Chapitre 3

Introduction à la logique

Option informatique MP1, MP2 & MP3

La logique est l'outil qui a été créé pour étudier et formaliser le raisonnement mathématique. Elle s'est par la suite imposée comme une partie essentielle de l'informatique. Une caractéristique principale de la logique est qu'elle fait apparaître nettement la différence entre la *syntaxe*, qui définit les règles formelles de manipulation des symboles utilisés, et la *sémantique*, qui étudie la signification ou l'interprétation des symboles utilisés.

I Introduction

La logique étudie les énoncés qui sont susceptibles d'être vrais ou faux.

Définition 1 : Proposition

Une proposition est un énoncé de quelque nature que ce soit qui peut être qualifié de vrai ou faux.

Dans le vocabulaire informatique on dira plutôt expression booléenne.
Ces énoncés peuvent être simples :
$1+3=5$, $x^2-3x+2=0$ admet 2 comme solution, "il fait beau", ...
Mais ils peuvent être composés d'un assemblage : "si le Groenland est recouvert de sucre glace alors 2+2 =5", "le ciel est bleu et les oiseaux chantent", ...
Il existe des énoncés qui ne sont pas des proposition : les phrases impératives ou interrogatives par exemple. On peut aussi penser aux énoncés qui ne peuvent avoir de sens, soit parce qu'ils emploient des mots non définis, "les duramiles sont blouvus", soit parce qu'il est impossible de leur donner une vérité (en général à cause d'une auto-référence), "cet énoncé est faux".

II Syntaxe

II.1 Formules bien formées

Nous allons représenter les énoncés simples par des **variables propositionnelles**. Dans la pratique nous n'utiliserons qu'un nombre fini de telles variables mais nous supposerons que l'ensemble de ces variables peut être dénombrable.
On notera $\mathcal{A}$ cet ensemble ; ses éléments pourront être notés p, q, r, ...ou p_0, p_1, p_2 ...
Si A est un ensemble on note A^* l'ensemble des assemblages (les **mots**) d'éléments de A (les **lettres**). Ils sont représentés en collant les lettres : $w = x_1x_2\cdots x_m$.
Si $\mathcal{A}$ est un ensemble de variables propositionnelles, on note $\mathcal{A}' = \mathcal{A} \cup \{(,), \neg, \wedge, \vee\}$.
Les ensembles considérés sont des sous-ensembles de $\left(\mathcal{A}'\right)^*$.

Définition 2 : Formules

L'ensemble des formules (bien formées) de la logique des propositions sur $\mathcal{A}$ est le plus petit ensemble $\mathcal{L}$ (ou $\mathcal{L}_\mathcal{A}$) tel que

- toute variable propositionnelle (élément de $\mathcal{A}$) appartient à $\mathcal{L}$,
- si $F \in \mathcal{L}$ est une formule, alors $(\neg F) \in \mathcal{L}$,
- si $F \in \mathcal{L}$ et $G \in \mathcal{L}$ alors $(F \wedge G) \in \mathcal{L}$,
- si $F \in \mathcal{L}$ et $G \in \mathcal{L}$ alors $(F \vee G) \in \mathcal{L}$.

On note $\mathbb{L}$ l'ensemble des ensembles qui vérifient les propriétés 1. à 4.
Cette définition d'apparence anodine contient plusieurs présupposés :

- il existe au moins un ensemble qui vérifie ces propriétés,
- on a une notion de "plus petit",
- il existe un plus petit ensemble.

Voici le sens que l'on donne à ces phrases.

- Un ensemble vérifiant les propriétés existe : $\left(\mathcal{A}'\right)^*$ est un exemple.
- Plus petit est défini par l'inclusion. Un élément minimal dans un ensemble d'ensembles $\mathbb{L}$ est alors inclus dans tous les éléments de $\mathbb{L}$. L'unicité d'un éventuel plus petit ensemble est alors acquise car, si L_1 et L_2 sont inclus dans tous les ensembles de $\mathbb{L}$, alors $L_1 \subset L_2$ et $L_2 \subset L_1$ donc $L_1 = L_2$.
- On définit $\mathcal{L} = \bigcap_{L\in\mathbb{L}} L$: montrons que $\mathcal{L}$ répond à toutes nos exigences.
  - On a $\mathcal{A} \subset L$ pour tout L donc $\mathcal{A} \subset \mathcal{L}$,
  - si F appartient à $\mathcal{L}$ alors il appartient à tous les ensembles L donc $(\neg F)$ appartient à tous les ensembles L ; on en déduit que $(\neg F) \in \mathcal{L}$,
  - si F et G appartiennent à $\mathcal{L}$ alors ils appartiennent à tous les ensembles $L \in \mathbb{L}$ donc $(F \wedge G) \in L$ pour tout L d'où $(F \wedge G) \in \mathcal{L}$,
  - de même $(F \vee G) \in \mathcal{L}$ pour tous F et G appartenant à $\mathcal{L}$.

  Ainsi $\mathcal{L}$ vérifie les propriétés de la définition.

  De plus, si L vérifie les propriétés, alors $L \in \mathbb{L}$ donc, par construction, $\mathcal{L} \subset L$: $\mathcal{L}$ est plus petit que tous les ensembles de $\mathbb{L}$.

II.2 Définition par induction

Cette construction de $\mathcal{L}$ ne permet pas facilement de reconnaître les formules.
Voici une autre manière de définir l'ensemble des formules.

1. On défini $\mathcal{L}_0 = \mathcal{A}$.

2. Si $\mathcal{L}_n$ est définit on construit $\mathcal{L}_{n+1}$ par

$$\mathcal{L}_{n+1} = \mathcal{L}_n \cup \{\neg F,\ F \in \mathcal{L}_n\} \cup \{F \wedge G,\ F, G \in \mathcal{L}_n\} \cup \{F \vee G,\ F, G \in \mathcal{L}_n\}$$

Théorème 1 : Définition inductive

$$\mathcal{L} = \bigcup_{n \in \mathbb{N}} \mathcal{L}_n$$

Ce théorème donne en fait seconde définition : on a deux manières de définisr le même objet.
On note $\mathcal{L}_\infty = \bigcup_{n\in\mathbb{N}} \mathcal{L}_n$. La démonstration se fait en deux temps.

$\mathcal{L}_\infty$ vérifie les propriétés 1 à 4 On remarque que $\mathcal{L}_n \subset \mathcal{L}_{n+1}$ donc $\mathcal{L}_p \subset \mathcal{L}_q$ pour $p \leq q$.

- On a $\mathcal{A} = \mathcal{L}_0 \subset \mathcal{L}_\infty$.
- Si F appartient à $\mathcal{L}_\infty$ alors il existe $n \in \mathbb{N}$ tel que $F \in \mathcal{L}_n$.
  On a alors $\neg F \in \mathcal{L}_{n+1}$ d'où $\neg F \in \mathcal{L}_\infty$.
- Si F et G appartient à $\mathcal{L}_\infty$ alors il existe $p, q \in \mathbb{N}$ tel que $F \in \mathcal{L}_p$ et $F \in \mathcal{L}_q$.
  Si on pose $n = \max\{p, q\}$, on a $F \in \mathcal{L}_p \subset \mathcal{L}_n$ et $G \in \mathcal{L}_q \subset \mathcal{L}_n$.
  On en déduit que $(F \wedge G) \in \mathcal{L}_{n+1} \subset \mathcal{L}_\infty$.
- De même $(F \wedge G) \in \mathcal{L}_\infty$ pour F et G dans $\mathcal{L}_\infty$.

Ainsi $\mathcal{L}_\infty$ vérifie les propriétés 1. à 4.

$\mathcal{L}_\infty$ est minimal Soit L vérifiant les propriétés 1. à 4.

On montre par récurrence sur n que alors $\mathcal{L}_n \subset L$ pour tout n.

1. $\mathcal{L}_0 = \mathcal{A} \subset L$.
2. Si $\mathcal{L}_n \subset L$ alors
   pour tout $F \in \mathcal{L}_n$, on a $F \in L$ donc $\neg F \in L : \{\neg F,\ F \in \mathcal{L}_n\} \subset L$,
   pour tout $F, G \in \mathcal{L}_n \subset L$ on a $(F \wedge G) \in E \in L : \{F \wedge G,\ F, G \in \mathcal{L}_n\} \subset L$,
   de même $\{F \vee G,\ F, G \in \mathcal{L}_n\} \subset L$.
   Ainsi l'union des ensembles, $\mathcal{L}_{n+1}$, est incluse dans L.
3. La récurrence montre bien que $\mathcal{L}_n \subset L$ pour tout n.

On en déduit que $\mathcal{L}_\infty = \bigcup_{n\in\mathbb{N}} \mathcal{L}_n \subset L$ pour tout $L \in \mathbb{L}$.
$\mathcal{L}_\infty$ vérifie les 4 propriétés et est inclus dans tout ensemble de $\mathbb{L}$,
c'est bien le plus petit ensemble vérifiant les propriétés 1. à 4. d'où $\mathcal{L}_\infty = \mathcal{L}$.

