ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DES TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE
(Filière T.S.I.)

CONCOURS D'ADMISSION 2000

ÉPREUVE D'INFORMATIQUE

Filière MP
(Durée de l'épreuve : 3 heures)

Sujet mis à la disposition des concours ENSAE (Statistique), ENSTIM, INT et TPE-EIVP.

Les candidats et les candidates sont priés de mentionner de façon apparente sur la première page de la copie :
« ***INFORMATIQUE - Filière MP*** »

RECOMMANDATIONS AUX CANDIDATS ET CANDIDATES

L'énoncé de cette épreuve, y compris cette page de garde, comporte 11 pages.
Si, au cours de l'épreuve, un candidat ou une candidate repère ce qui lui semble être une erreur d'énoncé, il ou elle le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il ou elle a décidé de prendre.
Tout résultat fourni dans l'énoncé peut être utilisé pour les questions ultérieures même s'il n'a pas été démontré. Il ne faut pas hésiter à formuler les commentaires qui semblent pertinents même lorsque l'énoncé ne le demande pas explicitement.
L'utilisation d'une calculatrice est inutile et interdite.

COMPOSITION DE L'ÉPREUVE

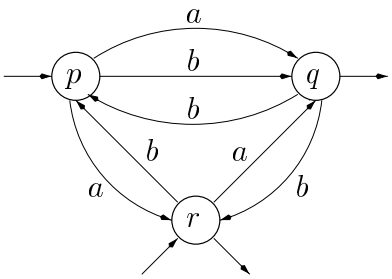
L'épreuve comprend un exercice et deux problèmes.

- **Exercice :** l'exercice de théorie des automates, n° 1 page 2, à résoudre en 30 minutes environ. Cet exercice est indépendant du reste de l'épreuve.
- **Premier problème :** le problème de logique, n° 2 page 2, à résoudre en 75 minutes environ. **Attention!** Ce problème servira de base au deuxième problème. La résolution totale du premier problème n'est pas nécessaire à la résolution du deuxième problème, seule la compréhension de l'énoncé est nécessaire.
- **Deuxième problème :** un problème de programmation au choix parmi deux, à résoudre en 75 minutes environ. **Attention !** Pour le deuxième problème, un seul choix est possible. Le candidat ou la candidate précisera son choix sur sa copie et seul le problème correspondant sera corrigé. Les deux choix possibles sont :
  - le problème de programmation en langage Pascal, n° 3 page 5, pour les candidats et les candidates désirant choisir ce langage ;
  - le problème de programmation en langage Caml, n° 4 page 8, pour les candidats et les candidates désirant choisir ce langage.

1 Exercice sur les automates — 30 mn environ

Il est demandé aux candidats et candidates une grande rigueur dans la rédaction. Si la première question est l'application d'une méthode ne demandant aucune justification, toutes les affirmations dans les solutions des autres questions doivent être rigoureusement justifiées.

L'alphabet $A = \{a,b\}$ est fixé. On considère l'automate $\mathcal{A}$ représenté ainsi :



L'ensemble des états de l'automate $\mathcal{A}$ est $\{p,q,r\}$. Les états initiaux p et r sont marqués par une flèche entrante et sans origine. Les états terminaux q et r sont marqués par une flèche sortante et sans destination. On note A^* l'ensemble des mots sur A.

1. Déterminiser l'automate $\mathcal{A}$, i.e. calculer un automate déterministe équivalent à $\mathcal{A}$.
2. Montrer que $L(\mathcal{A})$, le langage des mots acceptés par $\mathcal{A}$, est formé des mots de A^* qui ne contiennent pas le facteur aaa.
3. Montrer que le langage associé à l'expression rationnelle $b^*(a\,a^*b^*b)^*a^*$ est l'ensemble de tous les mots sur $\{a,b\}$, c'est-à-dire A^*.
4. En déduire une expression rationnelle dont le langage associé est $L(\mathcal{A})$.

FIN DU PROBLÈME SUR LES AUTOMATES

2 Problème de logique — 75 mn environ

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.

Rappels et notations

Soit $\mathcal{B} = \{0,1\}$ l'ensemble contenant les entiers 0 et 1. On définit les opérations binaires $+ : \mathcal{B}^2 \longrightarrow \mathcal{B}$ et $\cdot : \mathcal{B}^2 \longrightarrow \mathcal{B}$ par $x + y = max(x,y)$ et $x \cdot y = min(x,y)$ où $max(x,y)$ et $min(x,y)$ sont respectivement le plus grand et le plus petit des deux entiers x et y. On définit l'opération unaire $^- : \mathcal{B} \longrightarrow \mathcal{B}$ 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, etc.

