Exercice 1. Nombre de variables
Soit ϕ une formule logique. Soit n le nombre de connecteurs logiques ($\vee$ et $\wedge$) dans ϕ.
Exprimer le nombre de terminaux (variables, T ou F) de ϕ, en fonction de n.
► On peut représenter ϕ par un arbre (voir cours/résumé) où les noeuds internes sont les connecteurs logiques et les feuilles sont les occurrences variables.
S'il n'y a pas de $\neg$, tous les connecteurs logiques sont des binaires donc l'arbre est binaire strict.
D'après la formule vue dans le cours sur les arbres, le nombre de feuilles de cet arbre est égal à 1 + le nombre de noeuds internes.
Donc le nombre d'occurrences de variables dans ϕ est $\boxed{n+1}$.
On peut le vérifier sur un exemple : si $\phi = (x \wedge z) \vee (z \vee y)$ alors ϕ a 3 connecteurs logiques et $4 = 3 + 1$ occurrences de variables.
Remarque : on peut aussi le démontrer par récurrence (la formule sur les arbres est aussi démontrée par récurrence).

Exercice 2. Tautologie et équivalence
Soient ϕ_1, ϕ_2, ϕ_3 des formules. Montrer que les formules suivantes sont des tautologies :

1. $\phi_1 \longrightarrow (\phi_2 \longrightarrow \phi_1)$
2. $(\phi_1 \longrightarrow \phi_2) \vee (\phi_2 \longrightarrow \phi_3)$
3. $(\neg\neg\phi_1) \longrightarrow \phi_1$

Pour chacune des équivalences suivantes, la prouver ou trouver un contre-exemple :

4. $(\phi_1 \longrightarrow \phi_2) \longrightarrow \phi_3 \equiv \phi_1 \longrightarrow (\phi_2 \longrightarrow \phi_3)$
5. $(\phi_1 \wedge \phi_2) \longrightarrow \phi_3 \equiv \phi_1 \longrightarrow (\phi_2 \longrightarrow \phi_3)$

Exercice 3. Énigme gastronomique
Trois personnes (nommée A, B, C) mangent ensemble. On sait que :

- si A prend un dessert, B aussi
- soit B, soit C prennent un dessert, mais pas les deux
- A ou C prend un dessert
- si C prend un dessert, A aussi

Déterminer qui prend un dessert, en utilisant une table de vérité.
► On note a, b, c des variables booléennes indiquant si A, B, C prend un dessert (par exemple, $a = 1 \longleftrightarrow A$ prend un dessert).
Les 4 conditions deviennent :

- $a \longrightarrow b$ (qui est équivalent à $\neg a \vee b$)
- $(b \vee c) \wedge \neg(b \wedge c)$ (qui est équivalent à $(b \wedge \neg c) \vee (\neg b \wedge c)$)
- $a \vee c$
- $c \longrightarrow a$ (qui est équivalent à $\neg c \vee a$)

En écrivant les valeurs de vérités de ces 4 formules :

a	b	c	$\neg a \vee b$	$(b \wedge \neg c) \vee (\neg b \wedge c)$	$a \vee c$	$\neg c \vee a$
0	0	0	1	0	0	1
0	0	1	1	1	1	0
0	1	0	1	1	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	1	0
1	1	0	1	1	1	1
1	1	1	1	0	1	1

(on peut vérifier qu'on a bien $2^3 = 8$ lignes)
On trouve qu'une seule possibilité pour que les 4 conditions soit réunies simultanément : $a = 1$, $b = 1$, $c = 0$ c'est à dire A et B prennent un dessert mais pas C.

Exercice 4. Calcul booléen
Donner des formules équivalentes les plus simples possibles pour les formules suivantes (en utilisant le moins de littéraux possible) :

1. $\phi_1 = c(b + c) + (a + d)\overline{(a\overline{d} + c)}$
   ► $\phi_1 \equiv cb+c+(a+d)((\overline{a}+d)\overline{c}) \equiv c+(a\overline{a}+ad+d\overline{a}+dd)\overline{c} \equiv c + (0 + (a + \overline{a})d + d)\overline{c} \equiv c + (d + d)\overline{c} \equiv c + d\overline{c}$.
2. $\phi_2 = ab + c + \overline{b}\overline{c} + \overline{a}\ \overline{c}$
   ► $\phi_2 \equiv ab+c+(\overline{b}+\overline{a})\overline{c} \equiv ab+c+\overline{ab}\overline{c} \equiv ab+c+\overline{ab + c} \equiv 1$: ϕ_2 est une tautologie.
3. $\phi_3 = \neg(a \wedge b) \wedge (a \vee \neg b) \wedge (a \vee b)$

Exercice 5. Système complet logique
On définit les opérateurs NAND, NOR, XOR par leurs tables de vérité :

x	y	$x\ NAND\ y$	$x\ NOR\ y$	$x\ XOR\ y$
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	0	0	0

On dit qu'un ensemble S d'opérateurs logiques est **complet** si toute formule logique est équivalente à une formule qui n'utilise que des opérateurs dans S.