II.3 Induction structurelle

Cette définition permet d'introduire les démonstrations par **induction structurelle**.
Un propriété est vraie pour toute formule si :

1. elle est vraie pour toutes les variables propositionnelles,
2. lorsqu'elle est vraie pour F, elle est vraie pour $\neg F$,
3. lorsqu'elle est vraie pour F et G, elle est vraie pour $(F \wedge G)$,
4. lorsqu'elle est vraie pour F et G, elle est vraie pour $(F \vee G)$.

De même on peut définir une fonction f sur $\mathcal{L}$ par induction structurelle :

1. on définit $f(x)$ pour toutes les variables propositionnelles,
2. on définit $f(\neg F)$ à partir de $f(F)$,
3. on définit $f(F \wedge G)$ à partir de $f(F)$ et $f(G)$,
4. on définit $f(F \vee G)$ à partir de $f(F)$ et $f(G)$.

II.4 Simplification

L'écriture d'une formule contient beaucoup de parenthèses ce qui peut nuire à la lisibilité.

$$((\neg(x \wedge (y \vee (\neg z)))) \vee ((\neg y) \wedge x))$$

On décide alors d'un ordre de priorité sur les différents connecteurs :

1. $\neg$ est prioritaire devant $\wedge$
2. $\wedge$ est prioritaire devant $\vee$.
3. dans une association de connecteurs ($\wedge$ ou $\vee$) les priorités vont de la gauche vers la droite.

Exemples

1. $\big((x \wedge y) \vee z\big)$ pourra s'écrire $x \wedge y \vee z$,
2. $\big((\neg x) \wedge y\big)$ pourra s'écrire $\neg x \wedge y$
3. $((((x_1 \vee x_2) \vee x_3) \vee x_4) \vee x_5)$ pourra s'écrire $x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$.
4. Par contre les parenthèses dans la formule $\neg(x \vee y)$ ne peuvent pas être enlevées sous peine de changer la nature de la formule.

On admettra que la simplification est injective : deux formules distinctes ne peuvent pas donner la même simplification.

II.5 Arbre syntactique

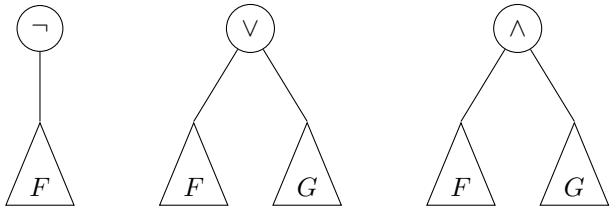
On définit par induction structurelle un arbre à partir d'une formule.

1. Une variable propositionnelle x est représentée par 

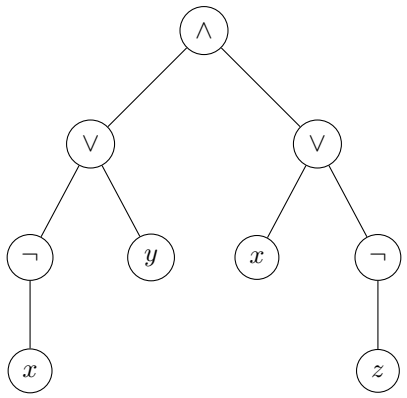
2. Si F est représentée par 

et G par 

,
on représente $(\neg F)$, $(F \vee G)$ et $(F \wedge G)$ respectivement par



Ainsi la formule (simplifiée) $(\neg x \vee y) \wedge (x \vee \neg z)$ est représentée par :



Cette représentation en arbre donne un type possible pour les formules.

```
type formule =  X of int
                |Non of formule
                |Ou of formule*formule
                |Et of formule*formule;;
```

`X k` désignera une variable booléenne indexée par k que l'on notera x_k.
Par exemple la proposition $x_1 \vee \big(\neg(x_2 \vee \neg x_1) \wedge x_0\big)$ sera implantée par

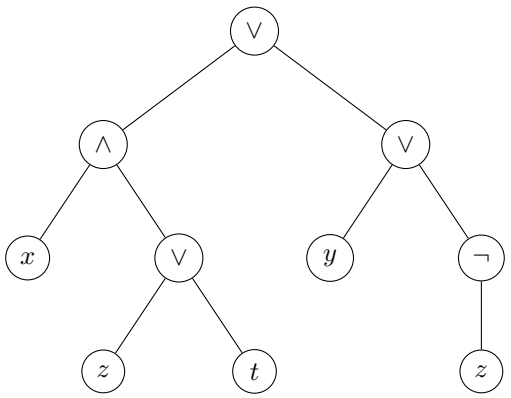
```
let p_ex = Ou (X 1, Et (Non (Ou (X 2, Non (X 1))), X 0));;
```

Nous n'utiliserons que des variables indexées par des entiers positifs.

II.6 Exercices

Exercice 1
Dessiner la représentation arborescente de la formule $((x \vee (x \vee y)) \wedge (\neg x))$.

Exercice 2
Donnez la formule représentée par l'arbre suivant :



On veut pouvoir écrire les expression booléennes de manière plus lisible.
Par exemple, la formule `p_ex` pourra être imprimée sous la forme

```
x1 V (!(x2 V !x1) & x0))
```

Exercice 3
Écrire une fonction `montrer` qui permet cette visualisation.
La concaténation de chaînes se fait avec `str1^str2`
et on convertit un entier en chaîne par `string_of_int`.

Exercice 4
Écrire une fonction `multiOu` de type `formule List -> formule` qui, en recevant une liste non vide de propositions logiques `[P1; P2; ...; Pn]`, renvoie la disjonction des P_i : $\Big(P_1 \vee \big(P_2 \vee \cdots (P_{n-1} \vee P_n) \cdots\big)\Big)$.
Écrire de même une fonction `multiEt`

III Sémantique

La sémantique d'une formule est le caractère vrai ($\mathbf{V}$) ou faux ($\mathbf{F}$) de la formule.
Pour cela on définit une valeur de vérité à chacune des variables propositionnelles puis on calcule la valeur de vérité de la formule.

Définition 3 : Valuation

Une valuation des variables propositionnelles est une application de $\mathcal{A}$, l'ensemble des variables propositionnelles, vers $\mathcal{B} = \{\mathbf{V}, \mathbf{F}\}$.

La sémantique consiste à prolonger une valuation v en une application $\widetilde{v}$ de l'ensemble des formules, $\mathcal{L}$, vers l'ensemble des valeurs de vérité, $\mathcal{B}$. Cela revient à donner une signification aux connecteurs.
On donne le sens suivant :

1. la négation pour $\neg$,
2. ou (disjonction) pour $\vee$, c'est un ou **non exclusif**,
3. et (conjonction) pour $\wedge$.

On définit $\widetilde{v}$ par induction structurelle :

1. si $x \in \mathcal{A}$ alors $\widetilde{v}(x) = v(x)$,
2. si $F, G \in \mathcal{L}$ alors $\widetilde{v}(\neg F)$, $\widetilde{v}(F \wedge G)$ et $\widetilde{v}(F \vee G)$ sont définis par les tables de vérité.

$\widetilde{v}(F)$	$\widetilde{v}(G)$	$\widetilde{v}(\neg F)$	$\widetilde{v}(F \wedge G)$	$\widetilde{v}(F \vee G)$
V	**V**	**F**	**V**	**V**
V	**F**	**F**	**F**	**V**
F	**V**	**V**	**F**	**V**
F	**F**	**V**	**F**	**F**

Exercice 5
Soit $x, y \in \mathcal{A}$ et soit v une valuation telle que $v(x) = \mathbf{V}$ et $v(y) = \mathbf{F}$,
calculer $\widetilde{v}\big(y \vee (x \wedge y)\big)$

III.1 Implantation

On rappelle que les variable booléennes sont implantées sous la forme `X k`.
Si les variables booléennes d'une proposition ont un indice majoré par $n-1$ on représentera la partie utile d'une valuation par un tableau de booléens de taille n, `val`. La valeur attribuée à `X k` sera `val.(k)`.

Exercice 6
Écrire une fonction **`eval`** qui renvoie la valeur booléenne d'une proposition booléenne pour une valuation. Par exemple

```
eval p_ex [|true; false; true|];;
#- : bool = true
```

III.2 Satisfiabilité

Définition 4 : Formules satisfaites

v désigne une valuation, F désigne une formule.

1. Si $\widetilde{v}(F)$ est **V** on dit que v **satisfait** F et que F **est satisfaite** par v.
2. F est **satisfiable** s'il existe au moins une valuation qui satisfait F.
3. F est une **tautologie** si elle est satisfaite par toutes les valuations.
4. F est une **contradiction** si elle n'est satisfaite par aucune valuation.

Exemples : $F \vee \neg F$ est une tautologie et $F \wedge \neg F$ est une contradiction pour toute formule F.

