D.S. 02

Problème SAT, mines 2010

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.
- Pour écrire une fonction, le candidat pourra définir des fonctions auxiliaires qu'il explicitera ou faire appel à d'autres fonctions ou procédures définies dans les questions précédentes.

Notations

1. 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 (F) et du point de vue informatique pour celle écrite en romain (`F`).
2. On appelle variable booléenne une variable qui ne peut prendre que les valeurs *faux* (`false`) ou *vrai* (`true`). Si x est une variable booléenne, on appelle complémenté de x, noté $\overline{x}$, la négation de x : $\neg x$.
3. On appelle littéral une variable booléenne ou son complémenté. Pour $p = \overline{x}$, on définit le complémenté $\overline{p}$ de p comme étant égal à x.
4. On représente la disjonction ("ou" logique) par $\wedge$ et la conjonction ("et" logique) par $\vee$.
5. On appelle clause une disjonction de littéraux et longueur d'une clause le nombre des littéraux qui composent cette clause. Cette longueur doit être au moins égale à 1.
6. 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.
7. 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 p, elle ne contient pas une deuxième fois p et elle ne contient pas $\overline{p}$. 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.

8. On appelle **valuation** des variables d'une formule logique une application de l'ensemble de ces variables dans l'ensemble $\{vrai, faux\}$. On prolonge les valuations sur les clauses : une clause vaut *vrai* si au moins un de ses littéraux vaut *vrai* et *faux* sinon.

   Une clause est dite satisfaite par une valuation des variables si elle vaut *vrai* pour cette valuation. Une formule logique vaut *vrai* si toutes ses clauses valent *vrai* et *faux* sinon.
9. Une formule logique est dite satisfaite par une valuation des variables si elle vaut *vrai* 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 $x \mapsto vrai$, $y \mapsto faux$ et $z \mapsto vrai$.
10. Une formule logique qui n'aurait aucune clause est dite vide et considérée comme satisfaite.
11. 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 $\max(F) = m$.

1 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 définit des variables booléennes x, y et z qui valent *vrai* si, respectivement, X, Y et Z portent un chapeau et *faux* sinon.
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.

Question 1
Exprimer chacune des propositions a), b), c) et d) comme une formule logique (sous forme normale conjonctive) exprimée avec les variables x, y et z.

Question 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 F_1 à quatre variables x, y, z et t :
$F_1 = (x \vee y \vee z) \wedge (\overline{x} \vee z \vee \overline{t}) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (x \vee y \vee \overline{t}) \wedge (\overline{x} \vee \overline{z} \vee \overline{t}) \wedge (\overline{x} \vee t) \wedge (x \vee \overline{y} \vee z)$

Question 3
Indiquer si F_1 est satisfiable ou non et, si elle est satisfiable, donner l'ensemble des solutions de F_1. 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.

Question 4
Déterminer une instance F_2 de **3-SAT** *non satisfiable et possédant exactement 8 clauses ; indiquer* $\max(F_2)$ *en justifiant la réponse.*
On considère une instance F de **3-SAT** définie sur n variables booléennes.
On Γ l'ensemble des clauses de F et V l'ensemble des 2^n valuations des variables de F.
Si $C \in \Gamma$ est une clause, on note $\varphi(C, \text{val}) = 1$ ou $\varphi(C, \text{val}) = 0$ selon que la valeur de C pour la valuation val est *vrai* ou *faux*.
On note $\psi(F, \text{val})$ le nombre de clauses de F qui valent *vrai* pour la valuation val.

On a $\psi(F, \text{val}) = \sum_{C \in \Gamma} \varphi(C, \text{val})$ et $\max(F) = \max\{\psi(F, \text{val}),\ \text{val} \in V\}$.

