeur vraie uniquement lorsque $a = b$. C'est-à-dire

$$\begin{aligned} M_b(a) &= 1 \text{ lorsque } a = b \\ &= 0 \text{ lorsque } a \neq b \end{aligned}$$

Une telle fonction booléenne est appelée un **minterme**.

Comme nous avons un minterme pour chaque valuation, nous avons 2^n mintermes différents.

On peut montrer qu'une fonction booléenne s'écrit comme la somme exclusive (somme avec le connecteur $\oplus$) de mintermes. Par exemple, pour la fonction f_3 précédente

$$f_3 = M_{(0,0,0)} \oplus M_{(0,1,0)} \oplus M_{(1,0,1)} \oplus M_{(0,1,1)}.$$

La somme exclusive contient les mintermes M_b tels que $f_3(b) = 1$. Pour f_3, les valuations b pour lesquelles $f_3(b) = 1$ sont $(0,0,0), (0,1,0), (1,0,1)$ et $(0,1,1)$.

Support d'une fonction booléenne
Le support d'une fonction booléenne est l'ensemble des valuations pour lesquelles la fonction prend la valeur 1.

$$\mathit{support}(f_n) = \{a \in \mathbb{F}_2^n \mid f_n(a) = 1\}.$$

Poids de Hamming
Le poids de Hamming d'une fonction booléenne est le nombre de valuations pour lesquelles elle prend la valeur vraie. C'est donc le nombre de 1 du mot la codant ou encore la cardinalité du support de f_n.

Notons $w_H(f_n)$ le poids de Hamming de f_n. Nous avons donc

$$\begin{aligned} w_H(f_n) &= \sum_{a\in\{0,1\}^n} f_n(a) \\ &= |support(f_n)|, \end{aligned}$$

où $|E|$ désigne le cardinal de l'ensemble E.

Fonction équilibrée
Comme cela a été dit précédemment, les propriétés considérées sur les fonctions booléennes sont plus variées que pour le calcul propositionnel.

Si l'on revient aux propriétés considérées en calcul propositionnel, être satisfaisable signifie que le poids de Hamming vaut 1 ou plus, ou encore que le support de la fonction est non vide. La fonction est une tautologie lorsque le poids de Hamming vaut 2^n, inversement c'est une antologie lorsque le poids de Hamming vaut 0.

Mais on peut définir de nombreuses autres propriétés à partir du poids de Hamming.

Par exemple, une fonction booléenne f_n est équilibrée lorsque $w_H(f_n) = 2^{n-1}$, c'est-à-dire qu'elle prend autant la valeur vraie que fausse.

Il existe une interprétation avec des probabilités de cette propriété, si une valuation a est tiré aléatoirement parmi toutes les 2^n valuations possibles alors

$$\Pr(f_n(a) = 1) = \Pr(f_n(a) = 0) = \frac{1}{2}.$$

C'est-à-dire lorsque la valuation est tirée aléatoirement, nous n'avons pas plus de chance de deviner la valeur de la fonction que de choisir cette valeur aléatoirement. Cette propriété est très utile lorsque les fonctions booléennes sont utilisées en cryptographie.

Opérations sur les fonctions booléennes
Soient f_n et g_n deux fonctions booléennes à n variables. On définit les fonctions $f_n \oplus g_n$ et $f_n \cdot g_n$ de la manière suivante

$\oplus$, ou exclusif, addition modulo 2.

$f_n(a)$	$g_n(a)$	$f_n(a) \oplus g_n(a)$
0	0	0
1	0	1
0	1	1
1	1	0

$\cdot$, et, multiplication.

$f_n(a)$	$g_n(a)$	$f_n(a) \cdot g_n(a)$
0	0	0
1	0	0
0	1	0
1	1	1

Exemple avec les deux fonctions booléennes à deux variables f_2 et g_2 suivantes :

x_1	x_2	f_2	g_2	$f_2 \oplus g_2$	$f_2 \cdot g_2$
0	0	0	1	1	0
1	0	1	1	0	1
0	1	1	0	1	0
1	1	0	1	1	0

5.3 Monômes

Nous avons déjà vu les monômes en calcul propositionnel comme étant une conjonction non nulle de littéraux dans lesquels on ne trouve pas à la fois une variable et sa négation. Pour les fonctions booléennes, les notations sont un peu différentes.