Remarques

1. F est une contradiction si et seulement si $\neg F$ est une tautologie.
2. F est satisfiable si et seulement si elle n'est pas une contradiction.
3. Si la formule F contient n variables propositionnelles distinctes, il existe 2^n valuations possibles de ces n variables. Pour tester si la formule est une tautologie (ou une contradiction) il faut évaluer la formule pour chacune de ces valuation.

   Déterminer si une formule est satisfiable se fait avec une complexité exponentielle.
4. Par contre vérifier qu'une valuation donnée satisfait F ne prend que le temps de l'évaluation.

L'étude d'une formule peut se faire en calculant sa table de vérité : on représente toutes les valuations concernant les variables de la formule puis on détermine les valuation des composants de F pour aboutir à F. On indiquera souvent F à la place de $\widetilde{v}(F)$.
Exemple : $F = (\neg x \vee y) \wedge (x \vee \neg z)$

x	y	z	$\neg x$	$\neg x \vee y$	$\neg z$	$x \vee \neg z$	F
V	**V**	**V**	**F**	**V**	**F**	**V**	**V**
V	**V**	**F**	**F**	**V**	**V**	**V**	**V**
V	**F**	**V**	**F**	**F**	**F**	**V**	**F**
V	**F**	**F**	**F**	**F**	**V**	**V**	**F**
F	**V**	**V**	**V**	**V**	**F**	**F**	**F**
F	**V**	**F**	**V**	**V**	**V**	**V**	**V**
F	**F**	**V**	**V**	**V**	**F**	**F**	**F**
F	**F**	**F**	**V**	**V**	**V**	**V**	**V**

Exercice 7 - Tables de vérités avec 2 variables
Écrire toutes les tables de vérité correspondant aux expressions construites sur un ensemble de deux variables propositionnelles. Identifier pour chaque table une expression simple dont elle est l'évaluation.

IV Équivalence

IV.1 Définition

Définition 5 : Formules équivalentes

eux formules F, G sont équivalentes si, et seulement si, pour toute valuation v, on a $\widetilde{v}(F) = \widetilde{v}(G)$.
Cette équivalence est notée $F \equiv G$.

Deux formules équivalentes ont donc la même table de vérité.
Exemple : $G = x \wedge y \vee \neg(x \vee z)$

x	y	z	$x \wedge y$	$x \vee z$	$\neg(x \vee z)$	G
V	**V**	**V**	**V**	**V**	**F**	**V**
V	**V**	**F**	**V**	**V**	**F**	**V**
V	**F**	**V**	**F**	**V**	**F**	**F**
V	**F**	**F**	**F**	**V**	**F**	**F**
F	**V**	**V**	**F**	**V**	**F**	**F**
F	**V**	**F**	**F**	**F**	**V**	**V**
F	**F**	**V**	**F**	**V**	**F**	**F**
F	**F**	**F**	**F**	**F**	**V**	**V**

On retrouve la table de vérité de $F = (\neg x \vee y) \wedge (x \vee \neg z)$ donc $F \equiv G$.

Théorème 2 : Hérédité

Si F est équivalente à F' alors on peut remplacer une apparition de F par F' dans une formule G pour obtenir une formule équivalente G'.

Cela se démontre par induction structurelle sur la construction de G à partir de F.

1. Le cas de base est évident : $F \equiv F'$
2. Si $G \equiv G'$ alors $\neg G \equiv \neg G'$
3. Si $G \equiv G'$ et $H \equiv H'$ alors $(G \vee H) \equiv (G' \vee H')$ et $(G \wedge H) \equiv (G' \wedge H')$

Les implications ci-dessus sont des conséquences du calcul de $\widetilde{v}$: $\widetilde{v}(\neg G)$, $\widetilde{v}(G \vee H)$ et $\widetilde{v}(G \wedge H)$ ne dépendent que de $\widetilde{v}(G)$ et $\widetilde{v}(H)$ qui sont égaux respectivement à $\widetilde{v}(G')$ et à $\widetilde{v}(H')$.

Théorème 3 : Simplifications

1. $\neg(\neg F) \equiv F$
2. Si G est une tautologie alors $F \wedge G \equiv F$
3. Si G est une contradiction alors $F \vee G \equiv F$

Théorème 4 : Distributivité

1. $F \wedge (G \vee H) \equiv F \wedge G \vee F \wedge H$
2. $F \vee G \wedge H \equiv (F \vee G) \wedge (F \vee H)$

Théorème 5 : Lois de Morgan

1. $\neg(F \wedge G) \equiv \neg F \vee \neg G$
2. $\neg(F \vee G) \equiv \neg F \wedge \neg G$

IV.2 Formes normales

Dans l'étude des formules il peut être utile de travailler avec une forme standardisée des formules. Nous allons définir 2 telles formes. Bien qu'elles soient semblables elles correspondent en fait à deux états différents de la connaissance de la formule : la forme normale conjonctive représente une formule initiale, on ne sait pratiquement rien de sa table de vérité, la forme normale disjonctive est très proche de la table de vérité.

Définition 6 : Formes normales

- Un **littéral** est une variable propositionnelle ou la négation d'une variable propositionnelle.
- Une **clause** (disjonctive) est une disjonction de littéraux.
  C'est une formule de la forme $p_1 \vee \cdots \vee p_n$ où $n \geq 1$ et $p_1, \ldots, p_n$ sont des littéraux.
- Une **clause conjonctive** est une conjonction de littéraux.
  C'est une formule de la forme $p_1 \wedge \cdots \wedge p_n$ où $n \geq 1$ et $p_1, \ldots, p_n$ sont des littéraux.
- Une **forme normale conjonctive** est conjonction de clause.
  C'est une formule de la forme $C_1 \wedge \cdots \wedge C_m$ où $m \geq 1$ et $C_1, \ldots, C_m$ sont des clauses.
- Une **forme normale disjonctive** est disjonction de clauses conjonctives :
  $D_1 \vee \cdots \vee D_m$.

Par exemple la formule $F = x \wedge \neg y \vee y \wedge \neg z$ est une forme normale disjonctive.

On peut construire une forme normale disjonctive équivalente à F à partir de sa table de vérité.

1. F est une formule de variables x_1, x_2, …, x_n.
2. V_F est l'ensemble des valuations v des n variables telles que $v(F) = \mathbf{V}$.
3. Pour une valuation v et pour chaque variable x on définit le littéral
   $l_v(x) = x$ si $v(x) = \mathbf{V}$ et $l_v(x) = \neg x$ si $v(x) = \mathbf{F}$.
4. On pose alors $D_v = \bigwedge_{i=1}^{n} l_v(x_i)$ et $F_N = \bigvee_{v \in V_F} D_v$.

Théorème 6 : Forme normale canonique

F_N est une forme normale disjonctive équivalente à F.
Ainsi toute formule admet une forme normale disjonctive équivalente.

Les lois de Morgan impliquent que la négation d'une forme normale disjonctive est une forme normale conjonctive. On obtient donc une forme normale conjonctive équivalente à F en déterminant comme ci-dessus une forme normale disjonctive équivalente à $\neg F$ puis en en calculant la négation.

Théorème 7 : Forme normale canonique

Toute formule admet une forme normale conjonctive équivalente.

Exemple : on revient à $F = (\neg x \vee y) \wedge (x \vee \neg z)$

x	y	z	$\neg x$	$\neg x \vee y$	$\neg z$	$x \vee \neg z$	F	$\neg F$
V	**V**	**V**	**F**	**V**	**F**	**V**	**V**	**F**
V	**V**	**F**	**F**	**V**	**V**	**V**	**V**	**F**
V	**F**	**V**	**F**	**F**	**F**	**V**	**F**	**V**
V	**F**	**F**	**F**	**F**	**V**	**V**	**F**	**V**
F	**V**	**V**	**V**	**V**	**F**	**F**	**F**	**V**
F	**V**	**F**	**V**	**V**	**V**	**V**	**V**	**F**
F	**F**	**V**	**V**	**V**	**F**	**F**	**F**	**V**
F	**F**	**F**	**V**	**V**	**V**	**V**	**V**	**F**

Une forme normale disjonctive équivalente à F est :

$$(x \wedge y \wedge z) \vee (x \wedge y \wedge \neg z) \vee (\neg x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge \neg z)$$

Une forme normale disjonctive équivalente à $\neg F$ est :

$$(x \wedge \neg y \wedge z) \vee (x \wedge \neg y \wedge \neg z) \vee (\neg x \wedge y \wedge z) \vee (\neg x \wedge \neg y \wedge z)$$

Ainsi F est équivalente à la forme normale conjonctive

$$(\neg x \vee y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \vee \neg y \neg z) \wedge (x \vee \vee y \neg z)$$

On retrouve la forme normale conjonctive initiale en regroupant les clauses deux par deux et en utilisant la distributivité, la forme intermédiaire étant $(\neg x \vee y \vee z \wedge \neg z) \wedge (x \vee y \wedge \neg y \vee \neg z)$.

