ÉCOLE NATIONALE DES PONTS ET CHAUSSÉES,
ÉCOLES NATIONALES SUPÉRIEURES DE L'AÉRONAUTIQUE ET DE L'ESPACE,
DE TECHNIQUES AVANCÉES, DES TÉLÉCOMMUNICATIONS,
DES MINES DE PARIS, DES MINES DE SAINT-ÉTIENNE, DES MINES DE NANCY,
DES TÉLÉCOMMUNICATIONS DE BRETAGNE,
ÉCOLE POLYTECHNIQUE (FILIÈRE TSI)

CONCOURS D'ADMISSION 2010

**ÉPREUVE D'INFORMATIQUE**

**Filière MP**
**Durée de l'épreuve : 3 heures.**
**L'usage de calculettes est autorisé. L'utilisation d'un ordinateur est interdite.**

**Sujet mis à disposition des concours : ENSAE ParisTech, TELECOM SudParis (ex INT), TPE-EIVP**

*L'énoncé de cette épreuve comporte 11 pages.*

*Les candidats sont priés de mentionner de façon apparente sur la première page de la copie :*

*INFORMATIQUE - MP*

**Recommandations aux candidats**

- **Si, au cours de l'épreuve, un candidat repère ce qui lui semble être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à 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.**

**Composition de l'épreuve**

L'épreuve comporte un seul problème

## Le problème de la satisfiabilité logique

**Préliminaire concernant la programmation** : il faudra écrire des fonctions ou des procédures à l'aide d'un langage de programmation qui pourra être soit **Caml**, soit **Pascal**, tout autre langage étant exclu. **Indiquer en début de problème le langage de programmation choisi ; il est interdit de modifier ce choix au cours de l'épreuve**. Certaines questions du problème sont formulées différemment selon le langage de programmation ; cela est indiqué chaque fois que nécessaire. Par ailleurs, pour écrire une fonction ou une procédure en langage de programmation, le candidat pourra définir des fonctions ou des procédures auxiliaires qu'il explicitera ou faire appel à d'autres fonctions ou procédures définies dans les questions précédentes.

Dans l'énoncé du problème, un même identificateur écrit dans deux polices de caractères différentes désigne la même entité, mais du point de vue mathématique pour la police écrite en italique (par exemple : $F$) et du point de vue informatique pour celle écrite en romain (par exemple : `F`).

On appelle *variable booléenne* une variable qui ne peut prendre que les valeurs 0 (synonyme de *faux*) ou 1 (synonyme de *vrai*). Si $x$ est une variable booléenne, on appelle complémenté de $x$ et on note $\overline{x}$ la négation de $x$ : $\overline{x}$ vaut 1 si $x$ vaut 0 et $\overline{x}$ vaut 0 si $x$ vaut 1.

On appelle *littéral* une variable booléenne ou son complémenté. Pour un littéral $\alpha = \overline{x}$, où $x$ est une variable booléenne, on définit le complémenté $\overline{\alpha}$ de $\alpha$ comme étant égal à $x$. On a ainsi défini le complémenté de tout littéral.

On représente la disjonction (« ou » logique ) par le symbole $\vee$ et la conjonction (« et » logique) par le symbole $\wedge$.

On appelle *clause* une disjonction de littéraux et *longueur d'une clause* le nombre des littéraux qui composent cette clause. La longueur d'une clause doit être toujours au moins égale à 1.

On appelle *formule logique sous forme normale conjonctive* une conjonction de clauses ; chacune de ces clauses est dite *appartenir* à la formule logique. Dans tout le problème, **quand on utilisera l'expression *formule logique*, il s'agira d'une formule logique sous forme normale conjonctive**. On ne suppose pas que toutes les clauses d'une formule logique sont distinctes.

Dans tout le problème, **les littéraux d'une même clause sont différents et une clause ne contient pas à la fois une variable et le complémenté de celle-ci** : si une clause contient le littéral $\alpha$, elle ne contient pas une deuxième fois $\alpha$ et elle ne contient pas $\overline{\alpha}$. Dans les algorithmes qui seront à écrire, on fera en sorte que cette propriété soit toujours vérifiée. En conséquence, la longueur d'une clause d'une formule logique n'est jamais supérieure au nombre total de variables de la formule logique considérée.

