UNIVERSITÉ
CAEN
NORMANDIE

Licence d'informatique – première année
Année 2018-2019
Cours de logique et raisonnement

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 se compose de plusieurs parties : une syntaxe (une façon de construire l'ensemble des formules), une sémantique (une description de ce que ces formules signifient), et un système 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 $\mathcal{F}_0$ l'ensemble des formules du calcul propositionnel sur V de variables propositionnelles récursivement (ou inductivement) avec le schéma d'induction suivante :

i) Toute variable propositionnelle est une formule du calcul propositionnel ;

ii) Si P et Q sont des formules, alors (P$\wedge$Q), (P$\vee$Q), (P$\rightarrow$Q) et ($\neg$P) sont des formules du calcul propositionnel.

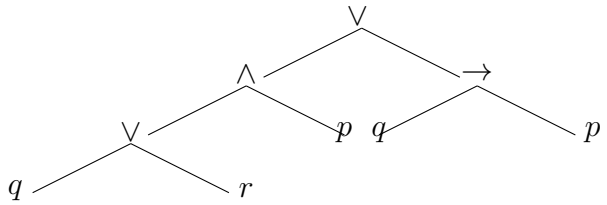
$\wedge$, $\vee$, $\rightarrow$ et $\neg$ sont appelés des *connecteurs logiques*. Par la suite, le terme *formule* désignera une formule du calcul propositionnelle et on utilisera les lettres minuscules pour désigner des variables propositionnelles et les lettres majuscules $A, B, C, \cdots$ pour écrire les formules.

Remarque 1.1 *Le schéma d'induction ci-dessus n'est pas unique, nous pouvons le faire varier, par exemple, en considérant 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.

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



1.3 Sémantique

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

- Si la formule A est une variable propositionnelle p, alors $v(A) = v(p)$.
- Sinon elle 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 :

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

$\top$	1
$\bot$	0

— **et** (notée $\wedge$ ou parfois . 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

— **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

— et l'opération unaire **non**, notée $\neg$ et appelée négation, pour laquelle on utilisera aussi la notation $\overline{\text{A}}$ (négation de A).

— L'implication : 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 :

— Double implication : 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

— **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 est une **tautologie** (ou encore une formule valide) si elle est vraie quelque soit sa valuation (elle prend toujours la valeur 1).

Une formule est **satisfaisable** si au moins l'une de ses valeurs est égale à 1. Dans le cas contraire, la formule est dite **insatisfaisable** ou **contradictoire**, on dit encore que c'est une **antilogie**. Pour toute formule A, on a évidemment 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

— $A \rightarrow B \equiv \neg A \vee B$ et $A \rightarrow B \equiv \neg B \rightarrow \neg A$ (contraposition).

— $A \leftrightarrow B \equiv A \rightarrow B \wedge\ B \rightarrow A$.

— $A \oplus B \equiv (A \wedge \neg B) \vee (\neg A \wedge B)$.

— A est une tautologie si et seulement si $A \equiv \top$.

— A est une antilogie si et seulement si $A \equiv \bot$.

Quelques propriétés des connecteurs
Pour toutes formules A et B, nous avons :
- les trois connecteurs $\wedge$, $\vee$ et $\oplus$ sont commutatifs : $A \wedge B \equiv B \wedge A$.
- les trois connecteurs $\wedge$, $\vee$ et $\oplus$ sont associatifs : $(A \wedge B) \wedge C = A \wedge (B \wedge C)$.
- $\top$ est l'élément neutre de $\wedge$: $\top \wedge A \equiv A \wedge \top \equiv A$.
- $\bot$ est l"élément neutre de $\vee$ et $\oplus$: $\bot \vee A \equiv A \vee \bot \equiv A$.
- $\wedge$ et $\vee$ sont idempotentes : $A \wedge A \equiv A$ et $A \vee A \equiv A$.
- $\bot$ est l'élément absorbant de $\wedge$: $\bot \wedge A \equiv \bot$
- $\top$ 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 et $3(a)$ (resp. 1 et $3(b)$) permettent de montrer que $(\neg, \vee)$ (resp. $(\neg, \wedge)$) est complet.

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 la formule p $\rightarrow$ 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.

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 (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 $\rightarrow$.
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 trois points :

- 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.*

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, ie. 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. A 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 (une conjonction 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.

Définition 1.9 *Un minterme est un monôme de degré n.*

Dans un minterme, chaque variable propositionnelle apparaît une et une seule fois, soit positivement, soit négativement.

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

On construit un maxterme en prenant la négation d'un minterme.

Chaque ligne de la table de vérité correspond à un de ces mintermes, nous avons donc 2^n mintermes et 2^n maxtermes pour une formule contenant n variables propositionnelles.

Exemple

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	$p \wedge q \wedge r$	$\neg p \vee \neg q \vee \neg r$

Considérons la formule A suivante :

ligne	p	q	r	A	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 pour lesquels A prend la valeur 1. On obtient bien une formule logiquement équivalente car A prend la valeur 1 pour une valuation v si et seulement si v correspond à un des monômes M de la disjonction (pour toute variable propositionnelle, $v(p) = 1$ si et seulement si p apparait positivement dans M).