Soit $u = (u_1, \ldots, u_i) \in \{0,1\}^n$. On définit le monôme x^u par

$$x^u = \prod_{i \in \{1,\ldots,n\}} x_i^{u_i}.$$

Pour déterminer x^u, on utilise les deux règles suivantes :

$$\begin{cases} x_i^{u_i} & = & x_i \text{ lorsque } u_i = 1, \\ x_i^{u_i} & = & 1 \text{ lorsque } u_i = 0. \end{cases}$$

À chaque valuation u correspond un monôme x^u.

Pour $n = 3$, voici le calcul de trois monômes.

— avec la valuation $u = (u_1, u_2, u_3) = (0,0,0)$, $x_1^0 = x_2^0 = x_3^0 = 1$ et donc $x^u = 1$.

— avec la valuation $u = (u_1, u_2, u_3) = (1,0,0)$, $x_1^1 = x_1$ et $x_2^0 = x_3^0 = 1$ et donc $x^u = x_1$.

— avec la valuation $u = (u_1, u_2, u_3) = (1,0,1)$, $x_1^1 = x_1, x_3^1 = x_3$ et $x_2^0 = 1$ et donc $x^u = x_1 x_3$.

La table ci-dessous donne le monôme x^u pour chaque valuation u lorsque $n = 3$.

x_1	x_2	x_3	x^u
0	0	0	1
1	0	0	x_1
0	1	0	x_2
1	1	0	x_1x_2
0	0	1	x_3
1	0	1	x_1x_3
0	1	1	x_2x_3
1	1	1	$x_1x_2x_3$

5.4 FAN : Forme Algébrique Normale

Toute fonction booléenne peut s'écrire de la forme

$$f_n = \bigoplus_{u \in \{0,1\}^n} a_u x^u,$$

où $a_u \in \{0,1\}$.

Cela revient à représenter une fonction booléenne en fonction des monômes présents sous cette forme. On utilise une table en remplaçant les valuations par les monômes. Nous appelerons cette table, la table des monômes. La FAN, la forme algébrique normale, correspond à cette table des monômes.

Par exemple, considérons la fonction g_3

x_1	x_2	x_3	x^u	g_3
0	0	0	1	0
1	0	0	x_1	1
0	1	0	x_2	1
1	1	0	x_1x_2	0
0	0	1	x_3	0
1	0	1	x_1x_3	0
0	1	1	x_2x_3	0
1	1	1	$x_1x_2x_3$	1

Les seuls monômes présents sont x_1, x_2 et $x_1x_2x_3$, nous avons donc

$$g_3 = x_1x_2x_3 \oplus x_1 \oplus x_2.$$

Ce n'est pas évident de comparer g_3 avec f_3 donnée précédemment car f_3 est représentée par sa table de vérité et g_3 par sa table des monômes. Pour pouvoir comparer les deux fonctions il faut utiliser la même représentation et nous devons donc être capable de passer d'une représentation à une autre.

Calcul de la FAN d'un minterme
Nous avons vu qu'un minterme M_b était une fonction booléenne qui prenait la valeur vraie une seule fois. Il est représenté par une table de vérité en mettant un 1 uniquement pour la valuation $a = b$.

Le minterme M_b peut aussi être défini comme un produit de littéraux

$$M_b = \prod_{i=1}^{n} (x_i \oplus b_i \oplus 1).$$

On obtient ainsi sa FAN.
Exemples Prenons $b = (0, 0, 0)$,

$$\begin{aligned} M_{(0,0,0)} &= (x_1 \oplus 0 \oplus 1)(x_2 \oplus 0 \oplus 1)(x_3 \oplus 0 \oplus 1) \\ &= (x_1 \oplus 1)(x_2 \oplus 1)(x_3 \oplus 1) \\ &= x_1x_2x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1 \oplus x_2 \oplus x_3 \oplus 1 \end{aligned}$$

Prenons maintenant $b = (1, 0, 1)$,

$$\begin{aligned} M_{(1,0,1)} &= x_1(x_2 \oplus 1)x_3 \\ &= x_1x_2x_3 \oplus x_1x_3 \end{aligned}$$

Calcul de la FAN d'une fonction booléenne avec les mintermes
Nous avons un algorithme très simple pour calculer la FAN d'une fonction booléenne. On commence par représenter celle-ci comme une somme de mintermes. On développe ces mintermes et on effectue des simplifications en utilisant les règles $1 \oplus 1 = 0$ et $x_i \oplus x_i = 0$. La fonction booléenne est donc la somme des monômes apparaissant un nombre impair de fois dans les mintermes.

Il existe des algorithmes plus efficaces basés sur la décomposition d'une fonction booléenne à n variables en deux fonctions booléennes à $n-1$ variables (voir la décomposition de Shannon).