On appelle *valuation des variables d'une formule logique* une application de l'ensemble de ces variables dans l'ensemble $\{0, 1\}$. Une clause vaut 1 si au moins un de ses littéraux vaut 1 et 0 sinon. Une *clause* est dite *satisfaite* par une valuation des variables si elle vaut 1 pour cette valuation. Une formule logique vaut 1 si toutes ses clauses valent 1 et 0 sinon. Une *formule logique* est dite *satisfaite* par une valuation des variables si elle vaut 1 pour cette valuation. Une formule logique est dite *satisfiable* s'il existe une valuation de ses variables qui la satisfait. Si une formule logique est satisfiable, on appelle alors *solution* de cette formule logique toute valuation des variables qui la satisfait. Par exemple, la formule logique :

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

est satisfiable et elle est satisfaite par la solution $x = 1$, $y = 0$ et $z = 1$.

Comme pour une variable booléenne, la valeur 0 est synonyme de *faux* et la valeur 1 est synonyme de *vrai* pour un littéral, une clause ou une formule logique.

Une formule logique qui n'aurait aucune clause est dite vide et considérée comme toujours satisfaite.

Étant donnée une formule logique $F$, on note dans ce problème $max(F)$ le nombre maximum de clauses de $F$ pouvant être satisfaites par une même valuation ; en notant $m$ le nombre de clauses de $F$, on remarque que $F$ est satisfiable si et seulement si $max(F) = m$.

## Première partie : introduction

**Pour les deux premières questions de ce problème**, on considère trois personnes nommées $X$, $Y$ et $Z$ et on s'intéresse au fait qu'elles portent ou non un chapeau. On considère les propositions suivantes :

a) au moins une des personnes porte un chapeau ;
b) au moins une des personnes ne porte pas de chapeau ;
c) si $X$ porte un chapeau, ni $Y$ ni $Z$ n'en portent ;
d) si $Y$ porte un chapeau, au moins une personne parmi $X$ ou $Z$ en porte un.

On définit des variables booléennes $x$, $y$ et $z$ qui valent 1 si, respectivement, $X$, $Y$ et $Z$ portent un chapeau et 0 sinon.

❒ 1 – Exprimer chacune des propositions a), b), c) et d) comme une formule logique (on rappelle que l'expression « formule logique » signifie « formule logique sous forme normale conjonctive ») exprimée avec les variables $x$, $y$ et $z$.

❒ 2 – Écrire une formule logique exprimant le fait que les propositions a), b), c) et d) doivent être satisfaites simultanément. Indiquer si cette formule logique est satisfiable ou non et, si elle est satisfiable, donner l'ensemble des solutions pour cette formule logique. On prouvera la réponse.

On considère maintenant la formule logique $F1$ dépendant de quatre variables $x$, $y$, $z$ et $t$ :

$$F1 = (x \vee y \vee z) \wedge (\overline{x} \vee z \vee \overline{t}) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (x \vee y \vee \overline{t}) \wedge (\overline{x} \vee \overline{z} \vee \overline{t}) \wedge (\overline{x} \vee t) \wedge (x \vee \overline{y} \vee z).$$

❒ 3 – Indiquer si $F1$ est satisfiable ou non et, si elle est satisfiable, donner l'ensemble des solutions de $F1$. On prouvera la réponse.

On considère une formule logique $F$ définie sur $n$ variables booléennes et dont **toutes les clauses sont de longueur 3** ; on dit alors qu'il s'agit d'une *instance de 3-SAT* ; on note $m$ le nombre de clauses dont $F$ est la conjonction.

❒ 4 – Déterminer une instance $F2$ de 3-SAT non satisfiable et possédant exactement 8 clauses ; indiquer $max(F2)$ en justifiant la réponse.

On considère une instance $F$ de 3-SAT définie sur $n$ variables booléennes. On note $V$ l'ensemble des $2^n$ valuations des variables de $F$. Soit *val* une valuation des $n$ variables ; si $C$

est une clause, on note $\varphi(C, val)$ la valeur de $C$ pour la valuation $val$ ; on note $\psi(F, val)$ le nombre de clauses de $F$ qui valent 1 pour la valuation $val$. On a : $\psi(F, val) = \sum_{C, \text{clause de } F} \varphi(C, val)$ et $max(F) = \max_{val \in V} \psi(F, val)$.

