Dans ce problème, on se propose de définir et de démontrer une méthode permettant de savoir si une formule de la logique des propositions est une tautologie.
Soit $\mathscr{B} = \{0, 1\}$ l'ensemble contenant les entiers 0 et 1. On définit les opérations binaires $+ : \mathscr{B}^2 \to \mathscr{B}$ et $\bullet : \mathscr{B}^2 \to \mathscr{B}$ par $x + y = \max(x, y)$ et $x \bullet y = \min(x, y)$. On définit l'opération unaire $\_ : \mathscr{B} \to \mathscr{B}$ par $\overline{0} = 1$ et $\overline{1} = 0$. On admettra les propriétés élémentaires de ces trois opérations : commutativité, associativité, éléments neutres et absorbants, diverses distributivités. On notera en particulier que pour $(x, y) \in \mathscr{B}^2$, $\overline{x \bullet y} = \overline{x} + \overline{y}$ et $\overline{x + y} = \overline{x} \bullet \overline{y}$
Soit $\mathscr{A}$ un ensemble dénombrable de variables propositionnelles. On note $\mathscr{F}$ l'ensemble des *formules bien formées* construites à partir des variables de $\mathscr{A}$, des deux constantes **v** pour *vrai* et **f** pour *faux*, du connecteur unaire de négation noté $\neg$, des connecteurs binaires de disjonction noté $\vee$, de conjonction noté $\wedge$ et d'implication noté $\Rightarrow$ et, enfin, des parenthèses.
Les variables de $\mathscr{A}$ et les constantes **v** et **f** sont appelées des *formules atomiques*.
Une *valuation* est une application $\lambda : \mathscr{A} \to \mathscr{B}$. Une valuation se prolonge en une *interprétation* $I_\lambda : \mathscr{F} \to \mathscr{B}$ définie par :

- si $x \in \mathscr{A}$ alors $I_\lambda(x) = \lambda(x)$;
- $I_\lambda(\mathbf{v}) = 1$ et $I_\lambda(\mathbf{f}) = 0$;
- si $F, G \in \mathscr{F}$ alors :
  - $I_\lambda(F \vee G) = I_\lambda(F) + I_\lambda(G)$;
  - $I_\lambda(F \wedge G) = I_\lambda(F) \bullet I_\lambda(G)$;
  - $I_\lambda(F \Rightarrow G) = \overline{I_\lambda(F)} + I_\lambda(G)$;
  - $I_\lambda(\neg F) = \overline{I_\lambda(F)}$.

Soit F et G deux formules, si pour toute valuation $\lambda : \mathscr{A} \to \mathscr{B}$ on a $I_\lambda(F) = I_\lambda(G)$ alors on dit que F et G sont *logiquement équivalentes* et on le note $F \equiv G$. Si $F \equiv \mathbf{v}$ alors F est appelée une *tautologie*. Si $F \equiv \mathbf{f}$ alors F est appelée une *contradiction*.
Une *formule conjonctive* est soit une formule atomique soit une conjonction de formules $F_1 \wedge F_2 \wedge \cdots \wedge F_n$ avec $n \geq 2$. Une *formule conjonctive élémentaire* est soit une formule atomique soit une conjonction de formules atomiques.
Une *formule disjonctive* est soit une formule atomique soit une disjonction de formules $F_1 \vee F_2 \vee \cdots \vee F_n$ avec $n \geq 2$. Une *formule disjonctive élémentaire* est soit une formule atomique soit une disjonction de formules atomiques.
Une *implication élémentaire* est une implication $F \Rightarrow G$ où F est une formule conjonctive élémentaire et G une formule disjonctive élémentaire.
Afin de vérifier si une formule est une tautologie, on se propose de se ramener à la vérification qu'une ou plusieurs implications élémentaires sont des tautologies.

