Option informatique — Mines-Ponts 2010

Partie I — Introduction

Question I.1
La proposition a) s'énonce : $x \vee y \vee z$; la proposition b) s'énonce : $\bar{x} \vee \bar{y} \vee \bar{z}$; la proposition c) s'énonce : $x \Rightarrow (\bar{y} \wedge \bar{z})$ ce qui est équivalent à $(\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee \bar{z})$; la proposition d) s'énonce : $y \Rightarrow (x \vee z)$ ce qui est équivalent à $x \vee \bar{y} \vee z$.

Question I.2
La formule logique demandée est donc $F_0 = (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee \bar{y}) \wedge (\bar{x} \vee \bar{z}) \wedge (x \vee \bar{y} \vee z)$.
On peut dresser une table de vérité pour déterminer l'ensemble des solutions de cette formule logique.

x	y	z	F_0
1	1	1	0
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	0

Tableau 1 une table de vérité

F_0 est satisfiable. Ses solutions sont les triplets $(x, y, z) = (1, 0, 0)$ ou $(0, 1, 1)$ ou $(0, 0, 1)$.

Question I.3
On peut de même dresser une table de vérité, pour vérifier que F_1 est satisfiable. Elle n'a qu'une solution : $(x, y, z, t) = (0, 0, 1, 0)$.

Question I.4
La clause $x \vee y \vee z$ n'est pas satisfaite quand $(x, y, z) = (0, 0, 0)$, la clause $x \vee y \vee \bar{z}$ n'est pas satisfaite quand $(x, y, z) = (0, 0, 1)$, etc. On en déduit que la clause

$$F_2 = (x \vee y \vee z) \wedge (x \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

n'est pas satisfiable.
En revanche $\max(F_2) = 7$ car la formule F_2 privée de sa dernière clause est satisfaite par $(x, y, z) = (1, 1, 1)$.

Question I.5
Une clause C est satisfaite par toute valuation sauf celles qui donne un triplet de valeurs booléennes particulier aux trois variables qui apparaissent dans C : donc $\displaystyle\sum_{val \in V} \varphi(C, val) = 2^n - 2^{n-3} = \frac{7}{8} 2^n$.

Question I.6
On écrit

$$\sum_{C \text{ clause de } F} \sum_{val \in V} \varphi(C, val) = \frac{7m}{8} 2^n = \sum_{val \in V} \psi(F, val) \leqslant 2^n \max(F),$$

et on en déduit que $\max(F) \geqslant \dfrac{7m}{8}$ donc, comme on travaille en entiers : $\max(F) \geqslant \left\lceil \dfrac{7m}{8} \right\rceil$.

Question I.7

Dire que F est non satisfiable signifie que $\max(F) \leqslant m - 1$. Alors $\left\lceil \dfrac{7m}{8} \right\rceil \leqslant m - 1$ donc $m \geqslant 8$.
Une instance de 3-SAT non satisfiable possède donc au moins 8 clauses.

Partie II — Satisfiabilité, méthode exacte

Question II.8
On écrit

```
1  let valeur_littéral x val =
2    if x > 0 then val.(x)
3    else 1 - val.(-x) ;;

4  let valeur_clause C val =
5    let p = C.(0) in
6    let i = ref 1 and s = ref 0 in
7      while !i <= p do s := !s + (valeur_littéral C.(!i) val) ; incr i done ;
8      if !s > 0 then 1 else 0 ;;
```

Programme 1 la fonction `valeur_clause`

On a écrit une fonction auxiliaire de calcul de la valeur d'un littéral pour une valuation donnée.
La variable `p` compte le nombre de littéraux de la clause. L'indice `!i` permet de balayer la clause et `!s` permet de compter les littéraux satifaits. La clause est satisfaite si en fin de boucle `!s > 0`.
La complexité de la fonction est clairement $O(p)$ où p est le nombre de littéraux qui apparaissent dans la clause.

Question II.9
Il suffit d'appeler la fonction précédente à chaque clause de la formule. On obtient :

```
9  let satisfait_formule F val =
10   let m = F.(0).(0) in
11   let i = ref 1 in
12     while !i <= m && valeur_clause F.(!i) val = 1 do incr i done ;
13     !i = m + 1 ;;
```

Programme 2 la fonction `satisfait_formule`

Cette fois, `m` désigne le nombre de clauses de la formules, l'indice `!i` permet de balayer le tableau de clauses, tant qu'elles sont satisfaites. La formule est satisfaite si la boucle `while` a décrit l'ensemble du tableau.
La complexité est majorée par $O(\ell)$ où ℓ est la somme des longueurs des clauses de F.

