TD Option informatique $n°01$ – Satisfiabilité

On considère un ensemble dénombrable de variables $V = \{v_1, \ldots, v_n, \ldots\}$. Dans toute la suite, on s'intéresse aux formules logiques *sous forme normale conjonctive* ayant pour ensemble de variables V.

On rappelle qu'un **littéral** est ou bien une variable ou bien la négation d'une variable, qu'une **clause** est une disjonction de littéraux dans laquelle chaque variable (niée ou pas) apparaît au plus une fois, et qu'une **formule logique sous forme normale conjonctive** est une conjonction de clauses.

On rappelle qu'une **valuation** est une application d'un ensemble de variables dans l'ensemble $\{v, f\}$; une valuation induit une application sur les clauses et sur les formules logique sous forme normale conjonctive comme suit : une clause s'évalue en v si au moins un de ses littéraux vaut v (et s'évalue en f sinon), une formule logique sous forme normale conjonctive s'évalue en v si toutes ses clauses s'évaluent en v (et s'évalue en f sinon). Une formule logique sous forme normale conjonctive est dite **satisfiable** s'il existe une valuation de ses variables pour laquelle elle s'évalue en v. Si une formule logique est satisfiable, on appelle alors **modèle** de cette formule logique toute valuation des variables pour laquelle elle s'évalue en v. Par exemple, la formule logique sous forme normale conjonctive

$$F_0 = (\overline{x} \vee y \vee z) \wedge (\overline{x} \vee \overline{y}) \wedge (x \vee \overline{z}) \wedge (x \vee \overline{y})$$

est satisfiable et elle a pour modèle (non unique !) $x = v$, $y = f$ et $z = v$.

Étant donnée une formule logique sous forme normale conjonctive F, on note dans ce problème $max(F)$ le nombre maximum de clauses de F s'évaluant en v pour une même valuation. En notant m le nombre de clauses de F, on obtient donc 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 considère les propositions suivantes :

1. au moins une des personnes porte un chapeau ;
2. au moins une des personnes ne porte pas de chapeau ;
3. si X porte un chapeau, ni Y ni Z n'en portent ;
4. 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 v si, respectivement, X, Y et Z portent un chapeau, (f sinon).

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.

2. Écrire une formule logique sous forme normale conjonctive exprimant le fait que les propositions a), b), c) et d) doivent être satisfaites simultanément. Indiquer si cette formule logique sous forme normale conjonctive est satisfiable et donner l'ensemble des modèles pour cette formule logique sous forme normale conjonctive. Une démonstration est attendue.

On considère maintenant la formule logique sous forme normale conjonctive F_1 dépendant des variables x, y, z :

$$F_1 = (x \vee y \vee z) \wedge (\overline{x} \vee z \vee \overline{t}) \wedge (x \vee \overline{y} \vee \overline{z}) \wedge (x \vee y \vee \overline{t}) \wedge (\overline{x} \vee \overline{z} \vee \overline{t}) \wedge (\overline{x} \vee t) \wedge (x \vee \overline{y} \vee z).$$

3. Indiquer si F_1 est satisfiable et donner l'ensemble des modèles pour F_1. Une démonstration est attendue.

Une instance de 3-SAT est une formule logique sous forme normale conjonctive dont toutes les clauses contiennent 3 littéraux.

4. Justifier qu'on peut transformer une formule sous forme normale conjonctive dont les clauses contiennent **au plus** 3 littéraux en instance de 3-SAT (c'est-à-dire en formule sémantiquement égale ayant exactement 3 littéraux).

5. 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 dans laquelle apparaissent au plus n variables booléennes.
On note $\mathcal{V}$ l'ensemble des 2^n valuations des variables de F.
Soit $val \in \mathcal{V}$ une valuation des n variables. Si C est une clause, on note $\varphi(C, val) = 1$ si C s'évalue en v pour la valuation val et $\varphi(C, val) = 0$ si C s'évalue en f pour la valuation val. On note $\Psi(F, val)$ le nombre de clauses de F s'évaluent en v pour la valuation val. On a donc $\Psi(F, val) = \sum_{C \text{ clause de } F} \varphi(C, val)$ et $max(F) = \max_{val \in \mathcal{V}} \Psi(F, val)$.