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

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

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

Indications pour la programmation

1. 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.
2. Pour coder une valuation d'un ensemble de n variables booléennes, on utilise un tableau de booléens de longueur au moins $n+1$. Pour i compris entre 1 et n, la case d'indice i donne la valeur de la variable de numéro i (`true` ou `false`).
   Les cases d'indice 0 ou $k > n$ ne sont pas utilisées.
3. Pour coder une clause de longueur p, on utilise un tableau d'au moins $p+1$ entiers.
   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 $(x \vee \overline{t} \vee y)$ peut être codée par `[|3; 1; -4; 2|]` ou `[|3; 1; -4; 2; 5|]` ou ...
4. Pour coder une formule logique, on utilise un tableau de tableaux d'entiers.
   Pour i supérieur ou égal à 1, la ligne d'indice i décrit la i-ième clause de la formule logique.
   On utilise deux cases de la ligne d'indice 0 : la case d'indices $(0,0)$ 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 $F_3 = (x \vee \overline{y} \vee z \vee t) \wedge (\overline{x} \vee \overline{z}) \wedge (x \vee \overline{t} \vee y)$ est peut être codée par

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

   On a écrit des tableaux minimaux pour les clauses.
5. Soit `F` un tableau de tableaux 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-ième clause de F.
   Dans un calcul de complexité, on considérera l'instruction `F.(i) <- F.(j)` comme une seule opération élémentaire.

2 Satisfiabilité, méthode exacte

Question 8
Écrire une fonction `valeur_clause : int array -> bool array -> bool`
telle que, si `C` *est un tableau codant codant une clause C et si* `val` *est un tableau codant une valuation* val*d'un ensemble de variables contenant les variables de C, alors* `valeur_clause C val` *donne la valeur (booléenne) de C pour la valuation* val.
Indiquer la complexité de la fonction `valeur_clause`.

Question 9
Écrire une fonction `satisfait_formule : int array array -> bool array -> bool`
telle que, si `F` *code une formule F et si* `val` *code une valuation* val*, alors* `satisfait_formule F val` *donne la valeur (booléenne) de F pour la valuation* val.
Indiquer la complexité de la fonction `satisfait_formule`.

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 un booléen v_i pour la variable i (si k vaut 0, aucune variable n'a une valeur fixée).
On cherche à déterminer s'il existe une valuation valdes variables telle que

1. pour i compris entre 1 et k, la valeur dans valde la variable i est v_i,
2. val satisfait F.

On dit, dans ce cas que la formule vérifie la propriété $\mathcal{P}(k)$ pour la valuation val.
On dispose d'un tableau de booléens destiné à coder une valuation des variables ; dans le cas où la valuation cherchée existe, on modifie les cases de ce tableau à partir de l'indice $k+1$ pour qu'il code une telle valuation et on renvoie alors la valeur *vrai* ; si la valuation cherchée n'existe pas, on renvoie la valeur *faux*.

Question 10
Écrire une fonction récursive

```
resoudre_rec : int array array -> bool array -> int -> bool
```

telle que, si `F` *code une formule,* `val` *code une valuation et k est un entier compris entre 0 et le nombre de variables de F, n, alors* `resoudre_rec F val k` *renvoie* `true` *ou* `false` *selon que la formule vérifie ou non la propriété $\mathcal{P}(k)$ pour la valuation* val*en modifiant si besoin les cases d'indice $k+1$ à n de* `val` *pour obtenir une valuation satisfiant F si la réponse est* `true`.

Question 11
Écrire une fonction `resoudre : int array array -> bool array` *telle que, si* `F` *code une formule logique F, alors* `resoudre F` *renvoie un tableau de booléen.*
Si F est satisfiable, la réponse doit coder une valuation satisfiant F et son premier terme doit contenir la valeur `true`*, si F n'est pas satisfiable, le premier terme de la réponse doit contenir la valeur* `false` *(les autres valeurs ne sont pas signifiantes).*

Question 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, l.*

3 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 α de F, on note $\text{diff}(F, \alpha)$ le nombre de fois où α figure dans F diminué du nombre de fois où $\overline{\alpha}$ figure dans F.