Soit $\mathcal{A}$ un ensemble dénombrable de variables propositionnelles. On note $\mathcal{F}$ l'ensemble des *formules bien formées* construites à partir des variables de $\mathcal{A}$, de deux constantes notées **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 $\mathcal{A}$ et les constantes $\mathbf{v}$ et $\mathbf{f}$ sont appelées des *formules atomiques*.

Une *valuation* est une application $i : \mathcal{A} \longrightarrow \mathcal{B}$. Une valuation se prolonge en une *interprétation* $I_i : \mathcal{F} \longrightarrow \mathcal{B}$ définie par :

- si $x \in \mathcal{A}$ alors $I_i(x) = i(x)$;
- $I_i(\mathbf{v}) = 1$ et $I_i(\mathbf{f}) = 0$;
- si F,$G \in \mathcal{F}$ alors :
  - $I_i(F \vee G) = I_i(F) + I_i(G)$;
  - $I_i(F \wedge G) = I_i(F) \cdot I_i(G)$;
  - $I_i(F \Rightarrow G) = \overline{I_i(F)} + I_i(G)$;
  - $I_i(\neg F) = \overline{I_i(F)}$.

Soit F et G deux formules, si pour toute valuation $i : \mathcal{A} \longrightarrow \mathcal{B}$ on a $I_i(F) = I_i(G)$ 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 $n \geqslant 2$. 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 $n \geqslant 2$. 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 est 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.

Problème

Dans ce problème, on utilisera x_1,x_2,…,x_n,… pour désigner des variables propositionnelles tandis que les lettres majuscules A,B,…,F,G,… désigneront des formules quelconques.

1. Démontrer que pour que $F \Rightarrow G$ soit une tautologie, il faut et il suffit que pour toute valuation i on ait $I_i(G) = 1$ lorsque $I_i(F) = 1$.
2. À quelles conditions une implication élémentaire est-elle une tautologie ? Les implications élémentaires suivantes sont-elles des tautologies ?
   (a) $(x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5 \vee x_2)$ (c) $(\mathbf{f} \wedge x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5)$
   (b) $(x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5 \vee x_6)$ (d) $(x_1 \wedge x_2 \wedge x_3) \Rightarrow (x_4 \vee x_5 \vee \mathbf{v})$
3. Démontrer que si $A \equiv B$ alors $A \Rightarrow G \equiv B \Rightarrow G$.
4. Démontrer que si $A \equiv B$ alors $F \Rightarrow A \equiv F \Rightarrow B$.
5. Démontrer $(A \wedge \mathbf{v}) \Rightarrow B \equiv A \Rightarrow B$.
6. Démontrer $A \Rightarrow (B \vee \mathbf{f}) \equiv A \Rightarrow B$.
7. Démontrer $F \Rightarrow (G \vee \neg A) \equiv (F \wedge A) \Rightarrow G$ et $F \Rightarrow \neg A \equiv (F \wedge A) \Rightarrow \mathbf{f}$.
8. On a $\neg(\neg A) \equiv A$, en déduire $(F \wedge \neg A) \Rightarrow G \equiv F \Rightarrow (G \vee A)$ et $(\neg A) \Rightarrow G \equiv \mathbf{v} \Rightarrow (G \vee A)$.

9. 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. Puis démontrer 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.
10. On a $\neg(A \wedge B) \equiv (\neg A) \vee (\neg B)$, 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. Puis démontrer 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.
11. On a $A \Rightarrow B \equiv (\neg A) \vee B$, démontrer $F \Rightarrow (G \vee (A \Rightarrow B)) \equiv (F \wedge A) \Rightarrow (G \vee B)$. Puis démontrer $F \Rightarrow (A \Rightarrow B) \equiv (F \wedge A) \Rightarrow B$.
12. Démontrer 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. Puis démontrer 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.
13. Déduire des questions précédentes 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.
14. Expliquer comment utiliser l'algorithme de la question 13 pour savoir si une formule F est une tautologie.
15. Expliquer comment utiliser l'algorithme de la question 13 pour savoir si une formule F est une contradiction.
16. La *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 occurrences du connecteur de conjonction, une occurrence du connecteur de disjonction et une occurrence du connecteur de négation.