6. Soit C une clause de F. Indiquer avec démonstration en fonction de n la valeur de $\sum_{val \in \mathcal{V}} \varphi(C, val)$.

7. Soit m le nombre de clauses dont F est la conjonction. En considérant la somme $\sum_{C \text{ clause de } F} \sum_{val \in \mathcal{V}} \varphi(C, val)$, donner en fonction de m un minorant de $max(F)$.

8. 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 associer à chaque variable propositionnelle son numéro. Ainsi, si x, y et z désignent les variables v_1, v_3 et v_4, on associe à x le numéro 1, à y le numéro 3 et à z le numéro 4. Par ailleurs, on numérote aussi les littéraux obtenus en niant une variable propositionnelle, avec l'opposé du numéro associé à la variable niée. Ainsi, avec l'exemple ci-dessus, au littéral $\overline{x}$ on associe le numéro -1, au littéral $\overline{y}$ on associe le numéro -3, au littéral $\overline{z}$ on associe le numéro -4. **Pour alléger les explications, on identifie désormais un littéral avec son numéro.**

Pour coder une valuation d'un ensemble de n variables propositionnelles, 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 du tableau donne la valeur de la variable de numéro i, codée par un entier (0 si la variable s'évalue en f, 1 si la variable s'évalue en v, 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, 3, 4 et 6, la clause $(x \vee \overline{t} \vee y)$ est codé par le tableau `[|3; 1; -6; 3|]`.

Pour coder une formule logique sous forme normale conjonctive, on utilise un tableau de tableaux d'entiers. Pour i supérieur ou égal à 1, la ligne d'indice i décrit la $i^{\text{ième}}$ clause de la formule logique sous forme normale conjonctive. 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, si les variables sont x, y, z et t, numérotées respectivement par 1, 3, 4 et 6, la formule logique sous forme normale conjonctive $F_3 = (x \vee \overline{y} \vee z \vee t) \wedge (\overline{x} \vee \overline{z}) \wedge (x \vee \overline{t} \vee y)$ est codée par le tableau

```
[|[|3;4|]; [|4;1;-3;4;6|];[2;-1;-4|];[|3;1;-6;3|]|].
```

Soit `f` un tableau de tableaux d'entiers définissant une formule logique sous forme normale conjonctive F et soient i et j deux entiers compris entre 1 et `f.(0).(0)` (le tableau `F.(i)` définit donc la $i^{\text{ième}}$ clause de F). Dans un calcul de complexité, on considérera l'instruction `f.(i) <- f.(j)` comme une opération élémentaire (ainsi que les autres affectations, les additions d'entiers ou les comparaisons).

2 Satisfiabilité, méthode exacte

9. Écrire en OCaml une fonction `valeur_clause` telle que, si `c` est un tableau d'entiers codant une clause C et si `v` est un tableau codant une valuation val d'un ensemble de variables contenant les variables de C, alors `valeur_clause c v` retourne l'évaluation de C (c'est-à-dire 0 ou 1, l'entier 0 codant la valeur f et l'entier 1 la

valeur v) pour la valuation val.
Indiquer avec démonstration la complexité de la fonction `valeur_clause`.

10. Écrire en OCaml une fonction `satisfait_formule` telle que, si `f` est un tableau de tableaux d'entiers codant une formule logique sous forme normale conjonctive F et si `v` est un tableau codant une valuation val de l'ensemble des variables de F, alors `satisfait_formule f v` retourne `true` si val satisfait F et `false` sinon.
Indiquer avec démonstration la complexité de la fonction `satisfait_formule`.

11. On considère une formule logique sous forme normale conjonctive 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 val_i égale à 0 ou 1 (correspondant à l'évaluation de la variable i : si k vaut 0, aucune variable n'a une valeur fixée).
Écrire en OCaml une fonction récursive `resoudre_rec` qui détermine s'il existe une valuation val des variables telle que :
    - pour i compris entre 1 et k, l'évaluation pour val de la variable i correspond à val_i ;
    - val satisfait F.

