Calcul propositionnel

Image : Internet network vector created by rawpixel.com - www.freepik.com

I Syntaxe

I.1 Variables propositionnelles

On considère un ensemble infini dénombrable $\mathcal{V}$ dont les éléments sont appelés des *variables propositionnelles*. On note usuellement ces éléments par des lettres majuscules $A, B, X, Y \ldots$.

Par rapport à ce qui a été fait en mathématiques, ces variables représentent une sorte d'abstraction des formules atomiques des mathématiques. On a pu voir des formules comme « $x = y$ » ou « n est pair », ici on les représentera par des variables.

On va alors s'intéresser à la logique en elle-même indépendamment des mathématiques.

I.2 Formules en tant qu'arbre

Grâce aux arbres, on peut donner une définition très précise des formules du calcul propositionnel :

Définition I.1 Une formule du calcul propositionnelle est un arbre ayant des nœuds binaires, étiquetés par $\wedge, \vee, \to$ ou $\leftrightarrow$, des nœuds unaires étiquetés par $\neg$ et dont les feuilles sont étiquettés par $\mathcal{V}$.

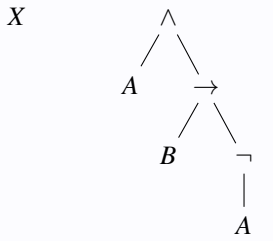
Remarque Les symboles $\wedge, \vee, \to$ et $\leftrightarrow$ et $\neg$ sont appelés des connecteurs logiques. Les quatres premiers sont dits binaires, ou d'arité 2, le dernier est dit unaire, ou d'arité 1.

Pour le moment, il ne s'agit que de symboles mais ils ne sont pas choisi au hasard et on leur attribuera un sens dans la suite de manière cohérente avec l'usage mathématique.

- $\wedge$ est appelé la conjonction et se lit *et*
- $\vee$ est appelé la disjonction et se lit *ou*

- $\rightarrow$ est appelé l'implication et se lit *implique*
- $\leftrightarrow$ est appelé l'équivalence et se lit *équivaut*
- $\neg$ est appelé la négation et se lit *non*

Exemple



où $A, B, X \in \mathcal{V}$.

De cette définition, on déduit directement la notion de taille et de hauteur d'une formule, il s'agit de la taille ou de la hauteur en tant qu'arbre.

On note $\mathcal{V} \dashv \nabla(f)$ l'ensemble des variables apparaissant dans f, c'est-à-dire l'ensemble des étiquettes des feuilles de l'arbre f.

I.3 Définition inductive des formules

Il est plus traditionnel, bine qu'équivalent, de définir les formules sous la forme d'un ensemble inductif :

Définition I.2 Les formules du calcul propositionnel sont les éléments du plus petit ensemble $\mathcal{F}$ tel que :

- $\mathcal{V} \subset \mathcal{F}$
- $\forall f, f' \in \mathcal{F}, \forall op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}, (f op f') \in \mathcal{F}$.
- $\forall f \in \mathcal{F}, \neg f \in \mathcal{F}$.

On peut résumer cela sous la forme de l'équation inductive suivante :

$$\mathcal{F} = \mathcal{V} | (\mathcal{F} \wedge \mathcal{F}) | (\mathcal{F} \vee \mathcal{F}) | (\mathcal{F} \rightarrow \mathcal{F}) | (\mathcal{F} \leftrightarrow \mathcal{F}) | \neg \mathcal{F}$$

Quand on définit ainsi l'ensemble, on peut se demander la nature réelle des objets que l'on a défini. Ici, il y a deux approches classiques :

- Les formules sont des mots sur l'alphabet $\mathcal{V} \cup \{(,), \vee, \wedge, \rightarrow, \leftrightarrow\}$. Par exemple $(X \vee Y) \wedge \neg Z \in \mathcal{F}$ mais $(X \neg Y) \notin \mathcal{F}$. C'est la vision *historique* des formules comme des chaînes de caractères.
- Les formules sont des arbres comme on l'a vu précédemment. L'avantage principal étant que tous ces arbres sont des formules sans ambigüité de lecture possible.