❒ 5 – On considère une clause $C$ de $F$ ; indiquer en fonction de $n$ la valeur de la somme : $\sum_{val \in V} \varphi(C, val)$.

❒ 6 – En considérant la somme $\sum_{C, \text{clause de } F} \sum_{val \in V} \varphi(C, val)$, indiquer en fonction de $m$ un minorant de $max(F)$.

❒ 7 – Donner le nombre minimum de clauses d'une instance de 3-SAT non satisfiable.

**Indications pour la programmation, à lire avec attention**

On va désormais numéroter à partir de 1 les variables booléennes d'une formule logique. Ainsi, si les variables sont $x$, $y$ et $z$, on associe à $x$ le numéro 1, à $y$ le numéro 2 et à $z$ le numéro 3. Par ailleurs, on numérote le complémenté d'une variable avec l'opposé du numéro associé à celle-ci ; ainsi, avec l'exemple ci-dessus, au littéral $\overline{x}$, on associe le numéro –1, au littéral $\overline{y}$, on associe le numéro –2, au littéral $\overline{z}$, on associe le numéro –3. **Pour alléger les explications, on identifie désormais un littéral avec son numéro.**

Le mot *tableau* utilisé plus bas est synonyme de ce qu'on appelle *vecteur* en Caml.

Pour coder une valuation d'un ensemble de $n$ variables booléennes, on utilise un tableau d'au moins $n + 1$ entiers indicé à partir de 0 ; pour $i$ compris entre 1 et $n$, la case d'indice $i$ donne la valeur de la variable de numéro $i$ (valeur qui vaut soit 0, soit 1, ou éventuellement une valeur quelconque si la valeur de la variable correspondante n'est pas spécifiée). La case d'indice 0 n'est pour l'instant pas utilisée.

Pour coder une clause de longueur $p$, on utilise un tableau d'au moins $p + 1$ entiers indicé à partir de 0 ; pour $i$ compris entre 1 et $p$, la case d'indice $i$ contient le numéro du littéral qui se trouve en position $i$ dans la clause. La case d'indice 0 de ce tableau indique la longueur de la clause. Ainsi, si les variables sont $x$, $y$, $z$ et $t$, numérotées respectivement par 1, 2, 3 et 4, la clause $(x \vee \overline{t} \vee y)$ est codé par un tableau représenté ci-dessous :

| Indice | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| Numéro | 3 | 1 | –4 | 2 |

Pour coder une formule logique, on utilise un tableau de tableaux d'entiers indicés par $(i, j)$ où les indices $i$ et $j$ sont supérieurs ou égaux à 0 ; pour $i$ supérieur ou égal à 1, la ligne d'indice $i$ décrit la $i^e$ clause de la formule logique. On utilise deux cases de la ligne d'indice 0 : **la case d'indice (0, 0) contient le nombre de clauses de la formule logique et la case d'indice (0, 1) contient le nombre total de variables**. Ainsi, la formule logique :

$F3 = (x \vee \overline{y} \vee z \vee t) \wedge (\overline{x} \vee \overline{z}) \wedge (x \vee \overline{t} \vee y)$

est codée par le tableau ci-dessous. Dans ce tableau, les cases contenant des pointillés existent ou non ; si elles existent, leurs valeurs n'ont pas d'importance :

| *i* \ *j* | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 3 | 4 | … | … | … |
| 1 | 4 | 1 | –2 | 3 | 4 |
| 2 | 2 | –1 | –3 | … | … |
| 3 | 3 | 1 | –4 | 2 | … |

**Indications pour Caml**
En Caml, la formule logique *F3* peut être définie comme ci-dessous par le vecteur de vecteurs `F3` :

```
let F3 = [|[|3;4|];[|4;1;-2;3;4|];[|2;-1;-3|];[|3;1;-4;2|]|];;
```