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

    $$\mu_g(F) = \begin{cases} \sum_{i=1}^{n} \mu_c(F_i) & \text{si } F \text{ est une formule conjonctive } F_1 \wedge F_2 \wedge \cdots \wedge F_n \text{ avec } n \geqslant 2 \\ \mu_c(F) & \text{sinon} \end{cases}$$

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

    $$\mu_d(F) = \begin{cases} \sum_{i=1}^{n} \mu_c(F_i) & \text{si } F \text{ est une formule disjonctive } F_1 \vee F_2 \vee \cdots \vee F_n \text{ avec } n \geqslant 2 \\ \mu_c(F) & \text{sinon} \end{cases}$$

    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 de $\mu_d(G)$.

    (a) Démontrer 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 $\mu_c(F) \geqslant \mu_g(F)$.
    (c) Démontrer que $\mu_d(F \vee G) = \mu_d(F) + \mu_d(G)$ et que $\mu_c(F) \geqslant \mu_d(F)$.
    (d) Expliquer comment démontrer la terminaison de l'algorithme de la question 13 en vous servant de la i-mesure μ_i. La rédaction complète de la démonstration étant particulièrement longue, les candidats et les candidates se contenteront de l'esquisser en donnant quelques éléments de celle-ci.

17. 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 construit dans ce problème.

FIN DU PROBLÈME DE LOGIQUE

3 Problème de programmation en Pascal — 75 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE PASCAL

Dans ce problème de programmation, on se propose de mettre en œuvre l'algorithme trouvé dans le problème de logique n° 2 page 2. Il n'est pas nécessaire d'avoir totalement résolu le problème de logique pour résoudre le problème de programmation : la connaissance et une bonne compréhension du sujet du problème de logique doivent suffire.

N.B. Dans les programmes, procédures et fonctions, qui seront demandés, on ne traitera pas les cas d'erreurs. On supposera que les arguments ont toujours des valeurs admissibles.

Mise en œuvre des formules

On se donne une constante exprimée en langage Pascal `VMAX = 127`. On suppose que l'ensemble $\mathcal{A}$ des variables propositionnelles est composé des variables x_i pour i entier compris entre `1` et `VMAX`. On se donne les constantes suivantes exprimées en langage Pascal : `CSTE_F = 0`, `CSTE_V = 1`, `CSTE_VARI = 2`, `CSTE_ET = 3`, `CSTE_OU = 4`, `CSTE_IMPL = 5` et `CSTE_NON = 6`.

On suppose l'existence d'un type `formule` décrivant des données Pascal servant à représenter les formules de la logique des propositions pour le programme qui sera écrit. On appellera *représentation* d'une formule la donnée du langage correspondant à cette formule. Les représentations des formules sont construites à l'aide des fonctions suivantes que l'on suppose déjà programmées et utilisables :

- `function vrai : formule ;`
  dont le résultat est la représentation de la constante **v** ;
- `function faux : formule ;`
  dont le résultat est la représentation de la constante **f** ;
- `function variable(K : integer) : formule ;`
  telle que si l'argument `K` est compris entre `1` et `VMAX` alors `variable(K)` est la représentation de la variable propositionnelle $x_{\texttt{K}}$;
- `function et(U,V : formule) : formule ;`
  telle que si `U` et `V` sont les représentations respectives des formules A et B alors `et(U,V)` est la représentation de la formule $A \wedge B$;
- `function ou(U,V : formule) : formule ;`
  telle que si `U` et `V` sont les représentations respectives des formules A et B alors `ou(U,V)` est la représentation de la formule $A \vee B$;
- `function implique(U,V : formule) : formule ;`
  telle que si `U` et `V` sont les représentations respectives des formules A et B alors `implique(U,V)` est la représentation de la formule $A \Rightarrow B$;
- `function non(U : formule) : formule ;`
  telle que si `U` est la représentation d'une formule A alors `non(U)` est la représentation de la formule $\neg A$.

Toutes les représentations de formules bien formées que nous considèrerons sont construites à l'aide des fonctions précédentes. Les fonctions suivantes sont aussi supposées être programmées et utilisables. Elles permettent d'identifier la nature d'une formule et d'accéder à ses sous-formules éventuelles.

