Devoir d'Informatique n°2
Concours blanc

14 juin 2025

*
* *

Durée de l'épreuve :
4 heures (non tiers-temps)
5h20 heures (tiers-temps)

L'usage de tout dispositif électronique est interdit.

Consignes

Pour répondre à une question, il vous est permis de réutiliser le résultat d'une question antérieure même si vous n'avez pas réussi à établir ce résultat.

Quand l'énoncé demande de coder une fonction, sauf demande explicite, il n'est pas nécessaire de justifier la correction ou la terminaison de cette fonction, ou de la commenter. Néanmoins, il ne faut pas hésiter à formuler les commentaires qui semblent pertinents.

Vous attacherez la plus grande importance à la clarté, à la précision et à la concision de la rédaction.

Si vous repérez ce qui vous semble être une erreur d'énoncé, signalez-le sur votre copie et poursuivez la composition en expliquant les éventuelles initiatives que vous aurez pris.

Le problème de la satisfiabilité logique (Mines 2010)

Préliminaires concernant la programmation : il faudra écrire des fonctions à l'aide du langage OCaml. Par ailleurs, pour écrire une fonction en langage de programmation, vous pourrez définir des fonctions ou des procédures auxiliaires que vous expliciterez, ou faire appel à d'autres fonctions 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 $\neg x$ la négation de x : $\neg x$ vaut 1 si x vaut 0 et $\neg x$ vaut 0 si x vaut 1.

On appelle *littéral* une variable booléenne ou son complémenté. Pour un littéral $\alpha = \neg x$, où x est une variable booléenne, on définit le complémenté $\bar{\alpha}$ de α 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 α, elle ne contient pas une deuxième fois α et elle ne contient $\bar{\alpha}$. 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 :

$$(x \vee y \vee z) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z)$$

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.

Q1. 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.

Q2. 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 dépendant de quatre variables x, y, z et t :

$$F_1 = (x \vee y \vee z) \wedge (\neg x \vee z \vee \neg t) \wedge (x \vee \neg y \vee \neg z) \wedge (x \vee y \vee \neg t) \wedge (\neg x \vee \neg z \vee t) \wedge (\neg x \vee t) \wedge (x \vee \neg y \vee z) \wedge$$

Q3. 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.

Q4. 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 note V l'ensemble des 2^n valuations des variables de F. Soit σ une valuation des n variables ; si C est une clause, on note $\varphi(C, \sigma)$ la valeur de C pour la valuation σ ; on note $\Psi(F, \sigma)$ le nombre de clauses de F qui valent 1 pour la valuation σ :

$$\Psi(F, \sigma) = \sum_{C \text{ clause de } F} \phi(C, \sigma)$$

On a $\max(F) = \max_{\sigma \in V} \Psi(F, \sigma)$.

Q5. On considère une clause C de F ; indiquer en fonction de n la valeur de la somme :

$$\sum_{\sigma \in V} \varphi(C, \sigma)$$

Q6. En considérant la somme $\displaystyle\sum_{C \text{ clause de } F} \sum_{\sigma \in V} \varphi(C, \sigma)$, indiquer en fonction de m un minorant de $\max(F)$.

Q7. 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 $\neg x$, on associe le numéro -1, au littéral $\neg y$, on associe le numéro -2, au littéral $\neg z$, on associe le numéro -3. **Pour alléger les explications, on identifie désormais un littéral avec son numéro**.

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 $(x \vee \neg t \vee y)$ 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-ème 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 :

$$F_3 = (x \vee \neg y \vee z \vee t)(\neg x \vee \neg z)(x \vee \neg t \vee y)$$

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	...

En OCaml, la formule logique F_3 peut être définie comme ci-dessous par le tableau de tableaux `F3` :

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

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-ème clause de F.

