

# Devoir surveillé n°6

## Des formes normales et des arbres de mots

### Consignes

La durée du devoir est de 4 h. Les documents, notes de cours, calculatrices et tous dispositifs électroniques sont interdits et doivent être éteints et rangés dans vos sacs. Ce sujet comprend 6 pages. **Vos réponses doivent être rigoureusement écrites et justifiées, même si la question ne le précise pas.** Votre nom doit être noté sur toutes vos copies, chaque page doit être numérotée sous la forme  $k/n$  avec  $n$  le nombre total de pages utilisées. L'ensemble de vos copies doit être glissé, sans le sujet, dans votre première copie double. Vos résultats doivent être soulignés ou encadrés, vous pouvez également souligner ou encadrer des résultats intermédiaires si vous voulez les mettre en valeur. Un espace de quelques lignes doit être laissé vide au début de la première page, une marge à gauche doit être réservée sur chacune des pages. Le non-respect de ces consignes pourra être l'objet d'un malus de points.

**Les questions de programmation sont toutes à traiter en OCaml ou en C :** toutes les fonctions des sections A.1 et B.1 “Traits et éléments techniques à connaître” du programme de la MP2I/MPI sont autorisées sans rappel. Les fonctions des sections A.2 et B.2 “Éléments techniques devant être reconnus et utilisables après rappel” du programme de la MP2I/MPI le sont si leurs types ou prototype, si besoin leurs complexités, sont rappelées. Les autres ne sont pas autorisées (sauf si l'énoncé les fournit). Pour C, les bibliothèques `<assert.h>`, `<stdbool.h>`, `<stddef.h>`, `<stdint.h>`, `<stdio.h>`, `<stdlib.h>` sont supposées chargées. Le langage attendu à une question est précisé par l'utilisation de la syntaxe OCaml ou C pour définir le type ou prototype de fonction attendu.

**Bon courage !**

Le sujet est divisé en deux problèmes entièrement indépendants. Le premier problème porte sur la manipulation de formes normales conjonctives en C, la seconde partie propose l'implémentation d'ensemble de mots binaires en OCaml par des arbres de mots.

## I Autour de la forme normale conjonctive (C)

### I.A Représentation des formules sous forme normale conjonctive

On note  $\mathcal{P}$  un ensemble de variables propositionnelles. Un **littéral** est soit une variable propositionnelle (dit littéral **positif**), soit la négation d'une variable propositionnelle (dit littéral **négatif**). Pour  $l$  un littéral, on note  $|l|$  la variable propositionnelle sous-jacente. Une **occurrence** d'un littéral  $l$  dans une formule est une occurrence de la variable propositionnelle  $|l|$  dans la formule.

On rappelle qu'une **clause** est une disjonction de littéraux et une formule sous **forme normale conjonctive** est une conjonction de clauses disjonctives. Une clause est dite **unitaire** si elle ne contient qu'un seul littéral.

Nous allons représenter une formule sous forme normale conjonctive par la structure C suivante :

```

1 #define MAX_VP 100
2 typedef int littéral;
3 typedef struct {
4     littéral* litteraux;
5     int nb_litteraux; } clause;
6 typedef struct {
7     clause* clauses;
8     int nb_clauses; } cnf;

```

C

La première instruction définit la constante littérale entière `MAX_VP`. La structure `clause` représente une clause, avec un tableau de littéraux `litteraux` de longueur `nb_litteraux`. Chaque littéral est un entier  $k$  tel que  $|k| \in \llbracket 1, \text{MAX\_VP} \rrbracket$ , avec  $k > 0$  représentant le littéral positif  $p_k$  et  $k < 0$  représentant le littéral négatif  $\neg p_k$ . La structure `cnf` représente une formule sous forme normale conjonctive, avec un tableau de clauses `clauses` de longueur `nb_clauses`.

La fonction `int abs(int x)` de la bibliothèque `stdlib.h` renvoie la valeur absolue de l'entier en entrée. Dans le reste du problème, on ignore les problématiques de libération de la mémoire allouée sur le tas en C.

► **Question 1** Écrire une fonction `int` `taille(cnf f)` qui prend en entrée une formule `f` et qui renvoie le nombre de littéraux de la formule (en comptant les répétitions).

► **Question 2** Écrire une fonction `int` `nb_clauses_unitaires(cnf f)` qui prend en entrée une formule `f` et qui renvoie le nombre de clauses unitaires de la formule.