L'heuristique utilise la transformation suivante :

1. on calcule pour chaque variable α de F le nombre $\text{diff}(F, \alpha)$,
2. on détermine une variable α_0 de F telle que, pour toute variable α de F, on ait : $\big|\text{diff}(F, \alpha_0)\big| \geqslant \big|\text{diff}(F, \alpha)\big|$,
3. on donne la valeur *vrai* à α_0 si on a $\text{diff}(F, \alpha_0) > 0$ et la valeur *faux* sinon,
4. on supprime la variable α_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 α_0 ; on comptabilise le nombre de clauses satisfaites.

   On obtient ainsi une formule logique notée F' : 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.

Question 13
Appliquer l'heuristique à la formule F_1 définie avant la question 3 ; détailler les différentes étapes.

Question 14
Écrire une fonction `place : int array -> int -> int` *telle que, si* `C` *est un tableau codant une clause C et si* `litt` *est un littéral,* `place C litt` *renvoie 0 si* `litt` *ne figure pas dans C ou la position de* `litt` *dans C si* `litt` *figure dans C.*
Évaluer la complexité de la fonction `place`.

Question 15
Écrire une fonction `supprimer_variable : int array -> int -> unit` *telle que, si* `C` *est un tableau 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.*

Question 16
Écrire une fonction `supprimer_clause : int array array -> int -> unit` *telle que, si* `F` *est un tableau de tableaux 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 F ni de la valeur de i.*

Question 17
Écrire une fonction `calculer_diff : int array array -> int array` *telle que, si* `F` *est un tableau de tableaux codant une formule logique F, alors* `calculer_diff F` *renvoie un tableau d'entiers donnant pour chaque variable α de F la valeur de* $\text{diff}(F, \alpha)$ *décrite ci-dessus.*
En supposant que toutes les variables (dont le nombre est `F.(0).(1)`*) figurent effectivement dans F directement ou par leur complémenté, cette fonction doit avoir une complexité de l'ordre de la somme des longueurs des clauses de F.*

On s'intéresse maintenant à une transformation automatique d'une formule logique F lorsqu'on pose qu'une variable α prend la valeur v.
Si v vaut *vrai*, on note p le littéral α et sinon on note p le littéral $\overline{\alpha}$.
p prend donc la valeur *vrai* et son complémenté la valeur *faux*.
Les clauses contenant p, prenant la valeur 1, sont supprimées de F.
Pour chaque clause contenant le complémenté de p, on retire ce complémenté ;
si une clause était réduite au complémenté de p, on supprime une telle clause.

Les clauses qui ne contiennent ni p ni son complémenté sont inchangées.

Question 18
Écrire une fonction `simplifier : int array array -> int -> bool -> int` *telle que, si* `F` *est un tableau de tableaux d'entiers codant une formule logique F, si* `a` *est un entier représentant une variable α de F et si* `v` *est un booléen, alors* `simplifier F a v` *transforme* `F` *pour que* `F` *code la formule logique obtenue à partir de F selon la description donnée ci-dessus et renvoie le nombre de clauses de la formule logique F initiale qui contenaient le littéral p défini ci-dessus.*
Évaluer la complexité de la fonction `simplifier`.

Question 19
Écrire une fonction `heuristique : int array array -> int*bool array` *telle que, si* `F` *est un tableau de tableaux d'entiers codant une formule logique F, alors* `heuristique F` *renvoie le nombre de clauses satisfaites et la valuation qui les a satisfaites comme indiqués par l'heuristique.*
Évaluer la complexité de la fonction heuristique.

4 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 toutes les clauses sont de longueur au moins 2**. On appelle **formule logique 1-occ** une telle formule logique.
Par exemple F_4 est une formule 1-occ pour
$F_4 = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{y} \vee \overline{v}) \wedge (u \vee v) \wedge (\overline{t} \vee \overline{u})$.
On cherche à déterminer une solution d'une formule logique 1-occ.