1. Exprimer NAND, NOR, XOR, à l'aide de $\vee$, $\wedge$, $\neg$.
   ► $x\ NAND\ y \equiv \neg(x \wedge y)$, $x\ NOR\ y \equiv \neg(x \vee y)$, $xXORy \equiv (x \wedge \neg y) \vee (\neg x \wedge y)$ $(\equiv (x \vee y) \wedge (\neg x \vee \neg y))$.
2. Montrer que $\{\wedge, \neg\}$ est complet.
   ► $x \vee y \equiv \neg(\neg x \wedge \neg y)$. Toute formule avec des $\vee$, $\wedge$, $\neg$ peut donc être réécrite avec que des $\wedge$ et $\neg$.
3. Montrer que $\{NAND\}$ est complet. (c'est pour cette raison que le NAND est très utilisé en électronique)
   ► Si $x = 1$ alors $x\ NAND\ x = 0$ et si $x = 0$ alors $x\ NAND\ x = 1$. D'où $\neg x \equiv x\ NAND\ x$.
   Vu la table vérité de $NAND$, $x \wedge y = \neg(x\ NAND\ y) = (x\ NAND\ y)\ NAND\ (x\ NAND\ y)$.
   Comme $\{\wedge, \neg\}$ est complet (d'après la question précédente), $\{NAND\}$ l'est aussi.

4. Montrer que $\{NOR\}$ est complet.
   ▶ $\neg x\ NOR\ \neg y = 1$ ssi $x = 1$ et $y = 1$. D'où $x\ NAND\ y = \neg(\neg x\ NOR\ \neg y)$.
   De plus $\neg x = x\ NOR\ x$. Donc $x\ NAND\ y$ peut s'écrire avec que des NOR. Comme $NAND$ est complet, NOR est complet.

5. Montrer que $\{XOR\}$ n'est pas complet.
   ▶ Nous allons trouver une propriété (qu'on appelle parfois un invariant) vérifiée par une formule n'utilisant que des XOR mais qui n'est pas vraie pour les formules en général.
   Soit ϕ une formule n'utilisant que des variables et des XOR.
   Soit d l'évaluation donnant la valeur 0 à toutes les variables de ϕ.
   On peut démontrer par récurrence sur la taille de ϕ que $[\![\phi]\!]_d = 0$:

   - si ϕ est une variable (cas de base) alors $[\![\phi]\!]_d = 0$ par définition de d
   - si $\phi = \phi_1\ XOR\ \phi_2$ alors, par hypothèse de récurrence (car ϕ_1 et ϕ_2 sont de tailles plus petites que ϕ et ne contiennent que des XOR), $[\![\phi_1]\!]_d = 0$ et $[\![\phi_2]\!]_d = 0$ donc $[\![\phi]\!]_d = 0$ (d'après la table de vérité de XOR).

   On ne peut donc pas obtenir la formule $\neg x$ (par exemple) avec que des XOR puisque cette formule est vraie quand $x = 0$.

Exercice 6. Forme normale conjonctive/disjonctive
On rappelle qu'une forme normale conjonctive (FNC) est une conjonction ($\wedge$) de disjonctions ($\vee$) de littéraux (chaque littéral étant une variable ou sa négation). Par exemple, $(a \vee b) \wedge (b \vee c \vee d) \wedge c$ est une FNC.
On rappelle qu'une forme normale disjonctive (FND) est une disjonction ($\vee$) de conjonctions ($\wedge$) de littéraux (chaque littéral étant une variable ou sa négation).