IV.3 Exercices

Exercice 8
Écrire une fonction (récursive) `uneSeule` qui prend pour argument une liste de propositions et calcule une proposition qui est vraie si et seulement s'il y a exactement une des propositions de la liste qui est vraie.

Exercice 9 - Démonstration du théoreme 3
Prouver les lois de simplification

1. $\neg(\neg F) \equiv F$
2. Si G est une tautologie alors $F \wedge G \equiv F$
3. Si G est une contradiction alors $F \vee G \equiv F$

Exercice 10 - Démonstration du théoreme 5
Prouver les lois de Morgan

1. $\neg(F \wedge G) \equiv \neg F \vee \neg G$
2. $\neg(F \vee G) \equiv \neg F \wedge \neg G$

Exercice 11 - Démonstration du théoreme 4
Prouver les lois de distributivité

1. $F \wedge (G \vee H) \equiv F \wedge G \vee F \wedge H$
2. $F \vee G \wedge H \equiv (F \vee G) \wedge (F \vee H)$

Exercice 12
Prouver les équivalences suivantes.

1. **Idempotence** : $F \wedge F \equiv F$.
2. **Absorption** : $F \wedge (F \vee G) \equiv F$.

Exercice 13 - Démonstration du théoreme 6
Prouver que la forme normale disjonctive F_N définie dans la préambule du théorème est équivalente à F.

V Autres connecteurs

V.1 Implication

On définit le connecteur logique $\Rightarrow$ par $F \Rightarrow G \equiv \neg F \vee G$.

Exercice 14
Écrire une fonction `implique` qui calcule une proposition logiquement équivalente à $P_1 \Rightarrow P_2$ à partir de deux propositions P_1 et P_2.

Exercice 15
Prouver les équivalences (classiques) suivantes.

Contraposition $\big[F \Rightarrow G\big] \equiv \big[\neg G \Rightarrow \neg F\big]$

Exportation $\big[F \Rightarrow (G \Rightarrow H)\big] \equiv \big[F \wedge G \Rightarrow H\big]$

Exercice 16 - Loi de Pierce
Prouver que $\big((F \Rightarrow G) \Rightarrow F\big) \Rightarrow F$ est une tautologie.

Exercice 17
Soit p, q et r trois variables propositionnelles, déterminer les tables de vérité des formules $F = (p \Rightarrow q) \vee (q \Rightarrow r)$ et $G = (p \vee q) \Rightarrow (q \wedge r)$.

V.2 Si et seulement si

On définit le connecteur logique $\Leftrightarrow$ par $\big[F \Leftrightarrow G\big] \equiv \big[(\neg F \vee G) \wedge (\neg G \vee F)\big]$.

Exercice 18
Écrire une fonction `equivaut` qui calcule une proposition logiquement équivalente à $P_1 \Rightarrow P_2$ à partir de deux propositions P_1 et P_2.
Écrire de même une fonction `ouExclusif`.

Exercice 19
Prouver que $\big[F \Leftrightarrow G\big] \equiv \big[F \wedge G \vee \neg F \wedge \neg F\big]$.

Exercice 20
Montrer que $F \equiv G$ si et seulement si $F \Leftrightarrow G$ est une tautologie.

V.3 If Then Else

On définit une fonction logique **ite** à 3 variables telle que

$$\mathbf{ite}\,(F, G, H) \equiv (F \wedge G) \vee (\neg F \wedge H)$$

Exercice 21
Quelle est la valeur de $\widetilde{v}\big(\mathbf{ite}\,(F, G, H)\big)$ si $\widetilde{v}(F) = \mathbf{V}$?
Même question si $\widetilde{v}(F) = \mathbf{F}$.

Exercice 22
Exprimer $F \wedge G$ et $F \vee G$ par une formule simple comportant un seul **ite**.

On définit les constantes **0** et **1** telles que $\widetilde{v}(\mathbf{0}) = \mathbf{F}$ et $\widetilde{v}(\mathbf{1}) = \mathbf{V}$ pour toute valuation v.

Exercice 23
Exprimer $\neg F$ par une formule comportant un seul **ite** et les constantes **0** et **1**.
Donner une expression simple équivalente à **ite** $(F, \mathbf{1}, \mathbf{0})$

On peut donc écrire toute formule logique en n'utilisant que les constantes **0** et **1** et l'opérateur **ite** : ce sont les **IF-expressions**.
On définit un nouveau type pour travailler avec ces IF-expressions.

```
type ifExpr =
  |VarI of string
  |Vrai (* c'est le 1 *)
  |Faux (* C'est le 0 *)
  |Ite of ifExpr * ifExpr * ifExpr;;
```

Exercice 24
En déduire une fonction `prop2if` qui transforme une proposition en une IF-expression équivalente n'utilisant que le connecteur **ite**.

V.4 Le connecteur de Sheffer

Le connecteur de Sheffer, noté $|$, est défini par :

$$[F|G] \equiv [\neg F \vee \neg G]$$

Exercice 25
Montrez que les formules $\neg F$ et $F|F$ sont équivalentes.

Exercice 26
Montrez que $F \vee G$ et $(F|F)|(G|G)$ sont des formules équivalentes.

Exercice 27
En déduire que toute formule est équivalent à une formule ne contenant que des variables propositionnelles et le connecteur de Sheffer.

On ajoute un nouveau type

```
type formuleS =  VarS of string |Shf of formuleS * formuleS;;
```

Exercice 28
Écrire une fonction qui transforme une formule avec le type défini dans le chapitre en une formule du nouveau type.

A Un algorithme de résolution par force brute

On se propose ici de d'étudier le problème de la satisfiabilité de manière élémentaire.
Le problème que nous allons essayer de résoudre se traduit par

- Une formule est-elle satisfiable ?
- Si oui donner une valuation qui la satisfait.

Exercice 29
Écrire une fonction `indiceMax` qui renvoie l'indice de variable maximal rencontré dans une proposition logique.

Pour pouvoir décrire toutes les valuations, on les construit une-par-une.
Pour cela nous allons utiliser une fonction `incremente` qui modifie un tableau de booléens (sans renvoyer sa valeur) et qui renvoie `true` sauf quand le tableau n'est pas incrémentable.
On suit l'algorithme qui correspond à l'incrémentation d'un entier exprimé en base 2 :

- on modifie les indices les plus petits de `true` à `false`
- s'il existe, le premier `false` est modifé en `true` et la fonction renvoie `true`
- s'il n'y avait que des `true` la fonction renvoie `false` pour signifier que toutes les valuations ont été parcourues.

On part alors de `[|false; false; ... ; false|]` et, après 2^n incrémentations, on aboutit à `[|true; true; ... ; true|]`

Exercice 30
Écrire la fonction `incremente`.

Exercice 31
Écrire une fonction `possibles` qui renvoie la liste des tableaux satisfaisant une proposition logique. On peut copier un tableaux à l'aide de la fonction `Array.copy`.

Exercice 32
Écrire deux fonctions `satisfiable` et `tautologie` qui déterminent respectivement si une proposition logique est satisfiable et si c'est une tautologie.

Exercice 33
Vérifier que les propositions suivantes sont des tautologies.

1. $\neg\neg x_1 \Leftrightarrow x_1$
2. $(x_1 \Rightarrow x_2) \Leftrightarrow (\neg x_2 \Rightarrow \neg x_1)$
3. $(x_1 \Rightarrow x_2) \wedge (x_2 \Rightarrow x_3) \Rightarrow (x_1 \Rightarrow x_3)$

B Un algorithme de résolution : les if-expressions

On va donner un algorithme de décision sur les formules logiques plus sphistiqué que le précédent. On utilise ici les IF-expressions de la partie V.3.

Exercice 34 -
Donner les IF-expressions correspondant aux trois propositions suivantes :

$$\begin{aligned} P_1 &: (A \wedge B) \to A \\ P_2 &: A \to (A \wedge B) \\ P_3 &: \big(A \to (A \to B)\big) \to B \end{aligned}$$

B.1 Forme normale

On dit qu'une IF-expression est **normale** si elle est

- une variable,
- $\mathbf{0}$ ou $\mathbf{1}$,
- ou $\mathbf{ite}\,(s, P, Q)$ où s est une variable et P et Q sont deux IF-expressions normales.

Exercice 35
Écrire une fonction `estNormale : ifExpr -> bool`
qui teste si une IF-expression est normale.