On considère une formule logique 1-occ qui s'écrit : $F = (x \vee l_1 \vee l_2 \vee \cdots \vee l_k) \wedge (\overline{x} \vee l_{k+1} \vee \cdots \vee l_{k+h}) \wedge G$ où x est une variable, $l_1, \ldots, l_{k+h}$ sont des littéraux et G est une formule logique 1-occ éventuellement vide. On doit avoir $k \geqslant 1$ et $h \geqslant 1$.

Question 20
a) *Montrer que si $\{l_1, \ldots, l_{k+h}\}$ 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 G.
b) *Si $\{l_1, \ldots, l_{k+h}\}$ 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 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.

Question 21
Indiquer la formule logique obtenue en réduisant
a) $(x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{y} \vee \overline{t})$ *par rapport à x,*
b) $(\overline{x} \vee \overline{z} \vee t) \wedge (\overline{t} \vee \overline{u}) \wedge (z \vee u)$ *par rapport à t.*

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

Question 23
Décrire sans utiliser le langage de programmation une fonction récursive `calculer_solution` *qui prend en paramètre une formule logique 1-occ F et qui renvoie une solution de F.*

Question 24
Appliquer la fonction précédente à la formule F_4. Détailler chaque appel récursif.

5 Éléments de solutions

Solution de la question 1 -

a) $x \vee y \vee z$

b) $\overline{x} \vee \overline{y} \vee \overline{z}$

c) $x \Rightarrow (\overline{y} \wedge \overline{z})$ équivalent à $\overline{x} \vee (\overline{y} \wedge \overline{z})$ donc à $(\overline{x} \vee \overline{y}) \wedge (\overline{x} \vee \overline{z})$.

d) $y \Rightarrow (x \vee z)$ équivalent à $\overline{y} \vee x \vee z$.

Solution de la question 2 -
On cherche les solutions de $F = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z}) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{x} \vee \overline{z}) \wedge (\overline{y} \vee x \vee z)$.
La table de vérité est

x	y	z	$x \vee y \vee z$	$\overline{x} \vee \overline{y} \vee \overline{z}$	$\overline{x} \vee \overline{y}$	$\overline{x} \vee \overline{z}$	$\overline{y} \vee x \vee z$	F
0	0	0	0	1	1	1	1	0
0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1
1	0	1	1	1	1	0	1	0
1	1	0	1	1	0	1	1	0
1	1	1	1	0	0	0	1	0

F est satisfiable pour les valuation $(0,0,1)$, $(0,1,1)$ et $(1,0,0)$.

Solution de la question 3 - On note $c_1 = x \vee y \vee z$, $c_2 = \overline{x} \vee z \vee \overline{t}$, $c_3 = x \vee \overline{y} \vee \overline{z}$, $c_4 = x \vee y \vee \overline{t}$, $c_5 = \overline{x} \vee \overline{z} \vee \overline{t}$, $c_6 = \overline{x} \vee t$ et $c_7 = x \vee \overline{y} \vee z$.
On a $c_2 \wedge c_5 \equiv \overline{x} \vee (z \wedge \overline{z}) \vee \overline{t} \equiv \overline{x} \vee \overline{t}$ d'où $c_2 \wedge c_5 \wedge c_6 \equiv \overline{x} \vee (\overline{t} \wedge t) \equiv \overline{x}$.
De même $c_3 \wedge c_7 \equiv x \vee \overline{y} \vee (\overline{z} \wedge z) \equiv x \vee \overline{y}$ donc
$c_2 \wedge c_5 \wedge c_6 \wedge c_3 \wedge c_7 \equiv \overline{x} \wedge (x \vee \overline{y}) \equiv \overline{x} \wedge x \vee \overline{x} \wedge \overline{y} \equiv \overline{x} \wedge \overline{y}$.
On a alors $\overline{x} \wedge \overline{y} \wedge c_1 \equiv \overline{x} \wedge \overline{y} \wedge x \vee \overline{x} \wedge \overline{y} \wedge y \vee \overline{x} \wedge \overline{y} \wedge z \equiv \overline{x} \wedge \overline{y} \wedge z$.
De même $\overline{x} \wedge \overline{y} \wedge c_4 \equiv \overline{x} \wedge \overline{y} \wedge \overline{t}$ donc $F1 \equiv \overline{x} \wedge \overline{y} \wedge z \wedge \overline{t}$.
F_1 est satisfiable pour la seule valuation $(0,0,1,0)$.