Le problème est composée de deux parties. La première partie consiste à mettre en place un algorithme permettant de vérifier si une formule est une tautologie, en se ramenant à la vérification qu'une ou plusieurs implications élémentaires sont des tautologies. La deuxième partie consiste à implémenter cet algorithme. Cependant, les deux parties sont en grande partie indépendantes et il n'est pas nécessaire d'avoir traité la première partie pour aborder la seconde (à part pour les questions 16 et 17) .

– Partie I –

Dans ce problème on utilisera $x_1, x_2, \ldots, x_n, \ldots$ pour désigner des variables propositionnelles tandis que les lettres majuscules $A, B, \ldots, F, G, \ldots$ désigneront des formules quelconques.
Dans toutes les preuves ci-dessous, on pourra au choix écrire des tables de vérité ou réaliser un calcul algébrique à l'aide des opérations $+$, $\bullet$ et -.
Par exemple pour montrer que si $A \equiv B$ alors $(A \Rightarrow G) \equiv (B \Rightarrow G)$ on peut

— Réaliser la table de vérité en donnant à A et B la même valeur

A	B	G	$A \Rightarrow G$	$B \Rightarrow G$
0	0	0	1	1
0	0	1	1	1
1	1	0	0	0
1	1	1	1	1

— Réaliser un calcul algébrique. On suppose $A \equiv B$. Pour toute valuation λ, on a donc $I_\lambda(A) = I_\lambda(B)$. De ce fait,

$$I_\lambda(A \Rightarrow G) = \overline{I_\lambda(A)} + I_\lambda(G) = \overline{I_\lambda(B)} + I_\lambda(G) = I_\lambda(A \Rightarrow G)$$

Donc $(A \Rightarrow G) \equiv (B \Rightarrow G)$.

1. Démontrer que si $A \equiv B$ alors $(F \Rightarrow A) \equiv (F \Rightarrow B)$
2. (a) À quelles conditions une implication élémentaire $F \Rightarrow G$ est-elle une tautologie ? *On n'oubliera pas de considérer les cas où les constantes* $\boldsymbol{v}$ *et* $\boldsymbol{f}$ *apparaissent dans* F *ou dans* G.
   (b) Les implications élémentaires suivantes sont-elles des tautologies ?
      (a) $(x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5 \vee x_2)$
      (b) $(x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5 \vee x_6)$
      (c) $(\mathbf{f} \wedge x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5)$
      (d) $(x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5 \vee \mathbf{v})$
3. Démontrer que

$$((F \wedge \neg A) \Rightarrow G) \equiv (F \Rightarrow (G \vee A)) \qquad (\mathrm{G}_1)$$

   et en déduire que

$$((\neg A) \Rightarrow G) \equiv (\mathbf{v} \Rightarrow (G \vee A)) \qquad (\mathrm{G}_2)$$

4. Démontrer que $(F \wedge (A \vee B)) \Rightarrow G$ est une tautologie si et seulement si $(F \wedge A) \Rightarrow G$ et $(F \wedge B) \Rightarrow G$ sont des tautologies. Nous appellerons cette propriété (G_3)
   En déduire que pour que $(A \vee B) \Rightarrow G$ soit une tautologie il faut et il suffit que $A \Rightarrow G$ et $B \Rightarrow G$ soient des tautologies. Nous appellerons cette propriété (G_4)

On admet que l'on peut montrer de même que $(F \wedge (A \Rightarrow B)) \Rightarrow G)$ est une tautologie si et seulement si $(F \wedge B) \Rightarrow G$ et $F \Rightarrow (G \vee A)$ sont des tautologies (nous appellerons cette propriété (G_5)) puis que pour que $(A \Rightarrow B) \Rightarrow G$ soit une tautologie il faut et il suffit que $B \Rightarrow G$ et $\mathbf{v} \Rightarrow (G \vee A)$ soient des tautologies (nous appellerons cette propriété (G_6)).