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

La FNC canonique B s'obtient en considérant les lignes où A prend la valeur 0. On exprime que la formule prend la valeur 1 lorsque les valuations ne correspondent pas à ces lignes. B est la conjonction des maxtermes correspondant.

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

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 forme normale conjonctive de $\neg A$.*

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 $C_i = l_1 \vee \ldots l_a$ et $C_j = l_1 \vee \ldots l_a \vee \ldots l_b$, 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 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 la formule étudiée. Pour simplifier la présentation de la méthode des tableaux, on suppose ici qu'elle ne contient que les connecteurs $\vee$, $\wedge$ et $\neg$ (si ce n'est pas le cas, on peut supprimer les autres connecteurs avec des règles de réécriture). On peut de plus, en utilisant les règles de De Morgan, supposer que toute sous-formule est de la forme $A \vee B$ ou $A \wedge B$, autrement dit les négations sont reportées au niveau des variables propositionnelles.

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

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

$$\begin{array}{ccc} E, A \vee B & , & E, A \wedge B \\ \diagup \quad \diagdown & & | \\ E, A \quad E, B & & E, A, B \end{array}$$

où E est un ensemble de formules.

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

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

Exemple

$$\begin{array}{c} (\neg p \vee q) \wedge (p \wedge \neg q) \\ | \\ (\neg p \vee q), (p \wedge \neg q) \\ | \\ (\neg p \vee q), p, \neg q \\ \diagup \quad \diagdown \\ \neg p, p, \neg q \quad q, p, \neg q \end{array}$$

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

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

Dans l'exemple précédent, les deux feuilles de F sont fermées, donc en appliquant le théorème on en déduit que F n'est pas satisfaisable (elle est insatisfaisable ou contradictoire).

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 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 $\mathbb{Z}$ et $\mathbb{R}$.

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

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

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* $\prod_{k=m}^{n} a_k$*,* $\prod_{k=m}^{n} a_k$ *ou* $\prod_{m \leq k \leq n} a_k$ *le produit* $a_m.a_{m+1}. \ldots .a_{n-1}.a_n$*.*

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

Propriété 3.3

$$\prod_{k=1}^{n} \lambda a_k = (\lambda)^n \prod_{k=1}^{n} a_k.$$

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

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.

1. 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

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.

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

$\times$	0	1
0	0	0
1	0	1

Soit $n \in \mathbb{N}$ et f_n une fonction booléenne à n variables, f est donc une application de $\mathbb{F}_2^n$ vers $\mathbb{F}_2$. Les variables sont toujours notées $x_1, \ldots, x_n$. Pour ne pas confondre les valuations et les variables nous noterons différemment celles-ci. 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
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

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

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

Par exemple, avec $n = 3$ et $k = 3$, $a = (1,1,0)$, en effet

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

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

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

On a ainsi $f_3(0,1,0) = 1$ et $f_3(1,1,1) = 0$. f_3 est codée par le mot $T(f_3) = 10100110$.

Nombre de fonctions booléennes
Pour fixer une fonction booléenne, il suffit de fixer le mot $b_0 \dots b_{2^n-1}$, nous avons donc 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 (10^{80}) !

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

5.2 Mintermes et monômes

Minterme
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

$$M_b(a) = 1 \iff a = b.$$

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

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

$$support(f_n) = \{a \in F_2^n \mid f_n(a) = 1\}.$$

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

$$\begin{aligned} w_H(f_n) &= \textstyle\sum_{a \in \mathbb{F}_2} f_n(a) \\ &= |support(f_n)|, \end{aligned}$$

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

Fonction équilibrée
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.

Autrement dit, si a est tiré aléatoirement parmi toutes les 2^n valuations possibles alors

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

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.

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 $f_n \times g_n$ 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

$\times$, et, multiplication.

f	g	$f \times g$
0	0	0
1	0	0
0	1	0
1	1	1

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

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

Nous avons

$$\begin{cases} a_i^{u_i} & = & a_i \text{ lorsque } u_i = 1 \\ a_i^{u_i} & = & 1 \text{ lorsque } u_i = 0 \end{cases}$$

Le minterme M_b peut être défini comme un produit de littéraux

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

Lorsque l'on développe ce produit, comme les coefficients sont sur $\mathbb{F}_2$, deux termes identiques s'annulent. C'est-à-dire, $1 \oplus 1 = 0$ et $x_i \oplus x_i = 0$. M_b peut s'écrire comme la somme des monômes apparaissant un nombre impair de fois.

5.3 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 $a_u \in \{0,1\}$.

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 $x_i \oplus x_i = 0$. 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

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

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

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 la taille de ses plus grands monômes. Une fonction de degré 1 est appelée fonction affine.

5.4 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.

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

On peut continuer la décomposition jusqu'à arriver à des fonctions booléennes à 1 variable. 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.

Décomposition de Reed-Müller
Le décomposition de Reed-Müller s'effectue à partir de la Forme Algébrique Normale de la formule.

$$f_n \quad = \quad g^0_{n-1} \oplus x_n g^1_{n-1}.$$

g^0_{n-1} est formée à partir des monômes de f_n qui contiennent la variable x_n et g^1_{n-1} avec ceux qui ne contiennent pas la variable x_n.

Nous avons $f^0_{n-1} = g^0_{n-1}$ et $f^0_{n-1} \oplus f^1_{n-1} = g^1_{n-1}$.