Soit `F` un vecteur de vecteurs définissant une formule logique *F* et soient *i* et *j* deux entiers compris entre 1 et `F.(0).(0)` ; le vecteur `F.(i)` définit la $i^e$ clause de *F* ; dans un calcul de complexité, on considérera l'instruction `F.(i) <- F.(j)` comme une seule opération élémentaire.
**Fin des indications pour Caml**

**Indications pour Pascal**
On définit les constantes et les types ci-dessous :

```
CONST
     MAX_VAR = 10;
     MAX_CLAUSES = 100;

type Valuation = array [0 .. MAX_VAR] of Integer;

type Clause =  array [0 .. MAX_VAR] of Integer;

type Formule = array [0 .. MAX_CLAUSES] of Clause;
```

La constante `MAX_VAR` indique le nombre maximum de variables d'une formule logique. La constante `MAX_CLAUSES` indique le nombre maximum de clauses d'une formule logique. Dans ce problème, on ne se préoccupera pas d'un éventuel débordement de ces tableaux.
La formule logique *F3* peut être codée avec une variable `F3` de type `Formule` par :

```
F3[0][0]:=3; F3[0][1]:=4;
F3[1][0]:=4; F3[1][1]:=1;  F3[1][2]:=-2; F3[1][3]:=3; F3[1][4]:=4;
F3[2][0]:=2; F3[2][1]:=-1; F3[2][2]:=-3;
F3[3][0]:=3; F3[3][1]:=1;  F3[3][2]:=-4; F3[3][3]:=2;
```

Soit `F` un tableau de type `Formule` définissant une formule logique *F* et soient *i* et *j* deux entiers compris entre 1 et `F[0][0]` ; le tableau `F[i]` définit la $i^e$ clause de *F* ; dans un calcul de complexité, on considérera l'instruction `F[i] := F[j]` comme une seule opération élémentaire.
**Fin des indications pour Pascal**

## Deuxième partie : satisfiabilité, méthode exacte

❒ 8 – La fonction `valeur_clause`.
**Caml** : Écrire en Caml une fonction `valeur_clause` telle que, si `C` est un vecteur d'entiers codant une clause *C* et si `val` est un vecteur codant une valuation *val* d'un ensemble de variables contenant les variables de *C*, alors

```
valeur_clause C val
```

donne la valeur de *C* (c'est-à-dire 0 ou 1) pour la valuation *val*.
Indiquer la complexité de la fonction `valeur_clause`.

**Pascal** : Écrire en Pascal une fonction `valeur_clause` telle que, si `C`, de type `Clause`, code une clause *C* et si `val`, de type `Valuation`, code une valuation *val* d'un ensemble de variables contenant les variables de *C*, alors

```
valeur_clause(C, val)
```

donne la valeur de la clause *C* (c'est-à-dire 0 ou 1) pour la valuation *val*.
Indiquer la complexité de la fonction `valeur_clause`.

❒ 9 – La fonction `satisfait_formule`.
**Caml** : Écrire en Caml une fonction `satisfait_formule` telle que, si `F` est un vecteur de vecteurs d'entiers codant une formule logique *F* et si `val` est un vecteur codant une valuation *val* de l'ensemble des variables de *F*, alors

```
satisfait_formule F val
```

vaut `true` si *val* satisfait *F* et `false` sinon.
Indiquer la complexité de la fonction `satisfait_formule`.

**Pascal** : Écrire en Pascal une fonction `satisfait_formule` telle que, si `F`, de type `Formule`, code une formule logique *F* et si `val`, de type `Valuation`, code une valuation *val* de l'ensemble des variables de *F*, alors

```
satisfait_formule(F, val)
```

vaut `true` si *val* satisfait *F* et `false` sinon.
Indiquer la complexité de la fonction `satisfait_formule`.

❒ 10 – La fonction `resoudre_rec`.
On considère une formule logique *F* et un nombre *k* compris entre 0 et le nombre total de variables de *F*. Pour *i* compris entre 1 et *k*, on fixe une valeur $v_i$ égale à 0 ou 1 pour la variable *i* (si *k* vaut 0, aucune variable n'a une valeur fixée). La fonction `resoudre_rec` détermine s'il existe une valuation *val* des variables telle que

- pour *i* compris entre 1 et *k*, la valeur dans *val* de la variable *i* est $v_i$,
- *val* satisfait *F*.