5. Montrer

$$(F \Rightarrow (G \vee \neg A)) \equiv ((F \wedge A) \Rightarrow G) \qquad (\mathrm{D}_1)$$

   et

$$(F \Rightarrow \neg A) \equiv ((F \wedge A) \Rightarrow \mathbf{f}) \qquad (\mathrm{D}_2)$$

6. Démontrer que $F \Rightarrow (G \vee (A \wedge B))$ est une tautologie si et seulement si $F \Rightarrow (G \vee A)$ et $F \Rightarrow (G \vee B)$ sont des tautologies. Nous appellerons cette propriété (D_3)
   En déduire que pour que $F \Rightarrow (A \wedge B)$ soit une tautologie il faut et il suffit que $F \Rightarrow A$ et $F \Rightarrow B$ soient des tautologies. Nous appellerons cette propriété (D_4)

On admet que l'on peut montrer de même

$$(F \Rightarrow (G \vee (A \Rightarrow B))) \equiv ((F \wedge A) \Rightarrow (G \vee B)) \tag{D_5}$$

et

$$(F \Rightarrow (A \Rightarrow B)) \equiv ((F \wedge A) \Rightarrow B) \tag{D_6}$$

7. Déduire des propriétés (G_i) et (D_i) pour i compris entre 1 et 6 un algorithme prenant en arguments deux formules F et G et qui permet de savoir si une implication $F \Rightarrow G$ est une tautologie en se ramenant à la vérification qu'une ou plusieurs implications élémentaires sont des tautologies. L'algorithme sera exprimé en langage naturel et on ne cherchera pas à prouver sa terminaison.

8. (a) Expliquer comment utiliser l'algorithme de la question 7 pour savoir si une formule F est une tautologie.

   (b) Expliquer comment utiliser l'algorithme de la question 7 pour savoir si une formule F est une contradiction.

9. Une *c-mesure* d'une formule F notée $\mu_c(F)$ est définie comme étant le nombre d'occurrences de connecteurs qu'elle contient. Ainsi $\mu_c((x_1 \wedge x_2) \vee (\mathbf{v} \wedge (\neg x_2))) = 4$ puisque la formule contient deux fois le connecteur $\wedge$, une fois le connecteur $\vee$ et une fois le connecteur $\neg$.

   La *g-mesure* ou *mesure à gauche* d'une formule F, notée $\mu_g(F)$ est définie par

$$\mu_g(F) = \begin{cases} \sum_{i=1}^{n} \mu_g(F_i) & \text{si } F = F_1 \wedge F_2 \wedge \cdots \wedge F_n \text{ où } n \geq 2 \\ \mu_c(F) & \text{sinon} \end{cases}$$

   La *d-mesure* ou *mesure à droite* d'une formule F, notée $\mu_d(F)$ est définie par

$$\mu_d(F) = \begin{cases} \sum_{i=1}^{n} \mu_d(F_i) & \text{si } F = F_1 \vee F_2 \vee \cdots \vee F_n \text{ où } n \geq 2 \\ \mu_c(F) & \text{sinon} \end{cases}$$

   La *i-mesure* d'une implication $F \Rightarrow G$ notée $\mu_i(F \Rightarrow G)$ est définie comme étant la somme de $\mu_g(F)$ et $\mu_d(G)$.

   (a) Montrer que la i-mesure d'une implication élémentaire est nulle.

   (b) Démontrer que $\mu_g(F \wedge G) = \mu_g(F) + \mu_g(G)$ et que $\mu_c(F) \geq \mu_g(F)$.

   (c) Démontrer que $\mu_d(F \vee G) = \mu_d(F) + \mu_d(G)$ et que $\mu_c(F) \geq \mu_d(F)$.

   (d) Expliquer comment démontrer la terminaison de l'algorithme de la question 7 en vous servant de la i-mesure. La rédaction complète de la démonstration étant longue, nous ne demandons pas une preuve exhaustive mais juste le principe.

10. Démontrer que la formule $(x_1 \Rightarrow (x_2 \Rightarrow x_3)) \Rightarrow ((x_1 \Rightarrow x_2) \Rightarrow (x_1 \Rightarrow x_3))$ est une tautologie en utilisant l'algorithme de l'énoncé.