    Précisément, si `f` est un tableau de tableaux d'entiers codant une formule logique sous forme normale conjonctive F, si `v` est un tableau d'entiers servant à coder une valuation, si k est un entier compris entre 0 et le nombre de variables de F, et si le tableau `v` contient soit 0, soit 1 dans les cases d'indices 1, 2, ..., k, alors `resoudre_rec F v k` retourne `true` s'il existe une telle valuation, et `false` sinon. De plus, cette fonction devra avoir pour effet de bord de modifier les cases de `v` à partir de la $(k+1)^{\text{ième}}$ uniquement de sorte que `v` code une telle valuation convenable.

12. Écrire en OCaml une fonction `resoudre` telle que, si `f` est un tableau de tableaux d'entiers codant une formule logique sous forme normale conjonctive F, alors `resoudre f` retourne un tableau d'entiers tel que :
    - si F est satisfiable, le tableau renvoyé code un modèle 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.

13. Évaluer la complexité de la fonction `resoudre` appliquée à une formule logique sous forme normale conjonctive F en fonction du nombre n de variables de F et de la somme ℓ des longueurs des clauses de F (la "longueur de F").

3 MAX-SAT

Dans cette partie, on ne s'intéresse plus seulement à savoir si une formule logique est satisfiable, mais au calcul, pour une formule logique sous forme normale conjonctive F, de $max(F)$.

On va définir une heuristique gloutonne, 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 sous forme normale conjonctive ; étant donnée une variable a de F, on note $\text{diff}(F,a)$ le nombre de fois où a figure dans F diminué du nombre de fois où $\overline{a}$ figure dans F. L'heuristique utilise la transformation suivante :

(a) On calcule pour chaque variable a de F le nombre $\text{diff}(F,a)$.

(b) On détermine une variable a_0 de F qui réalise le maximum de $|\text{diff}(F,a)|$.

(c) On pose $val(a_0) = v$ si on a $\text{diff}(F,a_0) > 0$ et $val(a_0) = f$ sinon.

(d) On supprime la variable a_0 de F en tenant compte de la valeur choisie (son évaluation pour val) de façon à ne conserver que les clauses qui restent à satisfaire après le choix de la valeur de a_0 ; on comptabilise le nombre de clauses satisfaites. On obtient ainsi une formule logique sous forme normale conjonctive 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 F'' 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.

Exemple : sur la formule logique sous forme normale conjonctive $F_0 = (\overline{x} \vee y \vee z) \wedge (\overline{x} \vee \overline{y}) \wedge (x \vee \overline{z}) \wedge (x \vee \overline{y})$ de l'introduction, l'heuristique se détaille comme suit. On a $\text{diff}(F_0,x) = 0$, $\text{diff}(F_0,y) = -1$ et $\text{diff}(F_0,z) = 0$, ce qui conduit à choisir comme première variable y, avec $y = f$. Ceci permet de satisfaire 2 clauses : $(\overline{x} \vee \overline{y})$ et $(x \vee \overline{y})$, ces

deux clauses seront donc retirées à F'_0. La clause $(\overline{x} \vee y \vee z)$, compte tenu de $y = f$, devient $(\overline{x} \vee z)$ dans F'_0. Enfin, y n'apparaît pas dans la clause $(x \vee \overline{z})$, qui n'est donc pas modifiée dans F'_0. Finalement, on a $F'_0 = (\overline{x} \vee z) \wedge (x \vee \overline{z})$.
On poursuit l'algorithme mais on a maintenant $\text{diff}(F'_0, x) = \text{diff}(F'_0, z) = 0$, on peut donc prendre indifféremment x ou y comme seconde variable, et lui donner indifféremment la valeur v ou f. Prenons par exemple $x = f$, ce qui conduit à $F''_0 = \overline{z}$ et donc $z = f$ également. Sur cet exemple l'heuristique gloutonne a bien permis de satisfaire les quatre clauses de F_0 (donc $max(F_0)$ clauses puisque c'est une formule satisfiable).

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

De façon à implémenter l'heuristique en OCaml, on définit cinq fonctions dans les cinq questions suivantes.

15. Écrire en OCaml une fonction `place` telle que, si `c` est un tableau codant une clause C, et si `litt` est un littéral, `place c litt` retourne :
    - 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.