Question II.10
Le plus simple est d'écrire une fonction récursive, comme le nom de la fonction le laissait supposer !

```
14 let rec resoudre_rec F val k =
15   let n = F.(0).(1) in
16     if k = n then satisfait_formule F val
17     else
18       (val.(k+1) <- 0 ; resoudre_rec F val (k+1))
19       || (val.(k+1) <- 1 ; resoudre_rec F val (k+1)) ;;
```

Programme 3 la fonction `resoudre_rec`

Question II.11
Il suffit d'appeler la fonction précédente en partant de $k = 0$.

```
20  let resoudre F =
21    let n = F.(0).(1) in
22    let val = make_vect (n+1) 0 in
23      val.(0) <- if resoudre_rec F val 0 then 1 else 0 ;
24      val ;;
```

Programme 4 la fonction `resoudre`

Question II.12
La complexité demandée est $O(2^n \ell)$, où ℓ est la somme des longueurs des clauses de F.

Partie III — MAX-SAT

Question III.13
$\text{diff}(F1, x) = 1$, $\text{diff}(F1, y) = 0$, $\text{diff}(F1, z) = 1$ et $\text{diff}(F1, t) = -2$. On choisit donc de poser $t = 0$ satisfaisant ainsi 3 clauses de $F1$.
La formule $F1$ devient $F'1 = (x \vee y \vee z) \wedge (x \vee \bar{y} \vee \bar{z}) \wedge \bar{x} \wedge (x \vee \bar{y} \vee z)$. Alors $\text{diff}(F'1, x) = 2$, $\text{diff}(F'1, y) = -1$ et $\text{diff}(F'1, z) = 1$. On choisit donc de poser $x = 1$, ce qui permet de satisfaire 3 nouvelles clauses. On élimine la clause insatisfiable $\bar{x}$. Il n'y a plus de clause : c'est terminé.
Ainsi l'heuristique n'a réussi à satisfaire que 6 clauses sur les 7 de $F1$, avec une valuation $x = 1$ et $t = 0$.

Question III.14
La fonction `place` se contente d'examiner la clause, sa complexité est $O(p)$ où p est la longueur de la clause.

```
25  let place C litt =
26    let p = C.(0) in
27    let i = ref 1 in
28    while !i <= p && C.(!i) <> litt do incr i done ;
29    if !i <= p then !i else 0 ;;
```

Programme 5 la fonction `place`

Question III.15
Il suffit d'échanger le littéral en position i avec le dernier littéral de la clause et de décrémenter le nombre de littéraux de C : cela garantira une complexité en $O(1)$.

```
30  let supprimer_variable C i =
31    let p = C.(0) in
32      C.(i) <- C.(p) ;
33      C.(0) <- p-1 ;;
```

Programme 6 la fonction `supprimer_variable`

Question III.16
Même méthode ici, pour écrire le programme 7, page 4.

```
34  let supprimer_clause F i =
35    let m = F.(0).(0) in
36      F.(i) <- F.(m) ;
37      F.(0).(0) <- m-1 ;;
```

Programme 7 la fonction `supprimer_clause`

On se rappelle qu'on est convenu que `F.(i) <- F.(m)` est de complexité constante.

Question III.17
Les deux boucles réalisent le parcours complet de la formule : la complexité est bien de l'ordre de la somme des longueurs des clauses.

```
38  let calculer_diff F =
39    let m = F.(0).(0) and n = F.(0).(1) in
40    let diff = make_vect (n+1) 0 in
41      for i=1 to m do
42        for j=1 to F.(i).(0) do
43         let litt = F.(i).(j) in
44           if litt>0 then diff.(litt) <- diff.(litt) + 1
45           else diff.(-litt) <- diff.(-litt) - 1
46        done
47      done ;
48      diff ;;
```

Programme 8 la fonction `calculer_diff`

Question III.18

```
49  let simplifier F alpha v =
50    let litt = if v = 1 then alpha else -alpha in
51    let satisfaites = ref 0 and i = ref 1 in
52      while !i <= F.(0).(0) do
53        if F.(!i).(0) = 1 && F.(!i).(1) = -litt then supprimer_clause F !i
54        else
55         let j = place F.(!i) litt in
56           if j > 0 then (incr satisfaites ; supprimer_clause F !i)
57           else let j = place F.(!i) (-litt) in
58             if j > 0 then supprimer_variable F.(!i) j ;
59             incr i
60      done ;
61      !satisfaites ;;
```