Deuxième partie : satisfiabilité, méthode exacte

Q8. Écrire en OCaml une fonction `valeur_clause` telle que, si `C` est un tableau d'entiers codant une clause C et si `val` est un tableau codant une valuation σ, alors `valeur_clause C val` renvoie $\varphi(C, \sigma)$.
Indiquer la complexité de la fonction `valeur_clause`.

Q9. Écrire en OCaml une fonction `satisfait_formule` telle que, si `F` est un tableau de tableaux d'entiers codant une formule F et si `val` est un tableau codant une valuation σ, alors `satisfait_formule F val` renvoie `true` si σ satisfait F, `false` sinon.
Indiquer la complexité de la fonction `satisfait_formule`.

Q10. 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 $v_i \in \{0, 1\}$ pour la variable i (si $k = 0$, aucune variable n'a de valeur fixée). La fonction `resoudre_rec` détermine s'il existe une valuation σ des variables telle que :

— pour i compris entre 1 et k, $\sigma(i) = v_i$;
— σ satisfait F ;

autrement dit, s'il est possible d'**étendre** la définition de σ pour obtenir une valuation satisfiant F.
De plus, la fonction `resoudre_rec` dispose d'un tableau d'entiers destinés 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.
Écrire en OCaml une fonction récursive `resoudre_rec` telle que :

— si `F` est un tableau de tableaux d'entiers codant une formule F,
— si `val` est un vecteur d'entiers servant à coder une valuation partielle,
— 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 $1, 2, \ldots$ `k`,

alors `resoudre_rec F val k` modifie le vecteur `val` comme indiqué ci-dessus, en considérant que les valeurs des cases d'indices $1, 2, \ldots,$ `k` sont les valeurs fixées $v_1, \ldots v_k$ des variables $1, 2, \ldots k$; la fonction revoie `true` si la valuation cherchée existe, `false` sinon. *Indication : la fonction pourra essayer de fixer $v_{k+1} = 0$, d'essayer de résoudre récursivement, puis de fixer $v_{k+1} = 0$ si le premier essai n'a pas abouti.*

Q11. Écrire en OCaml une fonction `resoudre` telle que, si `F` est un tableau de tableaux d'entiers codant une formule F, alors `resoudre F` renvoie un vecteur d'entiers codant une solution de F, sous la forme d'une option. Si F n'est pas satisfiable, la fonction renverra `None`.

Q12. Évaluer la complexité de la fonction `resoudre` appliquée à une formule logique F en fonction du nombre n de variables 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 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, que l'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 ; étant donné une variable x de F, on note $D(F, x)$ le nombre de fois où x apparaît dans F, moins le nombre de fois où $\neg x$ apparaît dans F ; l'heuristique utilise la transformation suivante :

a) On calcule pour chaque variable x de F le nombre $D(F, x)$.

b) On détermine une variable x_0 pour laquelle $|D(F, x)|$ est maximal.

c) On pose $x_0 = 1$ si $D(F, x_0) > 0$, et $x_0 = 0$ sinon

d) On supprime la variable x_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 x_0 ; on comptabilise le nombre de clauses satisfaites. On obtient ainsi une formule logique 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.

Q13. Appliquer l'heuristique à la formule F_1 définie avant la question 3 ; détailler les différentes étapes.

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

Q14. Écrire en OCaml une fonction `place` telle que, si `C` est un vecteur codant une clause C et si `x` est une variable (i.e. un entier entre 1 et le nombre total de variables), `place C x` renvoie :

— 0 si `x` ne figure pas dans `C`

— La position de `x` dans `C` sinon

Q15. Écrire en OCaml une fonction `supprimer_variable` 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. Cette fonction devra être de complexité $\mathcal{O}(1)$. On rappelle qu'une clause doit être représentée par un tableau sans trous.

Q16. Écrire en OCaml une fonction `supprimer_clause` telle que, si `F` est un tableau de tableaux d'entiers codant une formule 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 obtenue en supprimant de F la clause d'indice i. Cette fonction devra être de complexité $\mathcal{O}(1)$.