Que ce soit sous forme de chaîne ou d'arbre, on adopte une représentation plate des formules. Comme cela peut induire beaucoup de parenthèses, on va adopter une convention de priorité : en cas d'ambigüité, c'est la construction la plus prioritaire qui s'effectue.

On a les règles de priorité usuelles : $\neg > \wedge, \vee > \rightarrow, \leftrightarrow$. On justifiera de plus dans la suite qu'on peut considérer $\wedge$ et $\vee$ comme étant associatives, et $\leftrightarrow$ comme étant associative à droite.

Exemple

- $(A \wedge (B \vee C))$ s'écrit $A \wedge (B \vee C)$
- $(A \vee (B \vee C))$ s'écrit $A \vee B \vee C$
- $(A \rightarrow (B \rightarrow C))$ s'écrit $A \rightarrow B \rightarrow C$
- $((A \rightarrow B) \rightarrow C)$ s'écrit $(A \rightarrow B) \rightarrow C$

II Sémantique du calcul propositionnel

On considère ici un ensemble à deux éléments $\mathbb{B} = \{V, F\}$ dont les éléments sont appelés des valeurs de vérité. On parle de la valeur *vraie* pour V et *fausse* pour F. C'est une pure convention, il n'y a aucune véracité mathématique pour appuyer ces termes. Quelque part, c'est la même situation que pour les bases directes.

II.1 Valuation et valeur de vérité d'une formule

Définition II.1 Une valuation est une fonction $v : \mathcal{V} \rightarrow \mathbb{B}$.

C'est une forme d'environnement pour les formules comme pourrait l'être une affectation de valeurs à des

variables pour évaluer des expressions arithmétiques. On va donc pouvoir s'appuyer sur une telle valuation pour définir la valeur de vérité d'une formule comme une évaluation :

Définition II.2 Soit v une valuation et $f \in \mathcal{F}$. On appelle **valeur de vérité** de f sous la valuation f, l'élément $[f]_v \in \mathbb{B}$ défini inductivement par :

- $\forall X \in \mathcal{V}, [X]_v = v(X)$
- $\forall f \in \mathcal{F}, [\neg f]_v = \begin{cases} V & \text{si } [f]_v = F \\ F & \text{sinon} \end{cases}$
- $\forall f, f' \in \mathcal{F}, \forall \square \in \{\wedge, \vee, \to, \leftrightarrow\}$ on détermine $[f \square f']_v$ en fonction de $[f]_v$ et $[f']_v$ selon la table suivante :

$[f]_v$	$[f']_v$	$[f \vee f']_v$	$[f \wedge f']_v$	$[f \to f']_v$	$[f \leftrightarrow f']_v$
V	V	V	V	V	V
V	F	V	F	F	F
F	V	V	F	V	F
F	F	F	F	V	V

Exemple Si $f = A \vee B \to A \wedge C$ et qu'on sait que $v(A) = v(C) = V$ et $v(B) = F$, alors $[A \vee B]_v = V$ et $[A \wedge C] = V$ selon la table. On a alors $[f]_v = V$.

II.2 Table de vérité

Une formule ne pouvant faire apparaitre qu'un nombre fini de variable, les valuations comportent beaucoup d'information n'ayant aucune influence sur la valeur de vérité de f. Cela est précisé dans le théorème suivant :

Théorème II.1 Soit $f \in \mathcal{F}$ et $v, v' : \mathcal{V} \to \mathbb{B}$ deux valuations.
Si $\forall X \in Var(f), v_X = v'_X$ alors $[f]_v = [f]_{v'}$.

Preuve

Par induction structurelle sur f.

- Si $X \in \mathcal{V}$ et $f = X$, alors $[f]_v = v(X) = v'(X) = [f]_{v'}$.
- Si la propriété est vraie pour $f \in \mathcal{F}$ alors $[\neg f]_v = V \iff [f]_v = F = [f]_{v'} \iff [\neg f']_v = V$. Ainsi, comme il n'y a que deux valeurs possibles, $[\neg f]_v = [\neg f']_v$.
- Si la propriété est vraie pour $f, f' \in \mathcal{F}$, alors $[f]_v = [f]_{v'}$ et $[f']_v = [f']_{v'}$. Or, $[f \square f']_v$ ne dépend que de $[f]_v$ et $[f']_v$ selon les tables de la définition pour $\square \in \{\vee, \wedge, \to, \leftrightarrow\}$. On en conclut directement que $[f \square f']_v = [f \square f']_{v'}$.