Solution de la question 4 -
Si on développe $(x \wedge \overline{x}) \vee (y \wedge \overline{y}) \vee (z \wedge \overline{z})$, qui n'est pas satisfiable, on obtient une conjonction de 8 clauses de longueur 3 : on a donc une instance de 3-sat, $F2$, à 8 clauses et non satisfiable.
Si on retire la clause $x \vee y \vee z$ on obtient une formule logique satisfiable pour la valuation $(0,0,0)$ donc $m(F2) = 7$.

Solution de la question 5 -
Une clause de 3 variables est satisfaite pour 7 valuations sur les 8 qui concernent les 3 variables de la clauses. Comme les valeurs des $n-3$ autres variables sont indifférentes on en déduit que la clause est satisfaite pour 7.2^{n-3} valuations donc $\sum_{\text{val} \in V} \varphi(C, \text{val}) = 7.2^{n-3}$.

Solution de la question 6 -
Comme m est le nombre de clauses on a $\sum_{C \in \Gamma} \sum_{\text{val} \in V} \varphi(C, \text{val}) = m.7.2^{n-3}$.
On a donc $m.7.2^{n-3} = \sum_{\text{val} \in V} \sum_{C \in \Gamma} \varphi(C, \text{val}) = \sum_{\text{val} \in V} \psi(F, \text{val}) \leqslant \sum_{\text{val} \in V} \max(F) = 2^n \max(F)$.
On en déduit qu'on a $\max(F) \geqslant m\frac{7}{8}$.

Solution de la question 7 -
Pour que F soit non satisfiable on doit avoir $\max(F) < m$ donc $\max(F) \leqslant m-1$.
D'après le résultat ci-dessus on doit donc avoir $m\dfrac{7}{8} \leqslant m-1$ d'où $m \geqslant 8$.
Les instances de **3-SAT** non satisfiables ont au moins 8 clauses.

Solution de la question 8 - On commence par la valeur d'un littéral.

```
let valeur_litteral litt valeur =
  if litt > 0 then valeur.(x) else not valeur.(x);;
```

Il reste alors à faire la disjonction des valeurs des littéraux

```
let valeur_clause clause valeur =
  let s = ref false in
  for i = 1 to clause.(0) do
    s := !s || (valeur_litteral clause.(i) valeur) done;
  !s;;
```

La complexité est proportionnelle à la longueur de la clause.

Solution de la question 9 - On fait la disjonction des valeurs des clauses.
On renvoie le booléen d'égalité du produit à 1.

```
let satisfait_formule formule valeur =
  let prod = ref true in
  for i = 1 to formule.(0).(0) do
    prod := !prod && (valeur_clause formule.(i) valeur) done;
  !prod;;
```

La complexité est proportionnelle au nombre total de littéraux dans l'écriture de la formule.

Solution de la question 10 - Le cas d'arrêt est celui où $k = n$, le nombre de variables.

```
let rec resoudre_rec formule valeur k =
  let n = formule.(0).(1) in
  if k = n
  then satisfait_formule formule valeur
  else   (valeur.(k+1) <- false;
          resoudre_rec formule valeur (k+1))
       ||(valeur.(k+1) <- true;
          resoudre_rec formule valeur (k+1));;
```

Comme l'évaluation du `||` n'effectue pas la seconde évaluation si la première évaluation renvoie vraie , la valuation transformée correspond bien à une éventuelle solution.

Solution de la question 11 - On appelle bien entendu la fonction précédente après avoir construit une valuation.

```
let resoudre formule =
  let n = formule.(0).(1) in
  let valeur = Array.make (n+1) false in
  if resoudre_rec formule valeur 0 then valeur.(0) <- true;
  valeur;;
```

Solution de la question 12 - Dans le cas d'une formule non satisfiable on essaie toutes les valuations donc la complexité est un $\mathcal{O}(l2^n)$.