Q17. Écrire en OCaml une fonction `calculer_diff` telle que, si `F` est un tableau de tableaux d'entiers codant une formule logique F, alors `calculer_diff F` renvoie un vecteur d'entiers donnant pour chaque variable x de F la valeur de $D(F, x)$ 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.

Q18. On s'intéresse dans cette question à une transformation automatique d'une formule logique F lorsqu'on pose qu'une variable x prend la valeur v. Cette transformation est effectuée à l'aide d'une fonction nommée `simplifier` prenant F, x et v comme arguments. Si v vaut 1, on note ℓ le littéral x et sinon on note ℓ le littéral $\neg x$. Quand x prend la valeur v, ℓ prend la valeur 1 et le complémenté de ℓ prend la valeur 0. Les clauses contenant ℓ, prenant la valeur 1, sont supprimées de F. Pour chaque clause contenant le complémenté de ℓ, on retire ce complémenté ; si une clause devient alors vide, on la supprime. Les clauses qui ne contiennent ni ℓ ni son complémenté sont inchangées. La fonction `simplifier` compte le nombre de clauses qui contenaient ℓ avant simplification et renvoie ce nombre.
Écrire en OCaml la fonction `simplifier` telle que, si `F` est un tableau de tableaux d'entiers codant une formule logique F, si x est un entier représentant une variable x 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 simplification décrite ci-dessus et renvoie le nombre de clauses de la formule logique F initiale qui contenaient ℓ défini ci-dessus.
Évaluer la complexité de la fonction `simplifier`.

Q19. Écrire en OCaml une fonction heuristique telle que, si `F` est un tableau de tableaux d'entiers codant une formule logique F, alors `heuristique F` renvoie un tableau 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`.

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 F_4 ci-dessous, ayant six variables x, y, z, t, u, v, est une formule logique 1-occ :

$$F_4 = (x \vee y \vee z) \wedge (\neg x \vee \neg z \vee t) \wedge (\neg y \vee \neg v) \wedge (u \vee v) \wedge (\neg t \vee \neg u)$$

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

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

$$F = (x \vee \ell_1 \vee \ell_2 \cdots \vee \ell_k) \wedge (\neg x \vee \ell_{k+1} \vee \ell_{k+2} \cdots \vee \ell_{k+h})$$

avec $k \geq 1$, $h \geq 1$, x une variable, $\ell_1 \ldots \ell_{k+h}$ des littéraux, et G une formule 1-occ éventuellement vide.

Q20. Montrer que si $\{\ell_1, \ell_2 \ldots ell_{k+h}\}$ contient à la fois une variable et son complémenté, alors F est satisfiable si et seulement si G l'est. On dira dans ce cas que F **réduite** par rapport à x donne la formule G

Q21. On suppose maintenant que $\{\ell_1, \ell_2 \dots ell_{k+h}\}$ ne contient jamais à la fois une variable et son complémenté. Indiquer une formule logique 1-occ F' ne contenant ni x ni $\neg x$ telle que F est satisfiable si et seulement si F' l'est. On dira dans ce cas que F **réduite** par rapport à x donne la formule F'. *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.*

Q22. On considère la formule logique

$$(x \vee y \vee z) \wedge (\neg x \vee \neg z \vee t) \wedge (\neg y \vee \neg t)$$

Indiquer la formule obtenue en réduisant celle-ci par rapport à x.

Q23. On considère la formule logique

$$(\neg x \vee \neg z \vee t) \wedge (\neg t \vee \neg u) \wedge (z \vee u)$$

Indiquer la formule obtenue en réduisant celle-ci par rapport à t.

Q24. Montrer que toute formule 1-occ est satisfiable.

Q25. Décrire sans utiliser de langage de programmation un algorithme prenant en entrée une formule 1-occ, et permettant d'en construire une solution.

Q26. Appliquer l'algorithme de la question précédente à la formule F_4. Détailler chaque appel récursif.