■

Ainsi, on peut se concentrer sur les valeurs de vérité associées aux variables présentes dans une formule. Se faisant, on introduit le concept de tables de vérité :

Définition II.3 Soit $f \in \mathcal{F}$ à n variables $X_1, \ldots, X_n$.
On appelle **table de vérité** de f une fonction

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

telle que pour toute valuation v on ait

$$[f]_v = T_f(v(X_1), \ldots, v(X_n))$$

Théorème II.2 Toute formule a une table de vérité et elle est unique à l'ordre près de l'enumération de ses variables.

Preuve

Soit $f \in \mathcal{F}$ et $X_1, \ldots, X_n$ ses variables.

1. Si $b = (b_1, \ldots, b_n) \in \mathbb{B}^{\mathbb{N}}$, on note

$$\begin{array}{rcccl} v_b & : & \mathcal{V} & \mapsto & \mathbb{B} \\ & & Y & \mapsto & \begin{cases} b_i & \text{si } Y = X_i \text{ pour } i \in [\![1, n]\!] \\ F & \text{sinon} \end{cases} \end{array}$$

On pose alors $T_f : b \mapsto [f]_{v_b}$.

2. Soit v' une valuation et $b = (v'(X_1), \dots, v'(X_n))$. On a v_b et v' identiques sur les variables de f, donc $[f]_{v'} = [f]_{v_b} = T_f(b)$.
3. Si T' est une autre table de verité pour le même ordre d'énumération, on a pour $b \in \mathbb{B}^{\mathbb{N}}$, par définition de la table $T'(b) = [f]_{v_b} = T_f(b)$. Donc $T' = T_f$.

■

Remarque On représente usuellement une table de vérité sous la forme d'une table avec une colonne par variable et une colonne pour la valeur de f.

Par exemple, pour $f = (A \wedge B) \vee C$:

A	B	C	f
V	V	V	V
V	V	F	V
V	F	V	V
V	F	F	F
F	V	V	V
F	V	F	F
F	F	V	V
F	F	F	F

II.3 Satisfiabilité, tautologie

Définition II.4 Soit $f \in \mathcal{F}$ et $v : \mathcal{V} \to \mathbb{B}$.

On note $v \models f$ quand $[f]_v = V$ et on dit que v est un modèle de f. On note ainsi $Mod(f) = \{ v \mid v \models f \}$.

- Si $Mod(f) \neq \emptyset$ on dit que f est **satisfiable**.
- Si $Mod(f) = \mathbb{B}^{\mathcal{V}}$ on dit que f est **une tautologie**.
- Si $Mod(f) = \emptyset$ on dit que f est **une antilogie**.

Théorème II.3

- f est une tautologie ssi T_f est constante de valeur V
- f est une antilogie ssi T_f est constante de valeur F
- f est satisfiable ssi $V \in T_f(\mathbb{B}^N)$.

Exemple

- $X \vee \neg X$ est une tautologie appelée le tiers-exclu.
- $X \wedge \neg X$ est une antilogie appelée la non-contradiction.
- $((A \to B) \to A) \to A$ est une tautologie.

Remarque Il est pratique d'introduire des formules constantes $\top$ et $\bot$ pour représenter une tautologie et une antilogie quelconque.

II.4 Équivalence entre deux formules

Définition II.5 Soient $f, f' \in \mathcal{F}$. On dit que f et f' sont équivalentes, et l'on note $f \iff f'$, lorsque f et f' ont mêmes modèles, c'est-à-dire lorsque $Mod(f) = Mod(f')$.

Théorème II.4 L'équivalence logique est une relation d'équivalence.