– Partie II –

On se donne une constante : `let vmax = 127;;` On suppose que l'ensemble $\mathscr{A}$ des variables propositionnelles est composé des variables x_i pour i entier compris entre `1` et `vmax`.
On utilisera le type `formule` défini par

```
type formule =  | Vrai | Faux
  | Var of int
  | Et of formule * formule
  | Ou of formule * formule
  | Implique of formule * formule
  | Non of formule;;
```

On devra utiliser des listes de formules. On considère donc aussi le type `listeFormules` défini par
`type listeFormules = formule list;;`

11. (a) Écrire une fonction `estVide : listeFormules -> bool` telle que `estVide lf` renvoie un booléen qui vaut `true` si `lf` est vide et `false` sinon.
    (b) Écrire une fonction `ajoute : formule -> listeFormules -> listeFormules` telle que `ajoute f lf` renvoie la liste obtenue en ajoutant à la liste `lf` la formule `f`. Si la formule `f` était déjà dans la liste on ne l'ajoute pas.
    (c) Écrire une fonction `uneFormule : listeFormules -> formule` telle que `uneFormule lf` renvoie une formule de la liste. On supposera que la liste passée en paramètre n'est pas vide.
    (d) Écrire une fonction `reste : listeFormules -> listeFormules` telle que `reste lf` renvoie la liste des formules obtenue en enlevant la formule qui est renvoyée par la fonction `uneFormule`.

On considère maintenant le type `ensembleFormules` défini par
`type ensembleFormules = { v : bool ; f : bool ; tbl : bool array ; lf : listeFormules}`
Ce type décrit des ensembles de formules. Précisément, l'élément `ef` de type `ensembleFormules` décrit un ensemble de formules tel que :

- `ef.v` vaut `true` si et seulement si la constante **v** appartient à l'ensemble
- `ef.f` vaut `true` si et seulement si la constante **f** appartient à l'ensemble
- `ef.tbl` est un tableau permettant de savoir quelles sont les formules atomiques de la forme x_i qui appartiennent à l'ensemble. Le tableau aura `vmax` cases et `ef.tbl.(i)` vaut `true` si et seulement si x_i appartient à l'ensemble.
- `ef.lf` est la liste des formules non atomiques de l'ensemble.

12. (a) Écrire une fonction `ensembleVide : unit -> ensembleFormules` qui renvoie un ensemble de formules vides.
    (b) Écrire une fonction `queDesVariables : ensembleFormules -> bool` qui renvoie un booléen selon que l'ensemble considéré ne contient que des formules atomiques ou non.
13. Écrire une fonction `intersection (ensembleFormules * ensembleFormules) -> bool` telle que `intersection e1 e2` renvoie `true` si et seulement s'il existe une formule atomique de la forme x_i présente dans les deux ensembles.
14. Écrire une fonction `ajouteEns : formule -> ensembleFormules -> ensembleFormules` telle que `ajouteEns f e` renvoie l'ensemble obtenu en ajoutant à l'ensemble `e` la formule `f`.
15. Écrire une fonction `formuleSuivante : ensembleFormules -> formule * ensembleFormules` telle que, si on suppose que `e` ne contient pas que des formules atomiques, `formuleSuivante e` renvoie le couple `(f,r)` où `f` est une formule non atomique de l'ensemble et `r` l'ensemble obtenu en retirant cette formule.

Pour résoudre les deux dernières questions, on pourra s'aider de ce qui a été fait dans la première partie du problème.

16. Écrire une fonction `verifie : ensembleFormules * ensembleFormules -> bool` telle que si on note $(F_1, \ldots, F_n)$ les formules d'un ensemble `e1` et $(G_1, \ldots, G_p)$ celles d'un ensemble `e2`, la fonction `verifie e1 e2` renvoie un booléen qui vaut `true` si et seulement si $(F_1 \wedge F_2 \wedge \cdots \wedge F_n) \Rightarrow (G_1 \vee G_2 \vee \cdots \vee G_p)$ est une tautologie.
17. Écrire une fonction `tautologie : formule -> bool` qui renvoie `true` si et seulement si la formule passée est une tautologie.