On considère la fonction ϕ définie récursivement par

- $\phi(\mathbf{V}) = \phi(\mathbf{F}) = 1$
- $\phi(s) = 1$ pour toute variable s
- $\phi\big(\mathbf{ite}\,(M, P, Q)\big) = \phi(M)\big(1 + \phi(P) + \phi(Q)\big)$

Exercice 36
Prouver que les transformations suivantes diminuent la valeur de ϕ.

$$\begin{aligned} \mathbf{ite}\,(\mathbf{V}, P, Q) &\rightsquigarrow P \\ \mathbf{ite}\,(\mathbf{F}, P, Q) &\rightsquigarrow Q \\ \mathbf{ite}\,(\mathbf{ite}\,(M, P, Q), R, S) &\rightsquigarrow \mathbf{ite}\,\big(M, \mathbf{ite}\,(P, R, S), \mathbf{ite}\,(Q, R, S)\big) \end{aligned}$$

En déduire toute IF-expression est équivalente à une IF-expression normale.

Exercice 37
Écrire une fonction `normalise : ifExpr -> ifExpr`
qui transforme toute IF-expression en une IF-expression normale équivalente.

Exercice 38
Donner les IF-expressions normales correspondant à P_1, P_2 et P_3 de l'exercice 34.

B.2 Valuation partielle

Définition 7 : valuation partielle

Une valuation partielle est une fonction des variables dans $\{\textbf{Vrai}, \textbf{Faux}\}$ dont le domaine est fini.

Un programme informatique ne peut gérer quer des valuations partielles.

Définition 8 : valuation compatible

Une valuation partielle α est compatible avec une proposition P si les variables de P sont dans le domaine de définition de α.

On représentera une valuation partielle par une liste d'association :

```
type valuation = (string * bool) list;;
```

Exercice 39
Écrire une fonction `defini : string -> valuation -> bool` qui renvoie `true` ou `false` selon que la valuation partielle est ou non définie pour une variable représentée par une chaîne de caractères.

Exercice 40
Écrire une fonction `valeur : string -> valuation -> bool` qui renvoie la valeur prise par une valuation partielle pour une variable représentée par une chaîne de caractères.

On prolonge la notion de tautologie par la notion de α-tautologie de la manière suivante :

Définition 9 : α-tautologie

Si α est une valuation partielle, une expression P est une α-tautologie si elle est vraie pour toutes les valuations partielles qui prolongent α et qui sont compatibles avec P.

Ainsi P est une tautologie si et seulement si elle est une ω-tautologie où ω est la valuation partielle de domaine vide.

B.3 Algorithme

Nous allons étudier un algorithme qui détermine si une IF-expression normale P est une α-tautologie ou, si elle ne l'est pas, détermine une valuation partielle compatible avec P, qui prolonge α et qui réfute P.
Cet algorithme est décrit par les règles suivantes.

1. Si P est **1** ou **0** le résultat est immédiat.
2. Si p est une variable `s` alors trois cas se présentent :
   - si $\alpha(s)$ est défini et vaut **V** alors P est une α-tautologie,
   - si $\alpha(s)$ est défini et vaut **F** alors P n'est pas une α-tautologie, réfutée par α,
   - si $\alpha(s)$ n'est pas défini alors P n'est pas une α-tautologie
     et on obtient une réfutation en étendant α par $\alpha(s) = \mathbf{F}$.
3. Si P est une IF-expression `ite(s, Q, R)` alors
   - si $\alpha(s)$ est défini et vaut **V**
     alors P est une α-tautologie si et seulement si Q est une α-tautologie,
   - si $\alpha(s)$ est défini et vaut **F**
     alors P est une α-tautologie si et seulement si R est une α-tautologie,
   - si $\alpha(s)$ n'est pas défini alors on définit deux prolongements de α, α_v et α_f, par $\alpha_v(s) = \mathbf{V}$ et $\alpha_f(s) = \mathbf{F}$. P est une α -tautologie si et seulement si Q est une α_v-tautologie et R est une α_f-tautologie.

Exercice 41
Si `P = ite(s, Q, R)` n'est pas une α-tautologie donner une réfutation dans chacun des 3 cas précédents.

On définit donc le type suivant pour gérer le résultat de l'algorithme.

```
type resultat = Tautologie | Refutation of valuation;;
```

Exercice 42
Écrire une fonction `decisionP : valuation -> ifExpr -> resultat` qui décide si une IF-expression normale est une tautologie par rapport à une valuation (partielle) et donne une réfutation si elle ne l'est pas.

Exercice 43
En déduire une fonction `decisionIF : ifExpr -> resultat` qui décide si une IF-expression normale est une tautologie et donne une réfutation si elle ne l'est pas.

Exercice 44
Écrire enfin une fonction `decision : proposition -> resultat` qui décide si une proposition est une tautologie et donne une réfutation si elle ne l'est pas.

C Applications

On donne ici quelques problèmes dont la résolution peut être faite à la main parfois mais que les méthodes ci-dessus permettent de résoudre.

C.1 Le désert

Vous êtes perdu sur une piste dans le désert. Vous arrivez à une bifurcation. Chacune des deux pistes est gardée par un sphinx que vous pouvez interroger. Les pistes peuvent soit conduire à une oasis soit se perdre dans un désert profond (au mieux elles conduisent toutes à une oasis, au pire elles se perdent toutes les deux).
Vous disposez des informations suivantes :

1. A : le sphinx de droite dit : "*Une au moins des deux pistes conduit à une oasis*".
2. B : le sphinx de gauche dit : "*La piste de droite se perd dans le désert*".
3. C : vous savez que les sphinx disent tous les deux la vérité ou bien mentent tous les deux.

D est la proposition "*Il y a une oasis au bout de la route de droite*" et G est la proposition "*Il y a une oasis au bout de la route de gauche*".

Exercice 45

- Exprimer par une formule les affirmations A et B.
- Exprimer alors la connaissance C.
- Résoudre l'énigme.

C.2 Les dahuts

Le dahut est une espèce très rare de bouquetin qui a la particularité d'avoir les deux pattes d'un coté plus courtes que les autres. Il vit donc dans la montagne en ayant toujours le sommet du coté de ses pattes courtes. Un dahut est appelé de dextrogyre ou lévogyre selon les cas.
Ils ont d'autres particularités :

- Tout dahut non lévogyre a des rayures noires.
- Tout dahut qui a des oreilles blanches est lévogyre et vit dans les forêts.
- Tout dahut a des oreilles blanches ou n'a pas de rayures noires.
- Les dahuts qui vivent dans les forêts ne mangent pas de mulots.
- Un dahut mange des mulots si et seulement s'il est lévogyre.
- Tout dahut lévogyre a des oreilles blanches.

Exercice 46
Prouver que les dahuts n'existent pas.

C.3 La princesse

Un chevalier doit partir délivrer des princesses. Arrivé à une intersection, il a le choix entre trois chemins, chacun précédé d'un panneau. Le gardien des lieux lui déclare :

"*Parmi ces trois chemins, l'un mène à une princesse, et son panneau dit la vérité.*
Quant aux deux autres, ils aboutissent à une mort certaine.
Au moins l'un des panneaux ment."
Voici ce qui est écrit à l'entrée de chaque chemin :

- Le deuxième chemin mène à une mort certaine.
- Ce chemin mène à une mort certaine.
- Le premier chemin mène à une mort certaine.

Exercice 47
Comment délivrer la princesse ?

C.4 Le concours

Vous êtes admissible à l'Université de Logique Médiévale (ULM) et vous devez passer l'épreuve de logique pratique. Depuis le Hall d'entrée il y a 5 couloirs et vous savez qu'un seul mène à la salle d'examen.
Chaque couloir est indiqué par un panneau et vous savez aussi que le panneau qui mène à votre salle dit la vérité mais que les autres peuvent mentir.
Voici ce que disent les cinq panneaux.

1. Ce couloir vous mène à l'échec mais pas le quatrième.
2. C'est un chemin de numéro impair qui mène dans la salle.
3. Le deuxième panneau dit la vérité ou le cinquième ment.
4. Ce panneau ment mais pas le premier.
5. Le troisième panneau ment.

Exercice 48
Comment arriver à votre salle ?

C.5 Le cadeau

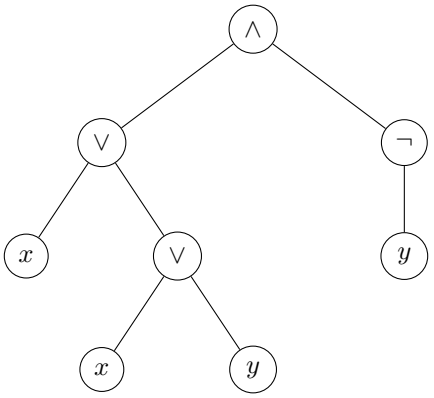
Vous êtes admis dans l'école de vos rêves ! Pour vous accueillir les camarades de deuxième année vous offrent un Kinder. Mais il faut le mériter : vous avez devant vous 9 boîtes dont une seule contient l'œuf. De plus chacune a une étiquette et votre parrain vous annonce que la boîte qui contient votre cadeau porte une étiquette qui dit la vérité les autres peuvent mentir.
Voici les énoncés.