Programme 9 la fonction `simplifier`

La boucle commandée par `!i`, qui débute en ligne 52, permet de parcourir les différentes clauses.
La ligne 53 traite le cas d'une clause réduite au complémenté de `litt` : on supprime simplement la clause.
Sinon, on cherche si le littéral figure dans la clause courant. Dans ce cas, en ligne 56, on peut supprimer la clause et compter une clause satisfaite de plus.
Dans le cas contraire, on part à la recherche du complémenté du littéral (ligne 57), qu'on supprime le cas échéant (ligne 58), avant de passer à la clause suivante.
Attention : étant donnée la manière dont a été écrite `supprimer_clause`, après suppression d'une clause, il ne faut pas incrémenter le compteur `i` de clause pour passer à l'examen de la suivante.

La complexité de la fonction `simplifier` est alors clairement un $O(\ell)$ où ℓ est la somme des longueurs des clauses.

Question III.19

```
62  let cherche_max val d =
63    let n = vect_length d - 1 in
64    let imaximum = ref 1 and maximum = ref (abs(d.(1))) in
65      for i = 2 to n do
66        if abs(d.(i)) > !maximum && val.(i) < 0 then
67          (maximum := abs(d.(i)) ; imaximum := i)
68      done ;
69      !imaximum ;;

70  let heuristique F =
71    let n = F.(0).(1) in
72    let val = make_vect (n+1) (-1) in
73      val.(0) <- 0 ;
74      while F.(0).(0) > 0 do
75        let diff = calculer_diff F in
76        let alpha = cherche_max val diff in
77        let v = if diff.(alpha) > 0 then 1 else 0 in
78         val.(alpha) <- v ;
79         val.(0) <- val.(0) + (simplifier F alpha v)
80      done ;
81    val ;;
```

Programme 10 la fonction `heuristique`

On a écrit une fonction auxiliaire `cherche_max` qui détermine la variable α telle que abs(diff(α)) soit maximale, mais on a fait attention de ne choisir qu'une variable qui n'aurait pas déjà été affectée lors d'une étape précédente de l'heuristique : c'est-à-dire une variable qui correspond à une valeur de la valuation non affectée. Pour cela, on a initialisé la valuation à des valeurs (-1).
D'ailleurs, l'appel `heuristique F1` renvoie `[|6; 1; -1; -1; 0|]` : on a réussi à satisfaire 6 clauses (sur les 7 de départ), en posant $x = 1$, $t = 0$, et sans avoir affecté ni y, ni z.
Chaque appel à `cherche_max` est de complexité $O(n)$, à `calculer_diff` en $O(\ell)$, à `simplifier` en $O(\ell)$.
Il reste à évaluer le nombre d'étapes dans l'heuristique : il est borné par le nombre m de clauses, puisque à chaque étape une clause au moins est supprimée.
Finalement, la complexité de l'heuristique est $O(m\ell)$ où m est le nombre de clauses, et où ℓ est la somme des longueurs des clauses.

Partie IV — Étude d'un cas particulier

Question IV.20

IV.20.a Toute solution de F est clairement également solution de G. Inversement, quitte à renuméroter les littéraux, et parce qu'un littéral et son complémenté ne peuvent pas se présenter dans la même clause, on considère les deux cas suivants :

- $F = (x \vee y \vee \cdots) \wedge (\bar{x} \vee \bar{y} \vee \cdots) \wedge G$. Comme F est 1-occ, ni x, ni y ne figurent dans G. Soit donc une valuation solution de G, augmentée des valeurs $x = 1$ et $y = 0$: on obtient bien une solution de F.
- $F = (x \vee \bar{y} \vee \cdots) \wedge (\bar{x} \vee y \vee \cdots) \wedge G$. On procède de même, en augmentant une solution de G par les valeurs $x = 1$ et $y = 1$, obtenant ainsi une solution de F.