Solution de la question 13 -
On notera que la formule transformée n'est pas équivalente à la formule initiale.

- On calcule $\text{diff}(F_1, x) = 1$, $\text{diff}(F_1, y) = 0$, $\text{diff}(F_1, z) = 1$, $\text{diff}(F_1, t) = -2$.
  On choisit $t = 0$ et on a satisfait 3 clauses. On aboutit à la formule
  $F_1' = (x \vee y \vee z) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge \overline{x} \wedge (x \vee \overline{y} \vee z)$.
- On calcule $\text{diff}(F_1', x) = 2$, $\text{diff}(F_1', y) = -1$, $\text{diff}(F_1', z) = 1$.
  On choisit $x = 1$ et on a satisfait 3 clauses. On aboutit à la formule vide.
- On a ainsi satisfait 6 clauses avec les valuations $(1, y, z, 0)$.

Solution de la question 14 - On parcourt simplement le tableau des littéraux.

```
let place c litt =
  let n = c.(0) in
  let p = ref 0 in
  for i = 1 to n do
    if c.(i) = litt
    then p := i done;
  !p;;
```

La complexité est proportionnelle à l, le nombre de littéraux de c.

Solution de la question 15 - Pour que la complexité soit constante il suffit de diminuer le nombre de littéraux et de placer le dernier à la place de celui que l'on retire.

```
let supprimer_variable clause i =
  let n = clause.(0) in
  clause.(i) <- clause.(n);
  clause.(0) <- (n-1);;
```

Solution de la question 16 - On emploie la même technique. Bien que ce ne soit pas explicitement décrit dans l'énoncé, le tableau peut avoir des lignes inutilisées.

```
let supprimer_clause formule clause =
  let n = formule.(0).(0) in
  formule.(clause) <- formule.(n);
  formule.(0).(0) <- (n-1);;
```

Solution de la question 17 - On va inscrire $\text{diff}(F, x)$ à la position x donc on crée un tableau de longueur $p + 1$ s'il y a p variables.

```
let calculer_diff formule  =
  let nbCl = formule.(0).(0) in
  let nbVar = formule.(0).(1) in
  let diff = Array.make (nbVar +1) 0 in
  for clause = 1 to nbCl do
    for i = 1 to formule.(clause).(0) do
      let litt = formule.(clause).(i) in
      if litt > 0
      then diff.(litt) <- diff.(litt) +1
      else diff.(-litt) <- diff.(-litt) - 1 done done;
  diff;;
```

On effectue un nombre borné d'instructions pour chaque littéral donc la complexité est de l'ordre de la somme des longueurs des clauses.

Solution de la question 18 - Quand on supprime une clause elle est remplacée par la dernière, il faut donc recommencer l'étude au même indice de clause. On ne peut donc pas utiliser une boucle `for` : il faut employer un boucle conditionnelle avec `while`.

```
1  let simplifier formule a v =
2    let litt = if v then a else -a in
3    let supp = ref 0 in
4    let clause = ref 1 in
5    while !clause <= formule.(0).(0) do
6      if place formule.(!clause) litt > 0
7      then begin
8           supprimer_clause formule !clause;
9           supp := !supp + 1 end
10     else begin
11          let j = place formule.(!clause) (-litt) in
12          if j > 0
13          then begin
14               supprimer_variable formule.(!clause) j;
15               if formule.(!clause).(0) = 0
16               then supprimer_clause formule !clause
17               else clause := !clause +1 end
18          else clause := !clause +1 end done;
19   !supp;;
```