1. C'est une boîte de numéro impair qui contient la récompense
2. Ouvre-moi si tu veux du chocolat.
3. La cinquième étiquette dit la vérité ou la septième ment. Mais la neuvième ment.
4. La première étiquette ment.
5. Parmi la deuxième et la quatrième étiquette, il y en a au moins une qui dit la vérité.
6. La troisième étiquette ment.
7. La première boîte est vide.
8. Je ne dis pas la vérité et la neuvième boîte contient le cadeau.
9. Il ne faut pas croire la sixième étiquette.

Exercice 49
Quelle boîte ouvrez-vous ?

Solutions

Solution de l'exercice 1



Solution de l'exercice 2

$((x \wedge (z \vee t)) \vee (y \vee (\neg z)))$

Solution de l'exercice 3

```
let symOu = " V ";;
let symEt = " & ";;
let symNon = "!";;
let rec montrer p =
   match p with
   |X k -> "x"^(string_of_int k)
   |Non q -> symNon^(montrer q)
   |Ou (q, r) -> "("^(montrer q)^symOu^(montrer r)^")"
   |Et (q, r) -> "("^(montrer q)^symEt^(montrer r)^")";;
```

Solution de l'exercice 4

```
let rec multiOu listeX =
   match listeX with
   |[] -> failwith "La liste est vide"
   |[p] -> p
   |t::q -> Ou(t, multiOu q);;

let rec multiEt listeX =
   match listeX with
   |[] -> failwith "La liste est vide"
   |[p] -> p
   |t::q -> Et(t, multiEt q);;
```

Solution de l'exercice 5

Le résultat est **F** .

Solution de l'exercice 6

```
let rec eval X t =
  match X with
  |Var(k) -> t.(k-1)
  |Non(q) -> not (eval q t)
  |Ou(q, r) -> (eval q t) || (eval r t)
  |Et(q, r) -> (eval q t) && (eval r t);;
```

Solution de l'exercice 7 - Tables de vérités avec 2 variables
On note φ la fonction associée à la table.

$\varphi(\mathbf{V},\mathbf{V})$	$\varphi(\mathbf{V},\mathbf{F})$	$\varphi(\mathbf{F},\mathbf{V})$	$\varphi(\mathbf{F},\mathbf{F})$	$\varphi(x,y)$
V	**V**	**V**	**V**	$x \vee \neg x$
V	**V**	**V**	**F**	$x \vee y$
V	**V**	**F**	**V**	$x \vee \neg y$
V	**V**	**F**	**F**	x
V	**F**	**V**	**V**	$\neg x \vee y$
V	**F**	**V**	**F**	y
V	**F**	**F**	**V**	$x \Leftrightarrow y$
V	**F**	**F**	**F**	$x \wedge y$
F	**V**	**V**	**V**	$\neg(x \wedge y)$
F	**V**	**V**	**F**	$x \Leftrightarrow \neg y$
F	**V**	**F**	**V**	$\neg y$
F	**V**	**F**	**F**	$x \wedge \neg y$
F	**F**	**V**	**V**	$\neg x$
F	**F**	**V**	**F**	$\neg x \wedge y$
F	**F**	**F**	**V**	$\neg(x \vee y)$
F	**F**	**F**	**F**	$x \wedge \neg x$

Solution de l'exercice 8

```
let rec uneSeule listeX =
  match listeX with
  |[] -> failwith "La liste est vide"
  |[p] -> p
  |t::q -> Ou(Et(Non t, uneSeule q), Et(t, Non (multiOu q)));;
```

Solution de l'exercice 9 - Démonstration du théoreme 3

1. On a $\neg(\neg\mathbf{V}) = \neg\mathbf{F} = \mathbf{V}$ et $\neg(\neg\mathbf{F}) = \neg\mathbf{V} = \mathbf{F}$; pour toute valuation v,
2. G est une tautologie donc $\widetilde{v}(G) = \mathbf{V}$ pour toute valuation v.
   On a $\mathbf{F} \wedge \mathbf{V} = \mathbf{F}$ et $\mathbf{V} \wedge \mathbf{V} = \mathbf{V}$; ainsi, pour toute valuation v,
   $\widetilde{v}\big(F \wedge G\big) = \widetilde{v}\big(F\big) \wedge \widetilde{v}\big(G\big) = \widetilde{v}\big(F\big) \wedge \mathbf{V} = \widetilde{v}\big(F\big)$ d'où l'équivalence $F \wedge G \equiv F$.
3. G est une contradiction donc $\widetilde{v}(G) = \mathbf{F}$ pour toute valuation v.
   On a $\mathbf{F} \vee \mathbf{F} = \mathbf{F}$ et $\mathbf{V} \vee \mathbf{F} = \mathbf{V}$; ainsi, pour toute valuation v,
   $\widetilde{v}\big(F \vee G\big) = \widetilde{v}\big(F\big) \vee \widetilde{v}\big(G\big) = \widetilde{v}\big(F\big) \vee \mathbf{F} = \widetilde{v}\big(F\big)$ d'où l'équivalence $F \vee G \equiv F$.

Solution de l'exercice 10 - Démonstration du théoreme 5

F	G	$G \wedge H$	$\neg(F \wedge G)$	$\neg F$	$\neg G$	$\neg F \vee \neg G$
V	**V**	**V**	**F**	**F**	**F**	**F**
V	**F**	**F**	**V**	**F**	**V**	**V**
F	**V**	**F**	**V**	**V**	**F**	**V**
F	**F**	**F**	**V**	**V**	**V**	**V**

On a bien l'équivalence $\neg(F \wedge G) \equiv \neg F \vee \neg G$.

F	G	$G \vee H$	$\neg(F \vee G)$	$\neg F$	$\neg G$	$\neg F \wedge \neg G$
V	**V**	**V**	**F**	**F**	**F**	**F**
V	**F**	**V**	**F**	**F**	**V**	**F**
F	**V**	**V**	**F**	**V**	**F**	**F**
F	**F**	**F**	**V**	**V**	**V**	**V**

On a bien l'équivalence $\neg(F \vee G) \equiv \neg F \wedge \neg G$.

Solution de l'exercice 11 - Démonstration du théoreme 4
On calcule les tables de vérités

F	G	H	$G \vee H$	$F \wedge (G \vee H)$	$F \wedge G$	$F \wedge H$	$F \wedge G \vee F \wedge H$
V	**V**	**V**	**V**	**V**	**V**	**V**	**V**
V	**V**	**F**	**V**	**V**	**V**	**F**	**V**
V	**F**	**V**	**V**	**V**	**F**	**V**	**V**
V	**F**	**F**	**F**	**F**	**F**	**F**	**F**
F	**V**	**V**	**V**	**F**	**F**	**F**	**F**
F	**V**	**F**	**V**	**F**	**F**	**F**	**F**
F	**F**	**V**	**V**	**F**	**F**	**F**	**F**
F	**F**	**F**	**F**	**F**	**F**	**F**	**F**

On a bien l'équivalence $F \wedge (G \vee H) \equiv F \wedge G \vee F \wedge H$.

F	G	H	$G \wedge H$	$F \vee G \wedge H$	$F \vee G$	$F \vee H$	$(F \vee G) \wedge (F \vee H)$
V	**V**	**V**	**V**	**V**	**V**	**V**	**V**
V	**V**	**F**	**F**	**V**	**V**	**V**	**V**
V	**F**	**V**	**F**	**V**	**V**	**V**	**V**
V	**F**	**F**	**F**	**V**	**V**	**V**	**V**
F	**V**	**V**	**V**	**V**	**V**	**V**	**V**
F	**V**	**F**	**F**	**F**	**V**	**F**	**F**
F	**F**	**V**	**F**	**F**	**F**	**V**	**F**
F	**F**	**F**	**F**	**F**	**F**	**F**	**F**

On a bien l'équivalence $F \vee G \wedge H \equiv (F \vee G) \wedge (F \vee H)$.

Solution de l'exercice 12
L'idempotence découle de $\mathbf{V} \wedge \mathbf{V} = \mathbf{V}$ et $\mathbf{F} \wedge \mathbf{F} = \mathbf{F}$.

F	G	$F \vee G$	$\neg(F \vee G)$	$F \wedge (F \vee G)$
V	**V**	**V**	**F**	**F**
V	**F**	**V**	**F**	**F**
F	**V**	**V**	**F**	**V**
F	**F**	**F**	**V**	**V**

On a bien l'équivalence $F \wedge (F \vee G) \equiv F$.