De plus, la fonction `resoudre_rec` dispose d'un tableau d'entiers destiné à coder une valuation des variables ; dans le cas où la valuation cherchée existe, la fonction modifie ce tableau pour qu'il code une telle valuation et elle renvoie alors la valeur 1 ; si la valuation cherchée n'existe pas, la fonction renvoie la valeur 0 sans avoir modifié les cases d'indices compris entre 1 et *k*.
**Caml** : Écrire en Caml une fonction récursive `resoudre_rec` telle que,

- si `F` est un vecteur de vecteurs d'entiers codant une formule logique *F*,
- si `val` est un vecteur d'entiers servant à coder une valuation,
- si *k* est un entier compris entre 0 et le nombre de variables de *F*,

- si le vecteur `val` contient soit 0, soit 1 dans les cases d'indices 1, 2, …, $k$,

alors `resoudre_rec F val k` modifie le vecteur `val` comme indiqué ci-dessus, en considérant que les valeurs des cases d'indices 1, 2, …, $k$ de ce vecteur sont les valeurs fixées $v_1$, $v_2$, …, $v_k$ des variables 1, 2, …, $k$ ; la fonction renvoie `true` si la valuation cherchée existe et `false` sinon.

**Pascal** : Écrire en Pascal une fonction récursive `resoudre_rec` telle que,

- si `F`, de type `Formule`, code une formule logique $F$,
- si `val` est de type `Valuation`,
- si $k$ est un entier compris entre 0 et le nombre de variables de $F$,
- si le tableau `val` contient soit 0, soit 1 dans les cases d'indices 1, 2, …, $k$,

alors `resoudre_rec(F, val, k)` modifie le tableau `val` comme indiqué ci-dessus, en considérant que les valeurs des cases d'indices 1, 2, …, $k$ de ce tableau sont les valeurs fixées $v_1$, $v_2$, …, $v_k$ des variables 1, 2, …, $k$ ; la fonction renvoie `true` si la valuation cherchée existe et `false` sinon.

❒ 11 – La fonction `resoudre`.
La fonction *resoudre* prend en argument une formule logique $F$ et renvoie un tableau d'entiers pouvant coder une valuation des variables de $F$. Si $F$ est satisfiable, le tableau renvoyé code une solution de $F$ et contient la valeur 1 dans la case d'indice 0 ; si $F$ n'est pas satisfiable, le tableau contient la valeur 0 dans la case d'indice 0.
**Caml** : Écrire en Caml une fonction `resoudre` telle que, si `F` est un vecteur de vecteurs d'entiers codant une formule logique $F$, alors `resoudre F` renvoie un vecteur d'entiers correspondant au résultat de la fonction *resoudre* décrite ci-dessus.

**Pascal** : Écrire en Pascal une fonction `resoudre` telle que, si `F`, de type `Formule`, code une formule logique $F$, alors `resoudre(F)` renvoie un tableau de type `Valuation` correspondant au résultat de la fonction *resoudre* décrite ci-dessus.

❒ 12 – Évaluer la complexité de la fonction *resoudre* appliquée à une formule logique $F$ en fonction du nombre $n$ de variables de $F$ et de la somme des longueurs des clauses de $F$.

## Troisième partie : MAX-SAT

Dans cette partie, on ne s'intéresse plus à savoir si une formule logique est satisfiable mais au calcul, pour une formule logique $F$, de $max(F)$. On va définir une heuristique, c'est-à-dire une méthode qui ne donne pas nécessairement la valeur de $max(F)$ mais une valeur approchée, qu'on souhaite proche de l'optimum : on calcule une valuation en choisissant une à une les variables et leurs valeurs. Plus précisément, soit $F$ une formule logique ; étant donnée une variable $\alpha$ de $F$, on note $diff(F, \alpha)$ le nombre de fois où $\alpha$ figure dans $F$ diminué du nombre de fois où $\overline{\alpha}$ figure dans $F$ ; l'heuristique utilise la transformation suivante :