- `function code(F : formule) : integer ;`
  qui renvoie un code numérique, `CSTE_V`, `CSTE_F`, `CSTE_VARI`, `CSTE_ET`, `CSTE_OU`, `CSTE_IMPL`, ou `CSTE_NON`, selon que la formule dont la représentation est `F` est la constante **v**, la constante **f**, une variable propositionnelle, une conjonction, une disjonction, une implication ou une négation ;
- `function indice(F : formule) : integer ;`
  qui prend en argument la représentation d'une variable propositionnelle x_i avec $i \in$ [1,`VMAX`] et qui renvoie l'indice i de cette variable ;
- `function gauche(F : formule) : formule ;`
  qui prend en argument la représentation d'une formule $A \wedge B$ ou $A \vee B$ ou $A \Rightarrow B$ et renvoie la représentation de la formule A ;
- `function droite(F : formule) : formule ;`
  qui prend en argument la représentation d'une formule $A \wedge B$ ou $A \vee B$ ou $A \Rightarrow B$ et renvoie la représentation de la formule B ;
- `function argument(F : formule) : formule ;`
  qui prend en argument la représentation d'une formule $\neg A$ et renvoie la représentation de la formule A.

Mise en œuvre des listes de formules

Pour représenter les ensembles de formules, on utilisera des listes de formules de type `liste_de_formules`. Les fonctions de construction relatives à ce type devront être les suivantes :

- `function liste_vide : liste_de_formules ;`
  qui renvoie la représentation d'une liste vide de formules ;
- `function ajoute(F : formule, L : liste_de_formules) : liste_de_formules ;`
  qui prend en argument une formule `F` et une liste de formules `L` et renvoie une liste contenant `F` et les formules déjà présentes dans `L`.

Les fonctions d'accès aux listes de formules devront être les suivantes :

- `function est_vide(L : liste_de_formules) : boolean ;`
  qui renvoie `true` si son argument est une liste vide de formules, `false` sinon ;
- `function un_element(L : liste_de_formules) : formule ;`
  qui prend en argument une liste non vide de formules et qui renvoie une formule de la liste ;
- `function reste(L : liste_de_formules) : liste_de_formules ;`
  qui prend en argument une liste `L` non vide et qui renvoie une liste de formules contenant les formules présentes dans `L` à l'exception de `un_element(L)`.

1 – Donner une définition du type `liste_de_formules`.

2 – Écrire les deux fonctions de construction.

3 – Écrire les trois fonctions d'accès.

Mise en œuvre des ensembles de formules

4 – Dans la suite du problème, on suppose toujours qu'il y a au plus `VMAX` variables propositionnelles indicées de 1 à `VMAX` où `VMAX` est une constante entière strictement positive. En raison du rôle particulier des variables propositionnelles, on décide de représenter un ensemble de formules par un enregistrement de type `ensemble_de_formules` contenant :

- un booléen valant `true` si et seulement si la constante **v** appartient à l'ensemble ;
- un booléen valant `true` si et seulement si la constante **f** appartient à l'ensemble ;
- un tableau de `VMAX` booléens, le i-ième élément de ce tableau valant `true` si et seulement si la i-ième variable propositionnelle x_i appartient à l'ensemble ;
- une liste de formules de type `liste_de_formules` contenant les formules qui ne sont pas des formules atomiques.

N.B. On acceptera que la liste de formules puisse contenir plusieurs fois une même formule. On dira alors que la liste contient plusieurs *occurrences* de la formule.

Donner une définition du type `ensemble_de_formules`.

5 – Écrire la fonction

```
function ensemble_vide : ensemble_de_formules ;
```

dont le résultat est un ensemble vide de formules.

6 – Écrire la fonction

```
function que_des_variables(E : ensemble_de_formules) : boolean ;
```

dont le résultat est `true` si l'ensemble `E` est vide ou bien ne contient que des formules atomiques, `false` sinon.

7 – Écrire la fonction

```
function intersection(E1,E2 : ensemble_de_formules) : boolean ;
```

qui renvoie `true` si l'ensemble des variables propositionnelles éléments de `E1` a une intersection non vide avec celui des variables propositionnelles éléments de `E2`, `false` sinon.