Solution de l'exercice 13 - Démonstration du théoreme 6
Pour toute valuation v et toute variable x_i, on a $\widetilde{v}\big(l_v(x_i)\big) = \mathbf{V}$ d'où $\widetilde{v}\big(D_v\big) = \mathbf{V}$.
Pour deux valuations v et v' distinctes il existe i tel que $v'(x_i) = \neg v(x_i)$;
on a alors $\widetilde{v}(\neg x_i) = \neg\widetilde{v'}(\neg x_i)$ d'où $\widetilde{v'}\big(l_v(x_i)\big) = \neg\widetilde{v}\big(l_v(x_i)\big) = \mathbf{F}$.
On a alors $\widetilde{v'}\big(D_v\big) = \mathbf{F}$.
Si v_0 appartient à V_F l'une des clauses de F_N est D_{v_0} donc $\widetilde{v_0}(F_N)$, qui est la disjonction des $\widetilde{v_0}(D_v)$, vaut $\mathbf{V}$. On a aussi $\widetilde{v_0}(F) = \mathbf{V}$ car $v_0 \in V_F$.
Si v_0 n'appartient pas à V_F toutes les clauses de F_N donnent une valeur $\mathbf{F}$ par $\widetilde{v_0}$ donc $\widetilde{v_0}(D_v)$ vaut $\mathbf{F}$. On a aussi $\widetilde{v_0}(F) = \mathbf{F}$ car $v_0 \notin V_F$.
Ainsi F et F_N ont la même valeur pour toute valuation, elles sont donc équivalentes.

Solution de l'exercice 14

```
let implique p q = Ou(Non p, q);;

let equivaut p q = Ou(Et(p, q), Et(Non p, Non q));;
```

Solution de l'exercice 15
$\big[F \Rightarrow G\big] \equiv \big[\neg F \vee G\big] \equiv \big[\neg\neg G \vee \neg F\big] \equiv \big[\neg G \Rightarrow \neg F\big]$
$\big[F \Rightarrow (G \Rightarrow H)\big] \equiv \big[\neg F \vee (G \Rightarrow H)\big] \equiv \big[\neg F \vee (\neg G \vee H)\big] \equiv \big[\neg(F \wedge G) \vee H\big] \equiv \big[F \wedge G \Rightarrow H\big]$

Solution de l'exercice 16 - Loi de Pierce

F	G	$F \Rightarrow G$	$(F \Rightarrow G) \Rightarrow F$	$\big((F \Rightarrow G) \Rightarrow F\big) \Rightarrow F$
V	**V**	**V**	**V**	**V**
V	**F**	**F**	**V**	**V**
F	**V**	**V**	**F**	**V**
F	**F**	**V**	**V**	**V**

Solution de l'exercice 17

p	q	r	$p \Rightarrow q$	$q \Rightarrow r$	F	$p \wedge q$	$(p \vee r)$	G
V	**V**	**V**	**V**	**V**	**V**	**V**	**V**	**V**
V	**V**	**F**	**V**	**F**	**F**	**V**	**F**	**F**
V	**F**	**V**	**F**	**V**	**F**	**V**	**F**	**F**
V	**F**	**F**	**F**	**V**	**F**	**V**	**F**	**F**
F	**V**	**V**	**V**	**V**	**V**	**V**	**V**	**V**
F	**V**	**F**	**V**	**F**	**F**	**V**	**F**	**F**
F	**F**	**V**	**V**	**V**	**V**	**F**	**F**	**V**
F	**F**	**F**	**V**	**V**	**V**	**F**	**F**	**V**

Les deux formules sont équivalentes.

Solution de l'exercice 18

```
let equivaut p q = Ou(Et(p, q), Et(Non p, Non q));;

let ouExclusif p q = Ou(Et(p, Non q), Et(Non p, q));;
```

Solution de l'exercice 19
Les loi de Morgan donnent
$\big[(\neg F \vee G) \wedge (\neg G \vee F)\big] \equiv \big[\neg F \wedge \neg G \vee \neg F \wedge F \vee G \wedge \neg G \vee G \wedge F\big]$.
Or $\neg F \wedge F \equiv \mathbf{F}$ est est absorbé dans une conjonction, de même pour $G \wedge \neg G$ d'où
$\big[(\neg F \vee G) \wedge (\neg G \vee F)\big] \equiv \big[\neg F \wedge \neg G \vee G \wedge F\big]$.

Solution de l'exercice 20
Si $F \equiv G$ alors, pour toute valuation v, $\widetilde{v}(F) = \widetilde{v}(G)$ donc
soit $\widetilde{v}(F) = \widetilde{v}(G) = \mathbf{V}$ d'où $\widetilde{v}(F \wedge G) = \mathbf{V}$
soit $\widetilde{v}(F) = \widetilde{v}(G) = \mathbf{F}$ d'où $\widetilde{v}(\neg F \wedge \neg G) = \mathbf{V}$.
Dans les deux cas on a $\widetilde{v}(F \wedge G \vee \neg F \wedge \neg G) = \mathbf{V}$.
C'est vrai pour toute valuation donc $F \wedge G \vee \neg F \wedge \neg G$ (c'est-à-dire $F \Leftrightarrow G$) est une tautologie.

Inversement si $F \Leftrightarrow G$ est une tautologie alors elle vraie pour toute valuation ce qui signifie que, pour toute valuation v, $\widetilde{v}(F)$ et $\widetilde{v}(G)$ sont tous deux vrais ou tous deux faux, ils sont toujours égaux : F et G sont équivalents.

Solution de l'exercice 21
Si $\widetilde{v}(F) = \mathbf{V}$ alors $\widetilde{v}\big(\mathbf{ite}\,(F, G, H)\big) = \widetilde{v}(G)$, si $\widetilde{v}(F) = \mathbf{F}$ alors $\widetilde{v}\big(\mathbf{ite}\,(F, G, H)\big) = \widetilde{v}(H)$.

Solution de l'exercice 22
$F \wedge G \equiv \mathbf{ite}\,(F, G, F)$, $F \vee G \equiv \mathbf{ite}\,(F, F, G)$

Solution de l'exercice 23
$\neg F \equiv \mathbf{ite}\,(F, \mathbf{0}, \mathbf{1})$
$\mathbf{ite}\,(F, \mathbf{1}, \mathbf{0}) \equiv F$, c'est la traduction de

```
if expr then true else false
```

Solution de l'exercice 24

```
let rec prop2if p =
   match p with
   |Var s       -> VarI s
   |Neg q       -> Ite (prop2if q, Faux, Vrai)
   |Et (q, r)    -> Ite (prop2if q, prop2if r, Faux)
   |Ou (q, r)    -> Ite (prop2if q, Vrai, prop2if r);;
```

Solution de l'exercice 25
$[F|F] \equiv [\neg F \vee \neg F] \equiv [\neg F]$

Solution de l'exercice 26
$[F \vee G] \equiv [\neg F|\neg F] \equiv [(F|F)|(G|G)]$.

Solution de l'exercice 27
Pour pouvoir transformer toute formule logique il faut encore écrire $F \wedge G$ en fonction uniquement du connecteur de Schaeffer.
$[F \wedge G] \equiv [\neg(\neg F \vee \neg G)] \equiv [\neg(F|G)] \equiv [(F|G)|(F|G)]$.

Solution de l'exercice 28

```
let rec conversion exp =
   match exp with
   |Variable s -> VarS s
   |Non f -> let ff = conversion f in Shf(ff, ff)
   |Ou(f, g) -> let ff = conversion f in
                let gg = conversion g in
                Shf(Shf(ff, ff), Shf(gg, gg))
   |Et(f, g) -> let ff = conversion f in
                let gg = conversion g in
                Shf(Shf(ff, gg), Shf(ff, gg));;
```

Solution de l'exercice 29

```
let rec indiceMax p =
  match p with
  |X k -> k
  |Non( q -> indiceMax q
  |Ou (q, r) -> max (indiceMax q) (indiceMax r)
  |Et (q, r) -> max (indiceMax q) (indiceMax r);;
```

Solution de l'exercice 30

```
let incremente t =
  let rec aux_inc pos =
    if pos = Array.length t
    then false
    else if t.(pos)
         then (t.(pos) <- false;  aux_inc (pos+1))
         else (t.(pos) <- true; true)
  in aux_inc 0;;
```

Solution de l'exercice 31

```
let possibles p =
  let n = indiceMax p in
  let t = Array.make n false in
  let sol = ref [] in
  if eval p t then sol := [Array.copy t];
  while (incremente t)
  do if eval p t
     then sol := (Array.copy t)::(!sol) done;
  !sol;;
```

Solution de l'exercice 32

```
let satisfiable p= possibles p <> [];;

let tautologie p = not (satisfiable (Non p));;
```

Solution de l'exercice 33

```
let x1 = X 1;;
let x2 = X 2;;
let x3 = X 3;;

let p1 = implique (Non(Non x1)) x1;;
let p2 = equivaut (implique x1 x2)
                  (implique (Non x2) (Non x1));;
let p3 = implique (Et (implique x1 x2,implique x2 x3))
                  (implique x1 x3);;

tautologie p1;;
tautologie p2;;
tautologie p3;;
```

Solution de l'exercice 34 -
$P_1 \equiv \mathbf{ite}\big(\mathbf{ite}(A, B, \mathbf{F}), A, \mathbf{V}\big)$,
$P_2 \equiv \mathbf{ite}(A, \mathbf{ite}(A, B, \mathbf{F}), \mathbf{V})$,
$P_3 \equiv \mathbf{ite}(\mathbf{ite}(A, \mathbf{ite}(A, B, \mathbf{V}), \mathbf{V}), B, \mathbf{V})$