Remarque

- Cela signifie que deux formules équivalentes ne peuvent être distingués par leur valeur de vérité sous une valuation.
- f est une tautologie si et seulement si $f \iff \top$
- f est une antilogie si et seulement si $f \iff \bot$

Théorème II.5 Soient $f, f' \in \mathcal{F}$ avec $Var(f) = Var(f')$.

On a $f \iff f'$ si et seulement si $T_f = T_{f'}$.

Remarque Cela permet de vérifier l'équivalence entre deux formules en établissant les deux tables de vérité et en vérifiant quelles sont identiques. Comme une telle table comporte 2^n lignes pour n variables, c'est souvent fastidieux.

Il y a de nombreuses équivalences logiques qui sont fondamentales car elles permettent de simplifier les formules pour travailler sur des formules plus simples. On va démontrer ici les principales :

Théorème II.6 Soient $P, Q, R \in \mathcal{V}$
- $\neg\neg P \iff P$
- $P \to Q \iff \neg P \vee Q$
- $P \leftrightarrow Q \iff (P \wedge Q) \vee (\neg P \wedge \neg Q)$

Lois de de Morgan :
- $\neg(P \vee Q) \iff \neg P \wedge \neg Q$
- $\neg(P \wedge Q) \iff \neg P \vee \neg Q$

Commutativité : $* \; P \wedge Q \iff Q \wedge P \; * \; P \vee Q \iff Q \vee P$

Associativité :
- $P \wedge (Q \wedge R) \iff (P \wedge Q) \wedge R$
- $P \vee (Q \vee R) \iff (P \vee Q) \vee R$

Distributivité :
- $P \wedge (Q \vee R) \iff (P \vee Q) \wedge (P \vee R)$
- $P \vee (Q \wedge R) \iff (P \wedge Q) \vee (P \wedge R)$

■ **Preuve**

Il suffit de faire de fastidieuses vérifications à l'aide de tables de vérité. On peut se contenter de ne démontrer qu'une partie d'entre elles grâce à la double négation et aux lois de de Morgan.

■

II.5 Substitution et congruence

On vient de démontrer des équivalences mais le fait d'utiliser des variables semblent en restreindre la portée. En effet, si on considère la formule $\neg\neg(A \to B)$ il ne semble pas possible d'en déduire directement qu'elle est équivalente à $A \to B$ car l'équivalence $\neg\neg X \iff X$ ne s'applique que pour une variable.

Or, dès le départ, les variables propositionnelles ne se voulaient qu'être des abstractions sur des propriétés. On va démontrer ici un lemme de substitution qui permettra de déduire de nouvelles équivalences en remplaçant une variable par une formule.

Définition II.6 Soit $f, g \in \mathcal{F}$ et $X \in \mathcal{V}$. On définit la substitution de X par g, notée $f[g/X]$ par induction ainsi :
- Si $Y \in \mathcal{V}\backslash\{X\}, Y[g/X] = Y$
- $X[g/X] = g$
- Si $f \in \mathcal{F}, (\neg f)[g/X] = \neg f[g/X]$
- Si $\square \in \{\wedge, \vee, \to, \leftrightarrow\}$ et $f, f' \in \mathcal{F}, (f \square f')[g/X] = f[g/X] \square f'[g/X]$

Lemme II.7 Soit $f, g \in \mathcal{F}, X \in \mathcal{V}$ et $v : \mathcal{V} \to \mathbb{B}$.
On pose $v' : \mathcal{V} \to \mathbb{B}$ telle que $v'(X) = [g]_v$ et $v'(Y) = v(Y)$ pour $Y \neq X$.
On a alors $[f[g/X]]_v = [f]_{v'}$.

Théorème II.8 Soient $f, g, h \in \mathcal{F}$ et $X \in \mathcal{V}$.
Si $f \equiv g$, alors $f[h/X] \equiv g[h/X]$.

■ **Preuve**

Soit v une valuation, on a défini v' comme dans le lemme précédent avec $v'(X) = [h]_v$. On a alors $[f[h/X]]_v = [f]_{v'}$ et $[g[h/X]]_v = [g]_{v'}$. Or, par hypothèse, $[f]_{v'} = [g]_{v'}$.
On a ainsi l'égalité $[f[h/X]]_v = [g[h/X]]_v$ et donc $f[h/X] \equiv g[h/X]$.