8 – Écrire la fonction

```
function ajouter_formule(E : ensemble_de_formules ;
                         F : formule               ) : ensemble_de_formules ;
```

qui a pour résultat l'ensemble de formules contenant `F` et les formules éléments de `E`.

9 – Écrire la procédure

```
procedure formule_suivante(    E : ensemble_de_formules ;
                           var R : ensemble_de_formules ;
                           var F : formule) ;
```

Cette procédure prend en argument un ensemble `E` de formules et deux variables `R` et `F`. On suppose que `E` ne contient pas que des formules atomiques. La procédure choisit une formule dans `E`, n'importe laquelle sauf une formule atomique, et la met dans `F`. Elle met dans `R`, un ensemble contenant les formules de `E` à l'exception d'une occurrence de la formule `F`.

Mise en œuvre de l'algorithme

10 – Soit `U` la représentation d'un ensemble de formules $\{F_1, \ldots, F_n\}$ avec $n \geqslant 1$ et `V` la représentation d'un ensemble de formules $\{G_1, \ldots, G_m\}$ avec $m \geqslant 1$, écrire la fonction

```
function verifier(U, V : ensemble_de_formules) : boolean ;
```

dont le résultat est `true` si la formule $F_1 \wedge \cdots \wedge F_n \Rightarrow G_1 \vee \cdots \vee G_m$ est une tautologie, `false` sinon. On s'inspirera utilement du problème de logique n° 2 page 2.

11 – À l'aide des procédures et fonctions précédentes, écrire la fonction

```
function tautologie(F : formule) : boolean ;
```

qui renvoie `true` si `F` est une tautologie, `false` sinon.

FIN DU PROBLÈME DE PROGRAMMATION EN PASCAL

4 Problème de programmation en Caml — 75 mn environ

UNIQUEMENT POUR LES CANDIDATS ET LES CANDIDATES CHOISISSANT LE LANGAGE CAML

Dans ce problème de programmation, on se propose de mettre en œuvre l'algorithme trouvé dans le problème de logique n° 2 page 2. Il n'est pas nécessaire d'avoir totalement résolu le problème de logique pour résoudre le problème de programmation : la connaissance et une bonne compréhension du sujet du problème de logique doivent suffire.

N.B. Dans les programmes qui seront demandés, on ne traitera pas les cas d'erreurs. On supposera que les arguments ont toujours des valeurs admissibles.

Mise en œuvre des formules

On se donne une constante exprimée en langage Caml : `let VMAX = 127 ;;`. On suppose que l'ensemble $\mathcal{A}$ des variables propositionnelles est composé des variables x_i pour i entier compris entre `1` et `VMAX`.

On désire définir un type `formule` décrivant des données Caml servant à représenter les formules de la logique des propositions pour le programme qui sera écrit. On appellera *représentation* d'une formule la donnée du langage correspondant à cette formule. Les *fonctions de construction* ou *constructeurs* de formules qui devront être disponibles sont les suivantes :

- `vrai` de type `formule`
  qui est la représentation de la constante **v** ;
- `faux` de type `formule`
  qui est la représentation de la constante **f** ;
- `variable : integer -> formule`
  telle que si l'argument `K` est compris entre `1` et `VMAX` alors `variable(K)` est la représentation de la variable propositionnelle $x_{\texttt{K}}$;

- `et : formule * formule -> formule`
  telle que si les arguments `U` et `V` sont les représentations respectives des formules A et B alors `et(U,V)` est la représentation de la formule $A \wedge B$;
- `ou : formule * formule -> formule`
  telle que si les arguments `U` et `V` sont les représentations respectives des formules A et B alors `ou(U,V)` est la représentation de la formule $A \vee B$;
- `implique : formule * formule -> formule`
  telle que si les arguments `U` et `V` sont les représentations respectives des formules A et B alors `implique(U,V)` est la représentation de la formule $A \Rightarrow B$;
- `non : formule -> formule`
  telle que si l'argument `U` est la représentation d'une formule A alors `non(U)` est la représentation de la formule $\neg A$.

1 – Donner une définition du type `formule`.

Mise en œuvre des listes de formules

Pour représenter les ensembles de formules, on utilisera des listes de formules de type `liste_de_formules`. Les fonctions de construction relatives à ce type devront être les suivantes :

- `liste_vide : unit -> liste_de_formules`
  qui construit la représentation d'une liste vide de formules ;
- `ajoute : formule * liste_de_formules -> liste_de_formules`
  qui prend en argument une formule `F` et une liste de formules `L` et renvoie une liste contenant `F` et les formules déjà présentes dans `L`.