a) On calcule pour chaque variable $\alpha$ de $F$ le nombre $diff(F, \alpha)$.
b) Si $p$ est un entier relatif, on note abs($p$) la valeur absolue de $p$. On détermine une variable $\alpha_0$ de $F$ telle que, pour toute variable $\alpha$ de $F$, on ait :

$$\text{abs}(diff(F, \alpha_0)) \geq \text{abs}(diff(F, \alpha)).$$

c) On pose $\alpha_0 = 1$ si on a $diff(F, \alpha_0) > 0$ et $\alpha_0 = 0$ sinon.
d) On supprime la variable $\alpha_0$ de $F$ en tenant compte de la valeur choisie de façon à ne conserver que les clauses qui restent à satisfaire après le choix de la valeur de $\alpha_0$ ; on comptabilise le nombre de clauses satisfaites. On obtient ainsi une formule logique notée $F'$ qui est la formule transformée à partir de $F$.

Pour exécuter l'heuristique, on transforme la formule $F$ comme décrit ci-dessus, puis on transforme de même la formule $F'$, puis on transforme la formule transformée à partir de $F'$ et ainsi de suite jusqu'à obtenir une formule vide. On somme au fur et à mesure les nombres de clauses satisfaites comptabilisés pendant chaque transformation ; le résultat de l'heuristique est constitué de cette somme et d'une valuation correspondant aux choix effectués pendant les transformations successives pour les valeurs des variables.

❒ 13 – Appliquer l'heuristique à la formule *F1* définie avant la question ❒ 3 ; détailler les différentes étapes.

De façon à écrire l'heuristique en langage de programmation, on définit cinq fonctions ou procédures dans les cinq questions suivantes.

❒ 14 – La fonction `place`.
**Caml** : Écrire en Caml une fonction `place` telle que, si `C` est un vecteur codant une clause *C* et si *litt* est un littéral, `place C litt` renvoie

- 0 si *litt* ne figure pas dans *C*,
- la position de *litt* dans *C* si *litt* figure dans *C* (valeur comprise entre 1 et le nombre de littéraux de *C*).

Évaluer la complexité de la fonction `place`.

**Pascal** : Écrire en Pascal une fonction `place` telle que, si `C`, de type `Clause`, code une clause *C* et si *litt* est un littéral, `place(C, litt)` renvoie

- 0 si *litt* ne figure pas dans *C*,
- la position de *litt* dans *C* si *litt* figure dans *C* (valeur comprise entre 1 et le nombre de littéraux de *C*).

Évaluer la complexité de la fonction `place`.

❒ 15 – La fonction ou procédure `supprimer_variable`.
**Caml** : Écrire en Caml une fonction `supprimer_variable` telle que, si `C` est un vecteur codant une clause *C* et si *i* est un entier compris entre 1 et le nombre de littéraux de *C*, alors `supprimer_variable C i` modifie `C` pour que `C` code la clause obtenue en supprimant de la clause *C* le littéral d'indice *i*. La complexité de cette fonction doit être de l'ordre d'une constante, et donc ne pas dépendre du nombre de littéraux de la clause *C* ni de la valeur de *i*.

**Pascal** : Écrire en Pascal une procédure `supprimer_variable` telle que, si `C`, de type `Clause`, code une clause *C* et si *i* est un entier compris entre 1 et le nombre de littéraux de *C*, alors `supprimer_variable(C, i)` modifie `C` pour que `C` code la

clause obtenue en supprimant de la clause $C$ le littéral d'indice $i$. La complexité de cette fonction doit être de l'ordre d'une constante, et donc ne pas dépendre du nombre de littéraux de la clause $C$ ni de la valeur de $i$.

❒ 16 – La fonction ou procédure `supprimer_clause`.
**Caml** : Écrire en Caml une fonction `supprimer_clause` telle que, si `F` est un vecteur de vecteurs d'entiers codant une formule logique $F$ et si `i` est un entier compris entre 1 et le nombre de clauses de $F$, alors `supprimer_clause F i` modifie `F` pour que `F` code la formule logique obtenue en supprimant la clause d'indice `i`. La complexité de cette fonction doit être de l'ordre d'une constante, et donc ne pas dépendre du nombre de clauses de la formule logique ni de la valeur de `i`.