1. Montrer par induction/récurrence que toute formule logique (construite avec $\wedge$, $\vee$, $\neg$) est équivalente à une FNC ainsi qu'à une FND.
   ▶ On définit la taille d'une formule comme son nombre de connecteurs logiques ($\wedge$, $\neg$, $\vee$) et de variables.
   Montrons $P(n)$: « si ϕ est une formule de taille n alors ϕ est équivalente à une FND ainsi qu'à une FNC ».
   $P(1)$ est vraie car une formule de taille 1 est une variable qui est à la fois une FND et une FNC.
   Soit $n \in \mathbb{N}^*$. Supposons $P(k)$ pour tout $k \leq n$.
   Soit ϕ une formule de taille $n+1$:

   - Si ϕ est la négation d'une formule, i.e $\phi = \neg\psi$. ψ est de taille n donc d'après $P(n)$, $\psi \equiv \psi'$ et $\psi \equiv \psi''$, où ψ' est une FNC et ψ'' une FND.
     Alors, en appliquant les lois de De Morgan sur $\neg\psi'$ on obtient une FND et en appliquant les lois de De Morgan sur $\neg\psi''$ on obtient une FNC.
   - Si $\phi = \psi \wedge \psi'$: soient (par hypothèse de récurrence) f et f' des FNC équivalentes à ψ et ψ', respectivement.
     Clairement, $f \wedge f'$ est une FNC équivalente à ϕ.
     Soient $g = c_1 \vee c_2 \vee ... \vee c_k$ et $g' = c'_1 \vee c'_2 \vee ... \vee c'_k$ des FND équivalentes à ψ et ψ', respectivement.
     Alors $g \wedge g' = (c_1 \vee c_2 \vee ... \vee c_k) \wedge (c'_1 \vee c'_2 \vee ... \vee c'_k) = \bigvee_{i,j}(c_i \wedge c'_j)$ est une FND.
   - Démonstration similaire si $\phi = \psi \vee \psi'$...

2. Donner une FNC et une FND équivalente à $\neg(x \longrightarrow (\neg y \wedge z)) \vee (z \longrightarrow y)$.
   ▶ $\neg(x \longrightarrow (\neg y \wedge z)) \vee (z \longrightarrow y) \equiv \neg(\neg x \vee (\neg y \wedge z)) \vee (\neg z \vee y) \equiv (x \wedge (y \vee \neg z)) \vee \neg z \vee y$.
   Pour trouver une FND, on peut utiliser la distributivité de $\wedge$ dans $x \wedge (y \vee \neg z$: $(x \wedge (y \vee \neg z)) \vee \neg z \vee y = \boxed{(x \wedge y) \vee (x \wedge \neg z) \neg z \vee y}$.
   Pour trouver une FNC, on peut utiliser la distributivité de $\vee$ dans $(x \wedge (y \vee \neg z)) \vee (\neg z \vee y)$:
   $(x \wedge (y \vee \neg z)) \vee (\neg z \vee y) = (x \vee (\neg z \vee y)) \wedge ((y \vee \neg z) \vee (\neg z \vee y)) \equiv \boxed{(x \vee (\neg z \vee y)) \wedge (y \vee \neg z)}$.

3. Écrire une fonction `fnc : 'a formula -> 'a formula` mettant une forme en FNC.

Exercice 7. Extrait Mines-Pont 2010

1. En construisant la table de vérité de f_1, on trouve qu'elle est satisfiable pour $(x, y, z) \in \{(1,0,0), (0,1,1), (0,0,1)\}$

x	y	z	f_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

2. Soit f_2 la conjonction de toutes les différentes clauses de taille 3 (au nombre de $2^3 = 8$) sur x, y, z, c'est à dire :

   $f_2 = (x \vee y \vee z) \wedge (x \vee y \vee \overline{z}) \wedge (x \vee \overline{y} \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee y \vee z) \wedge (\overline{x} \vee y \vee \overline{z}) /$

   Alors chaque valuation de x, y, z satisfait toutes les clauses de f_2 sauf une seule : par exemple, $(x, y, z) = (1, 0, 1)$ satisfait toues ses clauses sauf $\overline{x} \vee y \vee \overline{z}$.
   On en déduit que f_2 n'est pas satisfiable et que $\max(f_2) = 7$.

3. Une clause C est satisfaite par toute valuation sauf celles qui donne un triplet de valeurs booléennes particulier aux trois variables qui apparaissent dans C (les 2^{n-3} autres variables ont alors une valeur quelconque) :

$$\sum_{val \in V} \varphi(C, val) = 2^n - 2^{n-3} = \frac{7}{8} 2^n$$

4.

$$\sum_{C \text{ clause de } f} \sum_{val \in V} \varphi(C, val) = \frac{7m}{8} 2^n = \sum_{val \in V} \sum_{C \text{ clause de } f} \varphi(C, va$$

Donc $\max(f) \geq \frac{7}{8}m = m - \frac{m}{8}$ et, comme $\max(f)$ est un entier : $\max(f) \geq \lceil \frac{7}{8}m \rceil = m - \lfloor \frac{m}{8} \rfloor$.