■

Le fait de faire passer des équivalences sous des opérations, ici les connecteurs logiques, est traditionnellement appelé une congruence. On a ici le théorème suivant :

Théorème II.9 Soient $f, g, h \in \mathcal{F}$ où $f \equiv g$.
On a $* \neg f \equiv \neg g * \forall \square \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$, $f \square h \equiv g \square h$ et $h \square f \equiv h \square f$.

II.6 Conséquence logique

Définition II.7 Soit $f, g \in \mathcal{F}$, on dit que f est **conséquence logique** de g lorsque $Mod(g) \subset Mod(f)$.
Autrement dit, lorsque pour toute valuation v, $[g]_v = V$ implique $[f]_v = V$.
On note alors $g \models f$.
Si $\Gamma \subset \mathcal{F}$ on note $\Gamma \models f$ s'il existe $g_1, \ldots, g_n \in \Gamma$ tels que $g_1 \wedge \cdots \wedge g_n \models f$.

Lemme II.10 Soient $\Gamma, \Delta \subset \mathcal{F}$ et $f \in \mathcal{F}$.
Si $\Gamma \models f$ alors $\Gamma \cup \Delta \models f$.

■ **Preuve**
Pour simplifier, on peut supposer que Γ et Δ sont réduits à une formule, quitte à considérer $\top$ s'ils sont vides.
On suppose ainsi que $g \models f$ et on cherche à montrer que $g \wedge h \models f$.
Soit v une valuation telle que $v \in Mod(g \wedge h)$. On a donc $[g \wedge h]_v = V$ et par définition de la valeur de vérité d'une conjonction, $[g]_v = [h]_v = V$. Or, comme $[g]_v$ et que $g \models f$ on a $[f]_v = V$. On vient de montrer $g \wedge h \models f$.
■

Théorème II.11 Les propriétés suivants sont équivalentes :
1. $g \models f$
2. $g, \neg f \models \bot$
3. $g \rightarrow f$ est une tautologie

■ **Preuve**
1. $\Rightarrow$ 2. : Supposons que $g \models f$ et supposons qu'il existe une valuation v telle que $[g \wedge \neg f]_v = V$. On a donc $[g]_v = [\neg f]_v = V$ par l'argument usuel sur la table de la conjonction. Ainsi $[f]_v = F$ mais comme $g \models f$, on a $[f]_v = V$. Or, une formule a une unique valeur de vérité, donc on aboutit à une contradiction. Ainsi $Mod(g \wedge \neg f) = \emptyset$ et $g \wedge \neg f \models \bot$ vu que $Mod(\bot) = \emptyset$.
2. $\Rightarrow$ 3. Suposons que $g, \neg f \models \bot$. On a alors $\mathcal{M}\wr\lceil(g \wedge \neg f) = \emptyset$. Soit v une valuation et supposons, par l'absurde, que $[g \rightarrow f]_v = F$. Alors, de par la table de $\rightarrow$, cela signifie nécessairement que $[g]_v = V$ et $[f]_v = F$. Donc $[g \wedge \neg f]_v = V$. Contradiction car on a trouvé un modèle de $g \wedge \neg f$.
3. $\Rightarrow$ 1. Supposons que $g \rightarrow f$ soit une tautologie et soit v une valuation telle que $[g]_v = V$. Comme $[g \rightarrow f]_v = V$ on a nécessairement $[f]_v = V$ par étude de la table de $\rightarrow$. Donc, on a bien $g \models f$.
■

Remarque Le fait de se ramener à montrer $\Gamma, \neg f \models \bot$ pour prouver que $\Gamma \models f$ est un principe très important appelé la *résolution* qui a été introduit par le logicien John Alan Robinson et sert de base au langage de programmation logique Prolog.

Théorème II.12 Si $f \models g$ et $g \models h$ alors $f \models g$.

■ **Preuve**
Il s'agit juste de la transitivité de l'inclusion sur les modèles.
■

Remarque Cette propriété appelé le *Modus Ponens* est un principe fondamental du raisonnement identifiée très tôt dans l'histoire des mathématiques.