**Pascal** : Écrire en Pascal une procédure `supprimer_clause` telle que, si `F`, de type `Formule`, code une formule logique $F$ et si `i` est un entier compris entre 1 et le nombre de clauses de $F$, alors `supprimer_clause(F, i)` modifie `F` pour que `F` code la formule logique obtenue en supprimant la clause d'indice `i`. La complexité de cette fonction doit être de l'ordre d'une constante, et donc ne pas dépendre du nombre de clauses de la formule logique ni de la valeur de `i`.

❒ 17 – La fonction `calculer_diff`.
**Caml** : Écrire en Caml une fonction `calculer_diff` telle que, si `F` est un vecteur de vecteurs d'entiers codant une formule logique $F$, alors `calculer_diff F` renvoie un vecteur d'entiers donnant pour chaque variable $\alpha$ de $F$ la valeur de *diff*($F$, $\alpha$) décrite au-dessus de la question ❒ 13. En supposant que toutes les variables (dont le nombre figure dans la case d'indice 1 de la ligne d'indice 0 de `F`) figurent effectivement dans $F$ directement ou par son complémenté, cette fonction doit avoir une complexité de l'ordre de la somme des longueurs des clauses de $F$.

**Pascal** : Écrire en Pascal une fonction `calculer_diff` telle que, si `F`, de type `Formule`, code une formule logique $F$, alors `calculer_diff(F)` renvoie un tableau de type `Valuation` donnant pour chaque variable $\alpha$ de $F$ la valeur de *diff*($F$, $\alpha$) décrite au-dessus de la question ❒ 13. En supposant que toutes les variables (dont le nombre figure dans la case d'indice 1 de la ligne d'indice 0 de `F`) figurent effectivement dans $F$ directement ou par son complémenté, cette fonction doit avoir une complexité de l'ordre de la somme des longueurs des clauses de $F$.

❒ 18 – La fonction `simplifier`.
On s'intéresse dans cette question à une transformation automatique d'une formule logique $F$ lorsqu'on pose qu'une variable $\alpha$ prend la valeur $v$. Cette transformation est effectuée à l'aide d'une fonction nommée *simplifier* prenant $F$, $\alpha$ et $v$ comme arguments. Si $v$ vaut 1, on note *litt* le littéral $\alpha$ et sinon on note *litt* le littéral $\overline{\alpha}$. Quand $\alpha$ prend la valeur $v$, *litt* prend la valeur 1 et le complémenté de *litt* prend la valeur 0. Les clauses contenant *litt*, prenant la valeur 1, sont supprimées de $F$. Pour chaque clause contenant le complémenté de *litt*, on retire ce complémenté ; si une clause était réduite au complémenté de *litt*, on supprime une telle clause. Les clauses qui ne contiennent ni *litt* ni le complémenté de *litt* sont inchangées. La fonction *simplifier* compte le nombre de clauses qui contenaient *litt* avant simplification et renvoie ce nombre.

**Caml** : Écrire en Caml la fonction `simplifier` telle que, si `F` est un vecteur de vecteurs d'entiers codant une formule logique *F*, si `alpha` est un entier représentant une variable $\alpha$ de *F* et si `v` vaut 0 ou 1, alors

```
simplifier F alpha v
```

transforme `F` pour que `F` code la formule logique obtenue à partir de *F* par la fonction *simplifier* et renvoie le nombre de clauses de la formule logique *F* initiale qui contenaient *litt* défini ci-dessus.
Évaluer la complexité de la fonction `simplifier`.

**Pascal** : Écrire en Pascal la fonction `simplifier` telle que, si `F`, de type `Formule`, code une formule logique *F*, si `alpha` est un entier représentant une variable $\alpha$ de *F* et si `v` vaut 0 ou 1, alors

```
simplifier(F, alpha, v)
```

transforme `F` pour que `F` code la formule logique obtenue à partir de *F* par la fonction *simplifier* et renvoie le nombre de clauses de la formule logique *F* initiale qui contenaient *litt* défini ci-dessus.
Évaluer la complexité de la fonction `simplifier`.