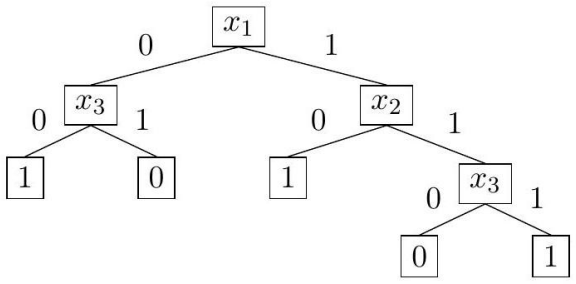
5. Supposons f non satisfiable. Alors $\max(f) < m$ donc, d'après la question précédente, $m - \lfloor \frac{m}{8} \rfloor < m$ donc $\lfloor \frac{m}{8} \rfloor \geq 1$ et $m \geq 8$.
Une formule non satisfiable contient donc au moins 8 clauses.
D'après la question 2, 8 est donc le nombre minimum de clauses d'une instance de 3-SAT non satisfiable.

Exercice 8. Arbre de décision (Oral ENS info)

On considère un ensemble de variables propositionnelles $\mathcal{X} = \{x_1, \ldots, x_n\}$ muni de l'ordre total où $x_i < x_j$ si et seulement si $i < j$. Un arbre de décision sur $\mathcal{X}$ est un arbre binaire dont les feuilles sont étiquetées par 0 ou par 1 , et dont les nœuds internes sont étiquetés par une variable de $\mathcal{X}$ et ont deux enfants appelés enfant 0 et enfant 1 . On impose que si un nœud interne n étiqueté par x_i a un descendant n' qui est un nœud interne étiqueté par x_j alors $x_i < x_j$.

Un arbre de décision T sur $\mathcal{X}$ décrit une fonction booléenne Φ_T qui à toute valuation $\nu : \mathcal{X} \to \{0,1\}$ associe une valeur de vérité calculée comme suit : si T consiste exclusivement d'une feuille étiquetée $b \in \{0,1\}$ alors la fonction Φ_T s'évalue à b quelle que soit ν. Sinon, on considère le nœud racine n de T et la variable x_i qui l'étiquette, on regarde la valeur $\nu(x_i) \in \{0,1\}$ que ν donne à x_i, et le résultat de l'évaluation de Φ_T sous ν est celui de l'évaluation de $\Phi_{T'}$ sous ν, où T' est le sous-arbre de T enraciné en l'enfant b de n.

1. On considère l'arbre de décision T_0 suivant et la fonction Φ_{T_0} qu'il définit. Évaluer cette fonction pour la valuation donnant à x_1, x_2, x_3 respectivement les valeurs $0, 1, 0$. Donner un exemple de valuation sous laquelle cette formule s'évalue en 0 .



2. On considère la formule de la logique propositionnelle $(x_1 \wedge x_2) \vee \neg(x_1 \wedge \neg x_3)$. Construire un arbre de décision sur les variables $x_1 < x_2 < x_3$ qui représente la même fonction.

3. Quels arbres de décision représentent des tautologies? des fonctions satisfiables?

4. Étant donné un arbre de décision représentant une fonction Φ, expliquer comment on pourrait construire un arbre de décision représentant sa négation $\neg\Phi$.

On définit un type pour les formules logiques :

```
type formule =
    | Var of int
    | Et of formule * formule
    | Ou of formule * formule
    | Non of formule
```

On définit aussi un type pour les arbres de décision (où le sous-arbre gauche est celui de valeur 0 et le sous-arbre droit celui de valeur 1) :

```
type arbre = F of int | N of int * arbre * arbre
```

5. Écrire une fonction `arb_to_for : arbre -> formule` permettant de convertir un arbre de décision en formule logique. Analyser sa complexité en temps et en espace.

6. Écrire une fonction `and : arbre -> arbre -> arbre` telle que, si `a1` et `a2` sont des arbres représentant des formules `f1` et `f2`, `and a1 a2` renvoie un arbre représentant la formule `Et (f1, f2)`.
Quelle est sa complexité en temps et la taille de l'arbre obtenu ?

7. Étant donné un arbre de décision et la séquence de variables $x_1, \ldots, x_n$, donner le pseudocode d'un algorithme qui calcule combien de valuations satisfont la fonction booléenne qu'il capture. Quelle est sa complexité en temps?

8. Peut-on efficacement récrire une formule quelconque de la logique propositionnelle en arbre de décision?