Solution de l'exercice 35

```
let rec estNormale p =
  match p with
  |Ite (VarI _, p, q) -> (estNormale p) || (estNormale q)
  |Ite (_,  _,  _) -> false
  |_ -> true;;
```

Solution de l'exercice 36
Chacune de ces transformation donne une IF-expression équivalente à l'expression initiale.
On a bien $\phi(P) < 1 + \phi(P) + \phi(Q)$, $\phi(Q) < 1 + \phi(P) + \phi(Q)$ et
$\phi(M)\big(1 + \phi(P)K + \phi(Q)K\big) < \phi(M)\big(1 + \phi(P) + \phi(Q)\big)K$ avec $K = \big(1 + \phi(R) + \phi(S)\big)$.
Si une If-expression n'est pas normale elle contient une composante de la forme $\mathbf{ite}(M, P, Q)$ avec $M = \mathbf{V}$, $M = \mathbf{F}$ ou $M = \mathbf{ite}(M', P', Q')$.
On peut alors appliquer une des transformations en diminuant strictement la valeur de ϕ.
Si les transformations ne permettaient jamais d'arriver à une forme normale on aurait une suite de formules équivalentes M_n et la suite des $\big(\phi(M_n)\big)$ formerait une suite strictement décroissante infinie d'entiers positifs ce qui est impossible.
On parvient donc à une IF-expression normale après un nombre fini de transformations.

Solution de l'exercice 37

```
let rec normalise prop =
  match prop with
  |Ite (VarI s, p, q) -> Ite (VarI s, normalise p, normalise q)
  |Ite (Vrai_ite, p, _) -> normalise p
  |Ite (Faux_ite , _, q) -> normalise q
  |Ite (Ite (m, p, q), r, s)
     -> normalise (Ite (m, Ite (p, r, s), Ite (q, r, s)))
  |_ -> prop;;
```

Solution de l'exercice 38
$P_1 \equiv \mathbf{ite}(A, \mathbf{ite}(B, A, \mathbf{V}), \mathbf{V})$,
$P_2 \equiv \mathbf{ite}(A, \mathbf{ite}(A, B, \mathbf{F}), \mathbf{V})$,
$P_3 \equiv (A, \mathbf{ite}(A, \mathbf{ite}(B, B, \mathbf{V}), B), B)$.

Solution de l'exercice 39

```
let rec defini x v =
  match v with
  |[] -> false
  |(s, _)::vv -> (s = x) || (defini x vv);;
```

Solution de l'exercice 40

```
let rec valeur x v =
  match v with
  |[] -> failwith "non valué"
  |(s,b)::vv ->  if s = x then b else valeur x vv;;
```

Solution de l'exercice 41

- Si $\alpha(s) = \mathbf{V}$ alors une réfutation de Q est une réfutation de P.
- Si $\alpha(s) = \mathbf{F}$ alors une réfutation de R est une réfutation de P.
- si $\alpha(s)$ n'est pas défini alors une réfutation de P est une réfutation de Q si Q n'est pas une α_v-tautologie ou une réfutation de R si R n'est pas une α_f-tautologie .

Solution de l'exercice 42
On applique l'algorithme

```
let rec decisionP valu prop =
  match prop with
  |Vrai -> Tautologie
  |Faux  -> Refutation valu
  |VarI s when defini s valu -> if valeur s valu
                                  then Tautologie
                                  else Refutation valu
  |VarI s -> Refutation ((s, false)::valu)
  |Ite (Var s, p, q) when defini s valu
     -> if valeur s valu
        then decisionP valu p
        else decisionP valu q)
  |Ite (Var s, p, q)
     -> let res = decisionP ((s, true)::valu) p in
        if res  = Tautologie
        then decisionP ((s, false)::valu) q
        else res
  |_ -> failwith "ceci ne devrait pas arriver ";;
```

Solution de l'exercice 43

```
let decisionIF p = decisionP [] p;;
```

Solution de l'exercice 44

```
let decision p = decisionIF (normalise (prop2if p));;
```

Solution de l'exercice 45

- A dit $G \vee D$, B dit $\neg D$.
- C est la propriété $(G \vee D) \wedge \neg D \vee \neg(G \vee D) \wedge \neg\neg D$
- C doit être vrai, sa table de vérité est

G	D	$G \vee D$	$\neg D$	C
V	**V**	**V**	**F**	**F**
V	**F**	**V**	**V**	**F**
F	**V**	**V**	**F**	**V**
F	**F**	**F**	**V**	**F**

C ne peut être vrai que si G est vrai et D est faux, il faut prendre le chemin de gauche et les 2 sphinx ont dit la vérité.

Solution de l'exercice 46

```
let l  = Var "Levogyre";;
let rn = Var "RayuresNoires";;
let ob = Var "OreillesBlanches";;
let f  = Var "vitForet";;
let mm = Var "mangeMulot";;

let d1 = Donc (Neg l, rn);;
let d2 = Ou (ob, Neg rn);;
let d3 = Donc (f, Neg mm);;
let d4 = Et(Donc (mm, l), Donc (l, mm));;
let d5 = Donc (ob, Et (l, f));;
let d6 = Donc (l, ob);;

let p = Neg (multiEt [d1; d2; d3; d4; d5; d6]);;
```

p est une tautologie donc les conditions donnent une contradiction.

Solution de l'exercice 47

```
let p1 = Var "Princesse en 1";;
let p2 = Var "Princesse en 2";;
let p3 = Var "Princesse en 3";;

let e1 = Neg p2;;
let e2 = Neg p2;;
let e3 = Neg p1;;

let o1 = multiEt [p1; e1; Neg p2; Neg p3];;
let o2 = multiEt [p2; e2; Neg p3: Neg p1];;
let o3 = multiEt [p3; e3; Neg p1: Neg p2];;

let test = Neg (multiOu [o1; o2; o3]);;

decision test;;

Refutation ["Princesse en 3", false; "Princesse en 2", false;
            "Princesse en 1", true]
```

Solution de l'exercice 48

```
let x1 = X 1;;
let x2 = X 2;;
let x3 = X 3;;
let x4 = X 4;;
let x5 = X 5;;
let p1 = X 6;;
let p2 = X 7;;
let p3 = X 8;;
let p4 = X 9;;
let p5 = X 10;;

let c1 = implique x1 p1;;
let c2 = implique x2 p2;;
let c3 = implique x3 p3;;
let c4 = implique x4 p4;;
let c5 = implique x5 p5;;

let u = uneSeule [x1; x2; x3; x4; x5];;

let e1 = equivaut p1 (Et (Non x1, x4));;
let e2 = equivaut p2 (multiOu [x1; x3; x5]);;
let e3 = equivaut p3 (Ou (p2, Non p5));;
let e4 = equivaut p4 (Et (Non p4, p1));;
let e5 = equivaut p5 (Non p3);;

let p = multiEt [c1; c2; c3; c4; c5; u; e1; e2; e3; e4; e5];;

possibles jeu1;;
```

Prendre le troisième couloir.

Solution de l'exercice 49

```
let x1 = X 1;;
let x2 = X 2;;
let x3 = X 3;;
let x4 = X 4;;
let x5 = X 5;;
let x6 = X 6;;
let x7 = X 7;;
let x8 = X 8;;
let x9 = X 9;;
let p1 = X 10;;
let p2 = X 11;;
let p3 = X 12;;
let p4 = X 13;;
let p5 = X 14;;
let p6 = X 15;;
let p7 = X 16;;
let p8 = X 17;;
let p9 = X 18;;

let c1 = implique x1 p1;;
let c2 = implique x2 p2;;
let c3 = implique x3 p3;;
let c4 = implique x4 p4;;
let c5 = implique x5 p5;;
let c6 = implique x6 p6;;
let c7 = implique x7 p7;;
let c8 = implique x8 p8;;
let c9 = implique x9 p9;;

let u = uneSeule [x1; x2; x3; x4; x5; x6; x7; x8; x9];;

let e1 = equivaut p1 (multiOu [x1; x3; x5; x7; x9]);;
let e2 = equivaut p2 x2;;
let e3 = equivaut p3 (Et (Ou (p5, Non p7), Non p9));;
let e4 = equivaut p4 (Non p1);;
let e5 = equivaut p5 (Ou (p2, p4));;
let e6 = equivaut p6 (Non p3);;
let e7 = equivaut p7 (Non x1);;
let e8 = equivaut p8 (Et (Non p8, x9));;
let e9 = equivaut p9 (Non p6);;

let jeu3 = multiEt [c1; c2; c3; c4; c5; c6; c7; c8; c9; u; e1; e2;
    e3; e4; e5; e6; e7; e8; e9];;

possibles jeu3;;
```

La septième.