► **Question 3** Écrire une fonction `int` `nb_occurrences(cnf f, literal l)` qui prend en entrée une formule `f` et un littéral `l` et qui renvoie le nombre d'occurrences de `l` dans la formule.

On définit une valuation en C par un tableau de booléens `bool*` `v` de taille `MAX_VP + 1`, la première case contenant une valeur quelconque et `v[i]` contenant la valeur de vérité de  $p_i$  pour  $i \in \llbracket 1, \text{MAX\_VP} \rrbracket$ .

► **Question 4** Écrire une fonction `bool` `satisfait(cnf f, bool* v)` qui prend en entrée une formule `f` et une valuation `v` et qui renvoie `true` si la valuation `v` satisfait la formule `f`, `false` sinon.

► **Question 5** Déterminer précisément sa complexité.

► **Question 6** En considérant qu'itérer sur toutes les valuations possibles prend un temps constant par valuation, déterminer la complexité d'un algorithme de recherche de valuation satisfaisant une formule sous forme normale conjonctive par force brute. *On exprimera cette complexité en fonction du nombre de variables propositionnelles considérées et de la taille de la formule.*

## I.B Validité d'une formule sous forme normale conjonctive

► **Question 7** Soit  $c$  une clause. Déterminer et justifier une condition nécessaire et suffisante pour que  $c$  soit valide.

► **Question 8** Soient  $\varphi, \psi$  deux formules quelconques. Montrer que  $\varphi \wedge \psi$  est valide si et seulement si  $\varphi$  et  $\psi$  sont valides.

► **Question 9** En déduire en le justifiant une condition nécessaire et suffisante pour qu'une formule sous forme normale conjonctive  $f$  soit valide.

► **Question 10** Écrire une fonction `bool` `est_valide(cnf f)` qui prend en entrée une formule `f` et qui renvoie `true` si la formule est valide, `false` sinon. Préciser sans la justifier sa complexité temporelle et spatiale.

## I.C Clauses d'Horn et satisfaisabilité

**Définition des clauses d'Horn** Dans cette section, on se restreint à manipuler des formules sous forme normale conjonctive ne contenant que des clauses d'Horn :

### Définition 1 – Clause et formule d'Horn

Une **clause d'Horn** est une clause qui contient au plus un littéral positif. Une clause d'Horn est dite **négative pure** si elle ne contient que des littéraux négatifs, **implication** si elle contient un littéral positif (et éventuellement aucun littéral négatif). Une formule d'Horn est une formule sous forme normale conjonctive dont toutes les clauses sont des clauses d'Horn.

► **Question 11** Pour les formules suivantes, déterminer celles qui sont des clauses d'Horn. On justifiera seulement les réponses négatives.

1.  $x \vee y \vee z$ ;

3.  $x \rightarrow \neg y \vee \neg z$ ;

5.  $\neg x \vee \neg y \vee \neg z$ ;

2.  $\neg x \vee z \vee \neg y$ ;

4.  $x \wedge \neg y \vee z \vee \neg x$ ;

6.  $\neg x \vee \neg(y \rightarrow z)$ .

► **Question 12** Montrer que pour toute clause d'Horn, il existe une formule équivalente à cette clause sous la forme d'une implication entre :

- à gauche, une conjonction de variables propositionnelles ou  $\top$  ;
  - à droite, une variable propositionnelle ou  $\perp$  ;
- où  $\top$  (resp.  $\perp$ ) est une formule satisfaite par toute valuation (resp. insatisfaite par toute valuation).

**Application à un problème de décision** Ces clauses d'Horn permettent d'énoncer des faits et des implications entre faits. On peut donc les utiliser pour modéliser des problèmes de décision concrets : on définit les variables propositionnelles  $p_1, \dots, p_5$  représentant les énoncés suivants :

- $p_1$  : le meurtre a eu lieu dans la cuisine ;
  - $p_2$  : la docteure était en mission hors de la maison à 20 h ;
  - $p_3$  : le meurtre a eu lieu à 20 h ;
  - $p_4$  : le colonel est innocent ;
  - $p_5$  : la docteure est innocente.
- **Question 13** Exprimer sous la forme d'une clause d'Horn les énoncés suivants :
- si la docteure était hors de la maison à 20 h et si le meurtre a eu lieu à 20 h, alors la docteure est innocente ;
  - la docteure était en mission hors de la maison à 20 h ;
  - l'un des deux suspects est coupable ;
  - le meurtre a eu lieu à 20 h.