III Formes normales

On a vu beaucoup d'équivalences logiques dans la partie précédente, elles permettent de réécrire une formule sous une forme plus pratique pour raisonner. On parle alors de **forme normale**.
Comme on a pu le voir, les opérateurs $\rightarrow$ et $\leftrightarrow$ peuvent s'exprimer avec les trois autres. On va donc considérer ici uniquement des formules avec les opérateurs $\vee, \wedge, \neg$.

III.1 Forme normale conjonctive et disjonctive

Définition III.1 Une formule de la forme X ou $\neg X$ avec $X \in \mathcal{V}$ est appelé un **littéral**, on précise parfois positif ou négatif.

Une formule de la forme $l_1 \vee \cdots \vee l_n$ où les l_i sont des littéraux est appelée une **clause**. On parle de clause vide pour $\bot$.

Une formule de la forme $C_1 \wedge \cdots \wedge C_m$ où les C_j sont des clauses est appelée une **forme normale conjonctive** (FNC). On parle de FNC vide pour $\top$.

Exemple
- $A \wedge (\neg A \vee B) \wedge \bot$ est une FNC.

En vertu de l'associativité des connecteurs, on note de manière synthétique

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n_i} l_{i,j}$$

une FNC, où les $l_{i,j}$ sont des littéraux.

Définition III.2 On définit de même les formes normales disjonctives (FND) comme les disjonctions de conjonctions de littéraux.

Ce sont les formules sous la forme :

$$\bigvee_{i=1}^{n} \bigwedge_{j=1}^{n_i} l_{i,j}$$

où les $l_{i,j}$ sont des littéraux.

III.2 Mise sous forme normale disjonctive

Soit $f \in \mathcal{F}$ dont les variables sont $X_1, \ldots, X_n$, on peut passer de sa table de vérité à une FND en posant :

$$\Phi(f) = \bigvee_{(b_1,\ldots,b_n) \in T_f^{-1}(\{1\})} \bigwedge_{i=1}^{n} l(X_i, b_i)$$

où $l(X_i, V) = X_i$ et $l(X_i, F) = \neg X_i$.

On a directement $\Phi(f) \equiv f$ car $\Phi(f)$ a été construite pour avoir la même table de vérité que f.

On remarque tout de suite que la taille de $\Phi(f)$ dépend du nombre de valeur à V dans sa table de vérité. Une tautologie portant sur n variables aura donc une FND de taille $O(2^n)$.

III.3 Mise sous forme normale conjonctive

Définition III.3 Soit $f \in \mathcal{F}$, on définit la formule $\varphi(f)$ ainsi par induction structurelle :
- Si $X \in \mathcal{V}$, $\varphi(X) = X$
- Si $f \in \mathcal{F}$,
  - ⋆ Soit $f = X \in \mathcal{X}$ et $\varphi(\neg X) = \neg X$
  - ⋆ Soit $f = \neg f'$ et $\varphi(\neg f) = \varphi(f')$
  - ⋆ Soit $f = g \wedge h$ et $\varphi(\neg f) = \varphi(\neg g) \vee \varphi(\neg h)$.
  - ⋆ Soit $f = g \vee h$ et $\varphi(\neg f) = \varphi(\neg g) \wedge \varphi(\neg h)$.
- Si $f, f' \in \mathcal{F}$, $\varphi(f \vee f') = \varphi(f) \vee \varphi(f')$ et $\varphi(f \wedge f') = \varphi(f) \wedge \varphi(f')$.

Théorème III.1 On a $\varphi(f) \iff f$ et dans $\varphi(f)$ les négations appartiennent toutes à des littéraux.

■ **Preuve**

On démontre ce théorème rapidement par induction structurelle rapide.

Pour l'équivalence, il s'agit d'équivalences déjà démontré (double négation et lois de de Morgan).

■

Il est possible de passer de FND à FNC à l'aide d'une négation et des lois de de Morgan. On en déduit donc une manière d'obtenir une FNC équivalente à f : on met $\neg\Phi(\neg f) \equiv f$ sous FNC en éliminant la négation.

Cependant, on peut obtenir une FNC plus directement par induction comme on peut le voir dans la définition suivante.