16. Écrire en OCaml une fonction `supprimer_variable` telle que, si `c` est 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.

17. Écrire en OCaml une fonction `supprimer_clause` telle que, si `f` est un tableau de tableaux d'entiers codant une formule logique sous forme normale conjonctive 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 sous forme normale conjonctive 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 sous forme normale conjonctive ni de la valeur de i.

18. Écrire en OCaml une fonction `calculer_diff` telle que, si `f` est un tableau de tableaux d'entiers codant une formule logique sous forme normale conjonctive F, alors `calculer_diff f` retourne un tableau d'entiers donnant pour chaque variable a de F la valeur de $\text{diff}(F, a)$ décrite avant 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.

19. On s'intéresse dans cette question à une transformation automatique d'une formule logique sous forme normale conjonctive F lorsqu'on impose qu'une variable a prend la valeur ν. Cette transformation est effectuée à l'aide d'une fonction nommée `simplifier` prenant F, a et ν comme arguments. Si ν vaut 1 (pour v), on note `litt` le littéral a et sinon on note `litt` le littéral $\overline{a}$. Quand a prend la valeur ν, `litt` prend la valeur 1 et la négation 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.

    Écrire en OCaml la fonction `simplifier` telle que, si `f` est un tableau de tableaux d'entiers codant une formule logique sous forme normale conjonctive F, si `a` est un entier représentant une variable a de F et si `v` vaut 0 ou 1, alors `simplifier f a v` a pour effet de bord de modifier `f` pour que `f` code la formule logique sous forme normale conjonctive obtenue à partir de F décrite ci-dessus, et retrouen le nombre de clauses de la formule logique sous forme normale conjonctive F initiale qui contenaient `litt` défini ci-dessus.

    Évaluer la complexité de la fonction simplifier.

20. Écrire en OCaml une fonction `heuristique` telle que, si `f` est un tableau de tableaux d'entiers codant une formule logique sous forme normale conjonctive F, alors `heuristique f` retourne 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 tableau

devra contenir le nombre de clauses satisfaites par la valuation déterminée par l'heuristique.
Évaluer la complexité de la fonction heuristique.

4 Étude d'un cas particulier

On revient au problème de la satisfiabilité. On s'intéresse dans cette partie à une formule logique sous forme normale conjonctive 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 1-occ** une telle formule logique sous forme normale conjonctive. Par exemple, la formule logique sous forme normale conjonctive F_4 ci-dessous, ayant six variables x, y, z, t, u, v, est une formule 1-occ :

$$F_4 = (x \vee y \vee z) \wedge (\overline{x} \vee \overline{z} \vee t) \wedge (\overline{y} \vee \overline{v}) \wedge (u \vee v) \wedge (\overline{t} \vee \overline{u})$$

On cherche à déterminer un modèle d'une formule 1-occ.

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

$$F = (x \vee l_1 \vee l2 \cdots \vee l_k) \wedge (\overline{x} \vee l_{k+1} \vee l_{k+2} \vee \cdots \vee l_{k+h}) \wedge G$$

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

(a) Montrer que si l'ensemble $\{l_1, l_2, \cdots, l_k, l_{k+1}, l_{k+2}, \cdots, l_{k+h}\}$ comprend à la fois une variable et sa négation, 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 G.

(b) On suppose que l'ensemble $\{l_1, l_2, \cdots, l_k, l_{k+1}, l_{k+2}, \ldots, l_{k+h}\}$ ne comprend jamais à la fois une variable et sa négation. Indiquer une formule 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 F'.

Remarque : on rappelle que, par définition dans ce problème, une clause ne contient pas à la fois une variable et sa négation, et que la formule logique sous forme normale conjonctive vide est toujours satisfaite.

22. *Exemples de réduction.*

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

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

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

24. Décrire sans l'implémenter une fonction récursive `calculer_solution` :
• qui prend comme argument un tableau `f` codant une formule 1-occ F et un tableau `val` permettant de coder une valuation des variables ;
• qui transforme le tableau `val` pour que, après exécution, val contienne un modèle de F.

25. Appliquer la fonction `calculer_solution` décrite dans la question précédente à la formule F_4. Détailler chaque appel récursif.