Les fonctions d'accès aux listes de formules devront être les suivantes :

- `est_vide : liste_de_formules -> bool`
  qui renvoie `true` si son argument est une liste vide, `false` sinon ;
- `un_element : liste_de_formules -> formule`
  qui prend en argument une liste non vide de formules et qui renvoie une formule de la liste ;
- `reste : liste_de_formules -> liste_de_formules`
  qui prend en argument une liste `L` non vide et qui renvoie une liste de formules contenant les formules présentes dans `L` à l'exception de `un_element(L)`.

2 – Donner une définition du type `liste_de_formules`.

3 – Écrire les deux fonctions de construction.

4 – Écrire les trois fonctions d'accès.

Mise en œuvre des ensembles de formules

5 – Dans la suite du problème, on suppose toujours qu'il y a au plus `VMAX` variables propositionnelles indicées de 1 à `VMAX` où `VMAX` est une constante entière strictement positive. En raison du rôle particulier des variables propositionnelles, on décide de représenter un ensemble de formules par une donnée de type `ensemble_de_formules` contenant :

- un booléen valant `true` si et seulement si la constante **v** appartient à l'ensemble ;
- un booléen valant `true` si et seulement si la constante **f** appartient à l'ensemble ;
- un tableau de `VMAX` booléens, le i-ième élément de ce tableau valant `true` si et seulement si la i-ième variable propositionnelle x_i appartient à l'ensemble ;
- une liste de formules de type `liste_de_formules` contenant les formules qui ne sont pas des formules atomiques.

N.B. On acceptera que la liste de formules puisse contenir plusieurs fois une même formule. On dira alors que la liste contient plusieurs *occurrences* de la formule.

Donner une définition du type `ensemble_de_formules`.

6 – Écrire la fonction

```
ensemble_vide : unit -> ensemble_de_formules
```

dont le résultat est un ensemble vide de formules.

7 – Écrire la fonction

```
que_des_variables : ensemble_de_formules -> bool
```

dont le résultat est `true` si l'argument est vide ou bien ne contient que des représentations de formules atomiques, `false` sinon.

8 – Écrire la fonction

```
intersection : ensemble_de_formules * ensemble_de_formules -> bool
```

qui prend en arguments deux ensembles de formules `E1` et `E2` ; elle renvoie `true` si l'ensemble des variables propositionnelles éléments de `E1` a une intersection non vide avec celui des variables propositionnelles éléments de `E2`, `false` sinon.

9 – Écrire la fonction

```
ajouter_formule : ensemble_de_formules * formule -> ensemble_de_formules
```

Cette fonction a pour résultat l'ensemble de formules contenant les formules éléments de son premier argument et son deuxième argument.

10 – Écrire la fonction

```
formule_suivante : ensemble_de_formules -> ensemble_de_formules * formule
```

Cette fonction prend en argument un ensemble `E` de formules. On suppose que `E` ne contient pas que des formules atomiques. La fonction choisit une formule `F` dans `E`, n'importe laquelle sauf une formule atomique. Elle a pour résultat le couple `(F,R)` où `R` est un ensemble de formules contenant les formules de `E` à l'exception d'une occurrence de la formule `F`.

Mise en œuvre de l'algorithme

11 – Écrire la fonction

```
verifier : ensemble_de_formules * ensemble_de_formules -> bool
```

prenant en arguments deux ensembles de formules `U` et `V`. Si `U` est la représentation d'un ensemble de formules $\{F_1, \ldots, F_n\}$ avec $n \geqslant 1$ et si `V` est la représentation d'un ensemble de formules $\{G_1, \ldots, G_m\}$ avec $m \geqslant 1$ alors le résultat de cette fonction doit être `true` si la formule $F_1 \wedge \cdots \wedge F_n \Rightarrow G_1 \vee \cdots \vee G_m$ est une tautologie, `false` sinon. On s'inspirera utilement du problème de logique n° 2 page 2.

12 – À l'aide des fonctions précédentes, écrire la fonction

```
tautologie : formule -> bool
```

qui renvoie `true` si son argument est une tautologie, `false` sinon.

FIN DU PROBLÈME DE PROGRAMMATION EN CAML

FIN DE L'ÉPREUVE