- Ligne 2 : on choisit le littéral vérifié par la valuation
- Ligne 3 : on initialise le nombre de clauses satisfaites
- Lignes 4&5 : on parcourt les clauses. On peut noter que si on supprime la dernière clause alors `f.(0).(0)` a diminué et on sort de la boucle.
- Lignes 6 à 9 : si la clause contient le littéral, on la supprime et on incrémente le nombre de clauses satisfaites. On n'incrémente pas la clause car elle est remplacée.
- Lignes 10&11 : on teste si la clause contient la négation du littéral.
- Lignes 12 à 17 : si la clause contient la négation du littéral en j.
  - On supprime le littéral (ligne 14) ;
  - on teste si la clause est vide (ligne 15),
  - si oui, on la supprime (ligne 16) et on reste à cet indice de clause car la clause est remplacée ; si c'était la dernière clause, le test de la boucle `while` donnera `false`,
  - sinon, on incrémente.

  .
- Ligne 18 : si la clause ne contient ni le littéral ni sa négation du littéral on passe à la clause suivante.

Ici encore la complexité est de l'ordre de la somme des longueurs des clauses.

Solution de la question 19 - On commence par écrire la recherche de l'indice d'un maximum dans un tableau pour chercher la variable dont le `diff` est maximal. Cependant on doit choisir une variable qui n'a pas encore été affectée d'où l'usage d'un tableau d'indices libres

```
let maxLibre diff libres =
  let n = Array.length diff in
  let kMax = ref 1 in
  for k = 2 to n-1 do
    if abs(diff.(k)) > abs(diff.(!kMax)) && libre.(k)
    then kMax := var; libre.(k) <- false done;
  !kMax;;
```

On suit ensuite l'algorithme de l'énoncé en gérant le tableau des variable non encore affectées.

```
let heuristique formule =
  let nbVar = formule.(0).(1) in
  let valeur = Array.make (nbVar+1) false in
  let libres = Array.make (nbVar+1) true in
  let nbSat = ref 0 in
  valeur.(0) <- 0;
  while formule.(0).(0) > 0 do
    let diff = calculer_diff formule in
    let a = maxLibre diff libres in
    valeur.(a) <- (diff.(a) > 0);
    nbSat := !nbSat + (simplifier formule a valeur.(a)) done;
  valeur;;
```

Le nombre d'étape dans la boucle `while` est majoré par le nombre de clauses donc la complexité est un $\mathcal{O}(ml)$ où m est le nombre de clauses et l le nombres de littéraux dans les clauses.

Solution de la question 20 - a) On note y la variable telle que y et $\overline{y}$ apparaissent dans $l_1, \ldots, l_k, l_{k+1}, \ldots, lk+h$. Ces deux littéraux ne peuvent pas appartenir à la même clause donc, si y appartient à $y\{\{l_1, \ldots, l_k\}$, on doit avoir $\overline{y} \in \{l_{k+1}, \ldots, l_{k+h}\}$ et réciproquement. On peut supposer sans perte de généralité que $\{y, \overline{y}\} = \{l_1, l_{k+1}\}$.
Ni x ni y n'apparaissent dans G. On a deux cas :

- $F = (x \vee y \vee l_2 \cdots l_k) \wedge (\overline{x} \vee \overline{y} \vee l_{k+2} \cdots l_{k+h}) \wedge G$: dans ce cas toute solution de G complétée par $x = vrai$ et $y = faux$ est une solution de F.
- $F = (x \vee \overline{y} \vee l_2 \cdots l_k) \wedge (\overline{x} \vee y \vee l_{k+2} \cdots l_{k+h}) \wedge G$: dans ce cas toute solution de G complétée par $x = vrai$ et $y = faux$ est une solution de F.

Comme toute solution de F définit une solution de G on en déduit que F est satisfiable si et seulement si G est satisfiable.

b) Si les variables associées aux l_i sont distinctes on définit
$F' = (l_1 \vee l_2 \cdots l_k \vee l_{k+1} \vee l_{k+2} \cdots l_{k+h}) \wedge G$.
Les définitions indiquent que ni x ni $\overline{x}$ n'apparaît dans F' et que F' est 1-occ.