IV.20.b Notons $F' = (l_1 \vee \cdots \vee l_k \vee l_{k+1} \vee \cdots \vee l_{k+h}) \wedge G$. Les hypothèses permettent d'affirmer que ni x ni $\bar{x}$ ne figurent dans F', que F' est également 1-occ et qu'en outre la nouvelle clause respecte les contraintes de l'énoncé.
Montrons que F est satisfiable si et seulement si F' est satisfiable.
Soit v une valuation qui est solution de F : elle est bien sûr solution de G. Si $v(x) = 1$, alors l'un des littéraux l_{k+1}, …, l_{k+h} a nécessairement une valeur 1 dans la valuation v, ce qui garantit la satisfaction de la première clause de F', donc de F' tout entière. Si $v(x) = 0$, c'est cette fois l'un des littéraux l_1, …, l_k qui a pour valeur 1, mais là encore la première clause de F' est satisfaite, donc F' tout entière.
Réciproquement, soit v une valuation solution de F' : elle est bien sûr solution de G. En outre, l'un des littéraux l_i a pour valeur 1 dans la valuation v : si $1 \leqslant i \leqslant k$, on posera $v(x) = 0$, de sorte de satisfaire les deux premières clauses de F ; si $k + 1 \leqslant i \leqslant k + h$, on posera $v(x) = 1$, avec les mêmes effets. Il faut noter que comme ni x ni $\bar{x}$ ne figurent dans G, et ne sont pas répétés parmi les l_i, le choix fait pour $v(x)$ ne risque pas modifier la satisfiabilité de G ni de F'. On a ainsi trouvé une valuation v qui satisfait à la fois F' et F.

Question IV.21

IV.21.a On réduit par rapport à x la formule $(x \vee y \vee z) \wedge (\bar{x} \vee \bar{z} \vee t) \wedge (\bar{y} \vee \bar{t})$ grâce à la question 20.a : on obtient la formule réduite $(\bar{y} \vee \bar{t})$.

IV.21.b On réduit par rapport à t la formule $(\bar{x} \vee \bar{z} \vee t) \wedge (\bar{t} \vee \bar{u}) \wedge (z \vee u)$ grâce à la question 20.b : on obtient la formule réduite $(\bar{x} \vee \bar{z} \vee \bar{u}) \wedge (z \vee u)$.

Question IV.22

D'après les questions précédentes, il suffit ici de prouver qu'une formule 1-occ qui ne permet aucune réduction est satisfiable. Or une telle formule F est caractérisée par : chaque variable ne peut apparaître (sous forme complémentée ou non) qu'une seule fois dans la formule.
Si F est vide, F est satisfiable. Sinon, on trouve facilement une solution : il suffit de donner à chaque variable qui apparaît telle quelle la valeur 1, et à chaque variable qui apparaît sous forme complémentée la valeur 0.

Question IV.23

On commence par chercher une réduction de F par rapport à une variable x.
Si ce n'est pas possible, il suffit d'affecter dans le tableau val la valeur 0 à toute variable qui apparaît sous forme complémentée, et la valeur 1 à toute variable qui apparaît telle quelle, et c'est terminé.
Si on trouve une réduction selon 20.a, on opère la réduction, en affectant les valeurs idoines aux variables x et y dans la table val, et on appelle récursivement la fonction `calculer_solution` sur la nouvelle formule et la table val ainsi modifiée.
Dans le cas d'une réduction selon 20.b, on opère la réduction sans modifier la table val, et on appelle récursivement `calculer_solution` sur la nouvelle formule et la même table val. On détermine alors la valeur à affecter dans val à la variable x selon les indications de 20.b.

Question IV.24

On commence par réduire $F4$ par rapport à x (et z), selon 20.a, en posant $x = 1$ et $z = 0$. On travaille maintenant avec $F'4 = (\bar{y} \vee \bar{v}) \wedge (u \vee v) \wedge (\bar{t} \vee \bar{u})$.
On réduit maintenant $F'4$ par rapport à v, selon 20.b, obtenant $F''4 = (\bar{y} \vee u) \wedge (\bar{t} \vee \bar{u})$.
Nouvelle réduction par rapport à u, selon 20.b, donnant $F'''4 = (\bar{y} \vee \bar{t})$. Finalement on pose $y = 0$ et $t = 0$.
De retour à $F''4$, on achève son traitement en posant $u = 1$ (en fait ça n'a pas d'importance ici).
De retour à $F'4$, on achève son traitement en posant $v = 0$ (en fait ça n'a pas d'importance non plus).
Finalement une solution de $F4$ est donc $x = 1$, $y = 0$, $z = 0$, $t = 0$, $u = 1$, $v = 0$.