Exemple avec la fonction f_3

$$f_3 = M_{(0,0,0)} \oplus M_{(0,1,0)} \oplus M_{(1,0,1)} \oplus M_{(0,1,1)}.$$

$$\begin{aligned}
M_{(0,0,0)} &= (x_1 \oplus 1)(x_2 \oplus 1)(x_3 \oplus 1) \\
&= x_1x_2x_3 \oplus x_1x_2 \oplus x_1x_3 \oplus x_2x_3 \oplus x_1 \oplus x_2 \oplus x_3 \oplus 1. \\
M_{(0,1,0)} &= (x_1 \oplus 1)x_2(x_3 \oplus 1) \\
&= x_1x_2x_3 \oplus x_1x_2 \oplus x_2x_3 \oplus x_2 \\
M_{(1,0,1)} &= x_1(x_2 \oplus 1)x_3 \\
&= x_1x_2x_3 \oplus x_1x_3 \\
M_{(0,1,1)} &= (x_1 \oplus 1)x_2x_3 \\
&= x_1x_2x_3 \oplus x_2x_3
\end{aligned}$$

Après simplifications, nous obtenons la FAN

$$f_3 = x_2x_3 \oplus x_1 \oplus x_3 \oplus 1.$$

Une fonction booléenne à n variables est donc un polynôme à coefficients dans $\mathbb{F}_2$ sur les variables $x_1, \ldots, x_n$.

Degré algébrique

Le degré algébrique d'une fonction f_n est obtenu à partir de la FAN de la fonction. Il s'agit de la taille de ses plus grands monômes.

Par exemple, f_3 est de degré 2 car le monôme x_2x_3 est le plus grand monôme de sa table des monômes, alors que g_3 est de degré 3 (le plus grand degré possible pour une fonction à trois variables) car le monôme $x_1x_2x_3$ apparaît dans sa table des monômes.

Une fonction de degré 1 est appelée fonction affine.

Le degré algébrique de f_n est noté $deg(f_n)$, on dit également degré de f_n s'il n'y a pas d'ambiguïté.

5.5 Décompositions d'une fonction booléenne

Elles permettent de décomposer une fonction booléenne à n variables en deux fonctions booléennes à $n-1$ variables. Elles sont à la base de la plupart des études sur les fonctions booléennes. La décomposition s'effectue à partir de la table de vérité ou de la table des monômes. Dans le premier cas, l'ordre des valuations est important et dans le second cas, l'ordre des monômes est important.

Décomposition de Shannon

La décomposition de Shannon s'effectue à partir de la table de vérité.

Soit f_n une fonction booléenne à n variables. f_n peut être décomposée de manière unique en deux fonctions à $n-1$ variables f^0_{n-1} et f^1_{n-1} vérifiant

$$\begin{aligned}
f^0_{n-1}(x_1, \ldots, x_{n-1}) &= f_n(x_1, \ldots, x_{n-1}, 0). \\
f^1_{n-1}(x_1, \ldots, x_{n-1}) &= f_n(x_1, \ldots, x_{n-1}, 1).
\end{aligned}$$

Il vient alors

$$f_n = (1 \oplus x_n)f^0_{n-1} \oplus x_n f^1_{n-1}.$$

Avec l'ordre choisi sur les variables, il suffit de diviser en deux la table de vérité.

	f_2^1				f_2^2			
f_3	1	0	1	0	0	1	1	0
x_3	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**
x_2	0	0	1	1	0	0	1	1
x_1	0	1	0	1	0	1	0	1

En reprenant le codage précédent, nous avons $T(f_3) = 10100110$, $T(f_2^0) = 1010$ et $T(f_2^1) = 0110$.

$$T(f_3) = T(f_2^0)||T(f_2^1),$$

où $||$ est la concaténation de mots.

Expansion de Shannon

L'opération inverse est appelée expansion de Shannon. Elle consiste à construire une fonction à n variables f_n à partir de deux fonctions à $n-1$ variables f_{n-1}^0 et f_{n-1}^1. Le choix entre effectuer une décomposition ou une expansion dépend de l'étude que nous effectuons.

Décompositions récursives

Nous pouvons continuer la décomposition jusqu'à arriver à des fonctions booléennes à 1 variable. Pour la fonction f_3, f_2^1 et f_2^2 sont elles-mêmes décomposées en deux fonctions à 1 variable de la manière suivante :

	f_1^1		f_1^2		f_1^3		f_1^4	
f_3	1	0	1	0	0	1	1	0
x_3	0	0	0	0	1	1	1	1
x_2	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**
x_1	0	1	0	1	0	1	0	1

Calcul de la FAN avec la décomposition de Shannon

Nous allons calculer la FAN de f_3 avec les décompositions de Shannon précédentes.