❒ 19 – La fonction `heuristique`.
**Caml** : Écrire en Caml une fonction `heuristique` telle que, si `F` est un vecteur de vecteurs d'entiers codant une formule logique *F*, alors `heuristique F` renvoie un vecteur d'entiers codant la valuation des variables résultant de l'heuristique décrite au début de cette partie ; la case d'indice 0 de ce même vecteur devra contenir le nombre de clauses satisfaites par la valuation déterminée par l'heuristique.
Évaluer la complexité de la fonction `heuristique`.

**Pascal** : Écrire en Pascal une fonction `heuristique` telle que, si `F`, de type `Formule`, code une formule logique *F*, alors `heuristique(F)` renvoie un tableau de type `Valuation` codant la valuation des variables résultant de l'heuristique décrite au début de cette partie ; la case d'indice 0 de ce même tableau devra contenir le nombre de clauses satisfaites par la valuation déterminée par l'heuristique.
Évaluer la complexité de la fonction `heuristique`.

## Quatrième partie : étude d'un cas particulier

On revient au problème de la satisfiabilité. On s'intéresse dans cette partie à une formule logique dans laquelle **chaque littéral apparaît au plus une fois** et dans laquelle **toutes les clauses sont de longueur au moins 2**. On appelle dans ce problème *formule logique 1-occ* une telle formule logique. Par exemple, la formule logique *F4* ci-dessous, ayant six variables *x*, *y*, *z*, *t*, *u*, *v*, est une formule logique 1-occ :

$F4 = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{y} \vee \overline{v}) \wedge (u \vee v) \wedge (\overline{t} \vee \overline{u})$

On cherche à déterminer une solution d'une formule logique 1-occ.

❒ 20 – On considère une formule logique 1-occ qui s'écrit :

$$F = (x \vee l_1 \vee l_2 \ldots \vee l_k) \wedge (\overline{x} \vee l_{k+1} \vee l_{k+2} \vee \ldots \vee l_{k+h}) \wedge G$$

avec $k \geq 1$, $h \geq 1$, où $x$ est une variable, où $l_1, \ldots, l_{k+h}$ sont des littéraux et où $G$ est une formule logique 1-occ éventuellement vide.

a) Montrer que si $\{l_1, l_2, \ldots, l_k, l_{k+1}, l_{k+2}, \ldots, l_{k+h}\}$ contient à la fois une variable et son complémenté, alors $F$ est satisfiable si et seulement si $G$ est satisfiable ; on dira dans ce cas que *$F$ réduite par rapport à $x$* donne la formule logique $G$.

b) On suppose que $\{l_1, l_2, \ldots, l_k, l_{k+1}, l_{k+2}, \ldots, l_{k+h}\}$ ne contient jamais à la fois une variable et son complémenté. Indiquer une formule logique 1-occ $F'$ ne contenant ni $x$ ni $\overline{x}$ telle que $F$ est satisfiable si et seulement si $F'$ est satisfiable. On dira dans ce cas que *$F$ réduite par rapport à $x$* donne la formule logique $F'$.

*Remarque* : on rappelle que, par définition dans ce problème, une clause ne contient pas à la fois une variable et son complémenté et qu'une formule logique vide est considérée comme toujours satisfaite.

❒ 21 – Exemples de réduction

a) On considère la formule logique $(x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{y} \vee \overline{t})$ ; indiquer la formule logique obtenue en réduisant celle-ci par rapport à $x$.

b) On considère la formule logique $(\overline{x} \vee \overline{z} \vee t) \wedge (\overline{t} \vee \overline{u}) \wedge (z \vee u)$ ; indiquer la formule logique obtenue en réduisant celle-ci par rapport à $t$.

❒ 22 – Montrer que toute formule logique 1-occ est satisfiable.

❒ 23 – Décrire sans utiliser le langage de programmation une fonction (ou une procédure) récursive *calculer_solution*

- qui prend en paramètre une formule logique 1-occ $F$ et un tableau *val* permettant de coder une valuation des variables ;
- qui transforme le tableau *val* pour que, après exécution de la fonction (ou de la procédure), *val* contienne une solution de $F$.

❒ 24 – Appliquer la fonction (ou la procédure) *calculer_solution* décrite dans la question précédente à la formule *F4*. Détailler chaque appel récursif.