- Si v est une valuation qui satisfait F alors elle satisfait G. De plus,
  si $v(x) = 0$, la satisfaction de $x \vee l_1 \vee l_2 \cdots l_k$ impose la satisfaction de $l_1 \vee l_2 \cdots l_k$
  si $v(x) = 1$, la satisfaction de $\overline{x} \vee l_{k+1} \vee \cdots \vee l_{k+h}$ impose celle de $l_{k+1} \vee \cdots \vee l_{k+h}$.
  Dans les deux cas $l_1 \vee l_2 \cdots l_k \vee l_{k+1} \vee l_{k+2} \cdots l_{k+h}$ donc F' est satisfaite.
- Si v est une valuation qui satisfait F' alors elle satisfait G. De plus,
  - soit $l_1 \vee l_2 \cdots l_k$ est satisfaite donc, en posant $v(x) = 0$,
    $(x \vee \overline{y} \vee l_2 \cdots l_k) \wedge (\overline{x} \vee y \vee l_{k+2} \cdots l_{k+h})$ est satisfaite
  - soit $l_{k+1} \vee l_{k+2} \cdots l_{k+h}$ est satisfaite donc, en posant $v(x) = 1$,
    $(x \vee \overline{y} \vee l_2 \cdots l_k) \wedge (\overline{x} \vee y \vee l_{k+2} \cdots l_{k+h})$ est satisfaite.

  On a ainsi déterminé une valuation qui satisfait F.

F est satisfiable si et seulement si F' est satisfiable.

Solution de la question 21 -
a) $(x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{y} \vee \overline{t})$ se réduit par rapport à x en $\overline{y} \vee \overline{t}$
b) $(\overline{x} \vee \overline{z} \vee t) \wedge (\overline{t} \vee \overline{u}) \wedge (z \vee u)$ se réduit par rapport à t en $(\overline{x} \vee \overline{z} \vee \overline{u}) \wedge (z \vee u)$

Solution de la question 22 -
Les réductions permettent d'éliminer les variables qui apparaissent dans deux clauses.
Il reste à prouver qu'une formule où les variables n'apparaissent qu'une fois est satisfiable : il suffit de donner la valeur 1 aux variables apparaissant directement et la valeur 0 sinon.
Le cas d'une formule devenue vide est trivial.

Solution de la question 23 -
Si une formule est vide elle est satisfaite pour toute valuation : c'est le cas d'arrêt.
Si une formule 1-occ, F, admet plusieurs variables on considère la variable de plus haut indice, ζ.

- Si ζ n'apparaît qu'une fois on lui donne la valeur qui rend le littéral associé vrai, on retire la clause la contenant et on obtient une formule 1-occ avec moins de variables : F'. On construit une valuation en complétant une valuation qui satisfait F' obtenue récursivement, par la valeur de ζ calculée.
- Si ζ apparaît deux fois et qu'on effectue une réduction du type **20 a)** on calcule récursivement une valuation de G et on attribue la valeur de ζ adaptée.
- De même si ζ apparaît deux fois et qu'on effectue une réduction du type **20 b)**.

Solution de la question 24 -

- F_4 est satisfiable si et seulement si sa réduite par rapport à v, F_5, est statisfiable.
  $F_5 = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{t} \vee \overline{u}) \wedge (\overline{y} \vee u)$.
- On réduit F_5 par rapport à u : $F_6 = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{t} \vee \overline{y})$.
- On réduit F_6 par rapport à t : $F_7 = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$.
- On réduit F_7 par rapport à z, associé à y, on aboutit à la formule vide.
- On attribue une valeur quelconque à x, par exemple 0.
- Pour satisfaire F_7 on choisit $y = 1$, $z = 0$.
- Dans F_6 on choisit $t = 0$ pour satisfaire $\overline{t} \vee \overline{y}$.
- Dans F_5 on choisit $u = 1$ pour satisfaire $\overline{y} \vee u$.
- Dans F_4 on choisit $v = 0$ pour satisfaire $\overline{y} \vee \overline{v}$.
- On aboutit à `valeur = [|-1; 0; 1; 0; 0; 1; 0|]`.