Définition III.4 Soit f une formule dont les négations appartiennent toutes à des littéraux, on définit $\psi(f)$ par induction structurelle :

- $\psi(l) = l$ si l est un littéral
- $\psi(f \wedge g) = \psi(f) \wedge \psi(g)$
- Si $\psi(f) = f_1 \wedge f_2$ alors $\psi(f \vee g) = \psi(f_1 \vee g) \wedge \psi(f_2 \vee g)$.
- Si $\psi(f)$ n'est pas une conjonction et $\psi(g) = g_1 \wedge g_2$ alors $\psi(f \vee g) = \psi(f \vee g_1) \wedge \psi(f \vee g_2)$.
- Si ni $\psi(f)$ ni $\psi(g)$ ne sont des conjonctions alors $\psi(f \vee g) = \psi(f) \vee \psi(g)$.

Théorème III.2 Soit $f \in \mathcal{F}$, $\psi(\varphi(f)) \equiv f$ et $\psi(\varphi(f))$ est en forme normale conjonctive.

Remarque Il est possible d'obtenir ainsi une formule de taille exponentielle.

L'exemple caractéristique est

$$\bigvee_{i=1}^{n} (X_{i,1} \wedge X_{i,2}) \equiv \bigwedge_{a_1 \in \{1,2\}, \ldots, a_n \in \{1,2\}} (X_{1,a_1} \wedge \cdots \wedge X_{n,a_n})$$

On passe ainsi d'une formule de taille $O(n)$ à une formule de taille $O(2^n)$.

Il est possible d'implémenter directement cette méthode :

OCaml
```
type formula = Var of string
    | And of formula * formula
    | Or of formula * formula
    | Not of formula

let rec elim_not f = match f with
    | Var s -> Var s
    | Not (Var s) -> Not (Var s)
    | Not (Not f) -> elim_not f
    | Not (Or (f1,f2)) -> And( elim_not(Not f1), elim_not(Not f2) )
    | Not (And (f1,f2)) -> Or( elim_not(Not f1), elim_not(Not f2) )
    | Or(f1, f2) -> Or(elim_not f1, elim_not f2)
    | And(f1, f2) -> And(elim_not f1, elim_not f2)

let rec distrib_or f = match f with
    | Var _ -> f
    | Not (Var _) -> f
    | And(f1,f2) -> And(distrib_or f1, distrib_or f2)
    | Or(f1,f2) -> begin
          match distrib_or f1 with
          | And(f11, f12) -> And(distrib_or (Or(f11,f2)),
                                 distrib_or (Or(f12,f2)))
          | f1' -> match distrib_or f2 with
              | And (f21, f22) -> And(distrib_or(Or(f1', f21)),
                                      distrib_or(Or(f1',f22)))
              | f2' -> Or(f1', f2')
          end
    | _ -> failwith "Formule avec négation en dehors des littéraux"

let fnc f = distrib_or (elim_not f)
```

IV Problème SAT

IV.1 Définition

Un des problèmes les plus importants en logique et plus largement en informatique est le problème SAT :

Problème - SAT
- Entrées :
  Une formule f, le plus souvent en FNC.
- Sortie :
  Un booléen indiquant si f est satisfiable.

IV.2 k-SAT

Le problème suivant permet un plus grand contrôle sur les formules.

Problème - k-SAT
- Entrées :
  Une formule f en FNC dont les clauses comportent au plus k littéraux.
- Sortie :
  Un booléen indiquant si f est satisfiable.

Naturellement, si on sait résoudre k-SAT pour tout k, alors on sait résoudre SAT vu que les formules sont finies.

IV.3 1-SAT

Les formules à résoudre sont des conjonctions de littéraux vu que les clauses sont réduites à un littéral. Or, soit cette conjonction contient deux littéraux opposés auxquel cas $f \equiv \bot$, ou alors elle est satisfiable en considérant $v(X) = V$ si X apparait positivement ou $v(X) = F$ si elle apparait négativement.