On montre qu'il n'existe que quatre fonctions avec la seule variable x_1 :

x_1	$\overline{0}_1$	$\overline{1}_1$	$x_1 \oplus 1$
0	0	1	1
1	0	1	0

Nous avons la fonction x_1, la fonction $\overline{0}_1$ qui vaut toujours faux, la fonction $\overline{1}_1$ qui vaut toujours vraie et finalement la fonction $x_1 \oplus 1$ qui est la négation de x_1.

À partir de ces quatre fonctions, on retrouve les FAN de f_1^1, f_1^2, f_1^3 et f_1^4.

— $T(f_1^1) = 10$ et $f_1^1 = x_1 \oplus 1$
— $T(f_1^2) = 10$ et $f_1^1 = x_1 \oplus 1$
— $T(f_1^1) = 01$ et $f_1^1 = x_1$
— $T(f_1^2) = 10$ et $f_1^1 = x_1 \oplus 1$

On utilise alors la formule générale de la décomposition de Shannon

$$f_2 = (1 \oplus x_2)f_1^0 \oplus x_2 f_1^1.$$

On applique cette formule à f_2^1 et f_2^2. Ce qui nous donne

$$\begin{aligned} f_2^1 &= (1 \oplus x_2) f_1^1 \oplus x_2 f_1^2 \\ &= (1 \oplus x_2)(1 \oplus x_1) \oplus x_2(1 \oplus x_1) \\ &= (1 \oplus x_1 \oplus x_2 \oplus x_1 x_2) \oplus (x_2 \oplus x_1 x_2) \\ &= 1 \oplus x_1 \end{aligned}$$

$$\begin{aligned} f_2^2 &= (1 \oplus x_2) f_1^3 \oplus x_2 f_1^4 \\ &= (1 \oplus x_2) x_1 \oplus x_2(1 \oplus x_1) \\ &= (x_1 \oplus x_1 x_2) \oplus (x_2 \oplus x_1 x_2) \\ &= x_1 \oplus x_2 \end{aligned}$$

Pour finir, on applique la décomposition de Shannon pour f_3 :

$$\begin{aligned} f_3 &= (1 \oplus x_3) f_2^1 \oplus x_3 f_2^2 \\ &= (1 \oplus x_3)(1 \oplus x_1) \oplus x_3(x_1 \oplus x_2) \\ &= (1 \oplus x_1 \oplus x_3 \oplus x_1 x_3) \oplus (x_1 x_3 \oplus x_2 x_3) \\ &= x_2 x_3 \oplus x_1 \oplus x_3 \oplus 1 \end{aligned}$$

On retrouve bien le même résultat qu'avec la somme des mintermes.

On montre que cette méthode nécessite beaucoup moins de calculs qu'avec la somme des mintermes lorsque qu'elle s'applique à des fonctions booléennes avec un nombre suffisant de variables.

Décomposition de Reed-Müller

Le décomposition de Reed-Müller s'effectue à partir de la table des monômes.

$$g_n = g_{n-1}^0 \oplus x_n \, g_{n-1}^1.$$

g_{n-1}^0 est formée à partir des monômes de g_n qui ne contiennent pas la variable x_n et g_{n-1}^1 avec ceux qui contiennent la variable x_n, on remplaçant toutes les occurrences de x_n par 1 (par exemple, $x_1 x_2 x_n \oplus x_{n-1} x_n$ devient $x_1 x_2 \oplus x_{n-1}$).

Les calculs s'effectuent de la même manière que pour la décomposition de Shannon, mais à la place de travailler à partir de la table de vérité, on travaille avec la table des monômes.
Exemple Reprenons la fonction $g_3 = x_1 x_2 x_3 \oplus x_1 \oplus x_2$ définie précédemment. On cherche la FAN de g_2^1 et g_2^2, les deux fonctions booléennes à deux variables définies par

$$g_3 = g_2^0 \oplus x_3 g_2^1.$$

On utilise pour cela la table des monômes de g_3.

x_1	x_2	x_3	x^u	g_3	g_2^1	g_2^2
0	0	0	1	0		
1	0	0	x_1	1	x_1	
0	1	0	x_2	1	x_2	
1	1	0	x_1x_2	0		
0	0	1	x_3	0		
1	0	1	x_1x_3	0		
0	1	1	x_2x_3	0		
1	1	1	$x_1x_2x_3$	1		x_1x_2

Nous avons donc $g_2^1 = x_1 \oplus x_2$ et $g_2^2 = x_1 x_2$.