**Satisfaisabilité de formules d'Horn par un algorithme glouton** Il existe un algorithme glouton pour déterminer si une formule sous forme normale conjonctive dont les clauses sont d'Horn est satisfaisable, et qui permet même de construire une valuation la satisfaisant.

**Requiert :**  $f$  une conjonction de clauses d'Horn,  
 $\mathcal{P}$  l'ensemble des variables propositionnelles ayant une occurrence dans  $f$ .

```

1: Fonction HORNSAT( $f$ )
2:   initialement,  $v(p) = \text{faux}$  pour tout  $p \in \mathcal{P}$ .
3:   Tant que il existe  $c$  clause implication de  $f$  telle que  $v \not\models c$ , faire
4:      $p \leftarrow$  le littéral positif de  $c$ 
5:      $v(p) \leftarrow \text{vrai}$ 
6:   Pour chaque clause négative pure  $c$  de  $f$ , faire
7:     Si  $v \not\models c$  alors
8:       retourne la formule  $f$  n'est pas satisfaisable.
9:   retourne  $v$ 

```

Pseudo-code

- **Question 14** Montrer la terminaison de cet algorithme.
- **Question 15** Déterminer la complexité de cet algorithme en fonction de la taille de la formule en entrée.
- **Question 16** Que peut-on dire de la première clause d'Horn implication trouvée par l'algorithme HORNSAT dans la boucle **Tant que** ?
- **Question 17** Montrer l'invariant suivant de la boucle **Tant que** :
- « Pour tout  $p \in \mathcal{P}$ , si  $v(p) = \text{vrai}$ , alors pour toute valuation  $v'$  modèle de  $f$ ,  $v'(p) = \text{vrai}$ . »
- **Question 18** En déduire la correction de l'algorithme HORNSAT.
- **Question 19** Utiliser l'algorithme HORNSAT pour déterminer si la conjonction des clauses déterminées à la Question 13 est satisfaisable. En déduire le meurtrier.

**Implémentation de HORNSAT** ► **Question 20** Écrire une fonction `bool est_formule_horn(cnf f)` qui prend en entrée une formule `f` et qui renvoie `true` si `f` est une formule d'Horn, `false` sinon.

► **Question 21** Écrire une fonction `litteral get_litteral_positif(clause c)` qui prend en entrée une clause d'Horn `c` et qui renvoie 1 le littéral positif de `c` si elle est une clause implication, 0 si la clause est négative pure.

► **Question 22** Implémenter une fonction `bool* horn_sat(cnf f)` qui prend en entrée une formule `f` d'Horn et qui renvoie une valuation la satisfaisant si elle existe, `NULL` sinon.

*Une représentation spécialisée des formules d'Horn par des graphes permet de résoudre le problème HORN-SAT en temps linéaire, ce problème est d'ailleurs P-complet. Cette excellente complexité permet d'appliquer l'algorithme à des problèmes de programmation logique. Le langage Prolog utilise ces résultats en les étendant à la logique du premier ordre, ce qui permet une implémentation efficace d'algorithmes d'analyse syntaxique et lexicale, de résolution de requêtes déductives dans des bases de données, en intelligence artificielle, etc. Elles permettent aussi d'exprimer et d'analyser formellement de manière naturelle les systèmes de contrôle, les systèmes de production, ou encore les protocoles de communication comme avec logiciel ProVerif.*

## II Facteurs de mots binaires (OCaml)

### II.A Rappels, définitions et propriétés élémentaires

#### Langage OCaml

On rappelle que la fonction `List.rev : 'a list -> 'a list` renvoie la liste renversée de la liste passée en argument. De plus, la fonction `Array.of_list : 'a list -> 'a array` renvoie le tableau correspondant à la liste passée en argument. Enfin, la fonction `List.of_array : 'a array -> 'a list` renvoie la liste correspondant au tableau passé en argument.

#### Mots

Un **mot binaire** (dans la suite, simplement **mot**) est une suite finie  $m = m_0 m_1 \dots m_{\ell-1}$  où chaque  $m_i$  est un symbole dans l'alphabet  $\{0, 1\}$  et  $\ell$  est sa longueur. On note  $\varepsilon$  le mot vide, de longueur 0. On note  $\ell(m)$  la longueur du mot  $m$ .