IV.4 2-SAT

Pour résoudre 2-SAT on va construire un graphe en transformant les clauses en implications :

$$X \vee Y \iff \neg X \to Y \iff \neg Y \to X$$
$$\neg X \vee Y \iff X \to Y \iff \neg Y \to \neg X$$
$$\neg X \vee \neg Y \iff X \to \neg Y \iff Y \to \neg X$$

On peut alors construire un graphe où les sommets sont les littéraux apparaissant dans une formule ainsi que leur négation, et où l'on relie l et l' quand une clause est équivalente à $l \to l'$.

Un chemin dans ce graphe est une chaine d'implication. On peut montrer que f est satisfiable si et seulement un littéral n'est jamais relié à son opposé. Or, pour déterminer cela, il suffit de calculer la fermeture transitive du graphe, par exemple avec des parcours successifs, ou avec Floyd-Warshall. Cela se fait donc en temps et en espace polynomial dans le nombre de variables.

IV.5 3-SAT

On peut montrer un théorème assurant que ce chapitre aura un nombre fini de paragraphes :

Théorème IV.1 Si on sait résoudre 3-SAT alors on sait résoudre k-SAT pour tout k.

Cependant, comme on en reparlera dans le chapitre sur la NP-complétude, le problème 3-SAT est *a priori* très difficile et on a peu d'espoir de pouvoir le résoudre de manière générale autrement qu'en énumérant les 2^n valuations.

IV.6 Algorithme de Quine

L'algorithme de Quine consiste à construire un arbre de décision associé à une formule f inductivement ainsi :
- si f est sans variable, $f \equiv \top$ ou $\bot$ et on renvoie une feuille avec cette valeur.
- sinon on considère une variable X quelconque dans f et on renvoie l'arbre dont la racine est étiquetée par X, a pour fils gauche l'arbre associé à $f[\bot /X]$ et pour fils droit l'arbre associé à $f[\top/X]$.

Si f est satisfiable, alors cet arbre a une feuille de valeur de $\top$. Il suffit alors de lire les embranchements pour retrouver le modèle correspondant.

Si f est une tautologie, toutes ses feuilles sont étiquettes par $\top$.

On donne ici une implémentation rapide de cet algorithme :

OCaml
```
type formule = Et of formule * formule
  | Ou of formule * formule | Non of formule
  | Vrai | Faux | Var of string

let rec var f =
   match f with
   | Ou(f1, f2)
   | Et(f1, f2) -> var f1 @ var f2
   | Non f' -> var f'
   | Var s -> [s]
   | _ -> []

let rec subst f s v =
   match f with
   | Ou(f1, f2) -> Ou(subst f1 s v, subst f2 s v)
   | Et(f1, f2) -> Et(subst f1 s v, subst f2 s v)
   | Non f' -> Non (subst f' s v)
   | Var t when s = t -> v
   | _ -> f


let rec simpl f =
   match f with
   | Et(f1, f2) -> begin
         match simpl f1, simpl f2 with
         | Vrai, f -> f
         | f, Vrai -> f
         | Faux, _ -> Faux
         | _, Faux -> Faux
         | f1', f2' -> Et(f1', f2')
         end
   | Ou(f1, f2) -> begin
         match simpl f1, simpl f2 with
         | Vrai, _ -> Vrai
         | _, Vrai -> Vrai
         | Faux, f -> f
         | f, Faux -> f
         | f1', f2' -> Ou(f1', f2')
          end
   | Non f -> begin
      match simpl f with
      | Vrai -> Faux
      | Faux -> Vrai
      | f' -> Non f'
      end
   | _ -> f

type quine_resultat = Valide | Invalide | Branchement of string * quine_resultat * quine_resultat

let rec quine f =
   match simpl f with
   | Vrai -> Valide
   | Faux -> Invalide
   | f' -> let s = List.hd (var f') in
      Branchement(s, quine (subst f' s Vrai), quine (subst f' s Faux))

let imp f1 f2 = Ou(Non f1, f2)

let ex = imp (Et(Et(imp (Var "s") (Ou(Var "b", Var "t")),
   imp (Ou(Var "b", Var "a"))  (Et(Var "r", Var "m"))), Non (Var "r")))
   (imp (Var "s") (Var "t"))

let b = quine ex
```