Pour deux mots  $m = m_0 m_1 \dots m_{\ell-1}$  et  $m' = m'_0 m'_1 \dots m'_{\ell'-1}$ , on note  $m \cdot m' = m_0 m_1 \dots m_{\ell-1} m'_0 m'_1 \dots m'_{\ell'-1}$ , la **concaténation** de  $m$  et  $m'$ . On note  $m^k$  le mot  $m \cdot m \dots m$  ( $k$  fois). On pourra omettre le point «  $\cdot$  » dans la notation de la concaténation.

Pour  $0 \leq i \leq j \leq \ell(m)$ , on note  $m[i : j]$  le mot  $m_i m_{i+1} \dots m_{j-1}$ , qui est appelé **facteur** de  $m$ . Un **préfixe** (resp. **suffixe**) de  $m$  est un facteur de la forme  $m[0 : j]$  (resp.  $m[i : \ell(m)]$ ). On remarque que  $\varepsilon$  est préfixe, suffixe et facteur de tout mot  $m$  (avec  $i = j$ ,  $j = 0$  ou  $i = \ell(m)$ ). On note  $m[: j] = m[0 : j]$  et  $m[i :] = m[i : \ell(m)]$ . Enfin, on note :

- $P(M) = \{m[: j] \mid m \in M, 0 \leq j \leq \ell(m)\}$  l'ensemble des préfixes de mots de  $M$ ;
- $S(M) = \{m[i :] \mid m \in M, 0 \leq i \leq \ell(m)\}$  l'ensemble des suffixes de mots de  $M$ ;
- $F(M) = \{m[i : j] \mid m \in M, 0 \leq i \leq j \leq \ell(m)\}$  l'ensemble des facteurs de mots de  $M$ .

En OCaml, les mots sont représentés par des tableaux ne contenant que les entiers 0 et 1. On utilisera le type OCaml suivant : `type mot = int array`.

#### Arbres binaires

On définit l'ensemble des arbres binaires par induction.

- L'arbre vide, noté  $V$ , est un arbre binaire;
- si  $g$  et  $d$  sont deux arbres binaires et  $x$  une étiquette (dans ce sujet, les étiquettes sont des booléens), alors l'arbre binaire  $N(x, g, d)$  est un arbre binaire.

On utilise ici le type d'arbre suivant :

```
1 type arbre = V | N of bool * arbre * arbre
```

► **Question 23** Implémenter deux fonctions `hauteur : arbre -> int` et `taille : arbre -> int` qui renvoient respectivement la hauteur et la taille (le nombre de nœuds) d'un arbre binaire. On choisit ici la convention  $h(V) = 0$ .

Dans la suite, on notera mathématiquement  $t(a)$  la taille d'un arbre binaire  $a$  et  $h(a)$  sa hauteur.

► **Question 24** Montrer que pour tout arbre binaire  $a$ , on a  $t(a) \leq 2^{h(a)} - 1$ .

► **Question 25** Sans le justifier, donner une valeur minimale de la taille d'un arbre binaire en fonction de sa hauteur.

## II.B Représentation d'un ensemble de mots par un arbre binaire

Dans cette partie, on utilise les arbres binaires tels que définis précédemment pour représenter un ensemble de mots. Un **arbre de mots** est un arbre binaire dont chaque nœud est étiqueté par un booléen. L'arbre de mots  $a$  représente l'ensemble de mots  $\mathbb{M}(a)$  comme suit.

- Si  $a = V$ , alors  $\mathbb{M}(a) = \emptyset$ ;
- si  $a = N(b, g, d)$ , alors  $\mathbb{M}(a)$  est l'ensemble des mots composés de  $\varepsilon$  si  $b$  est vrai, les mots de la forme  $0m$  pour tout  $m \in \mathbb{M}(g)$  et les mots de la forme  $1m$  pour tout  $m \in \mathbb{M}(d)$ .

Autrement dit,  $\mathbb{M}(a)$  est l'ensemble des mots correspondant aux chemins de la racine de  $a$  aux nœuds étiquetés par `true`, en utilisant le caractère 0 quand on va à gauche et 1 quand on va à droite. On dessine graphiquement des arbres de mot comme suis, en écrivant  $T$  dans les nœuds étiquetés par `true` et  $F$  dans les nœuds étiquetés par `false`.

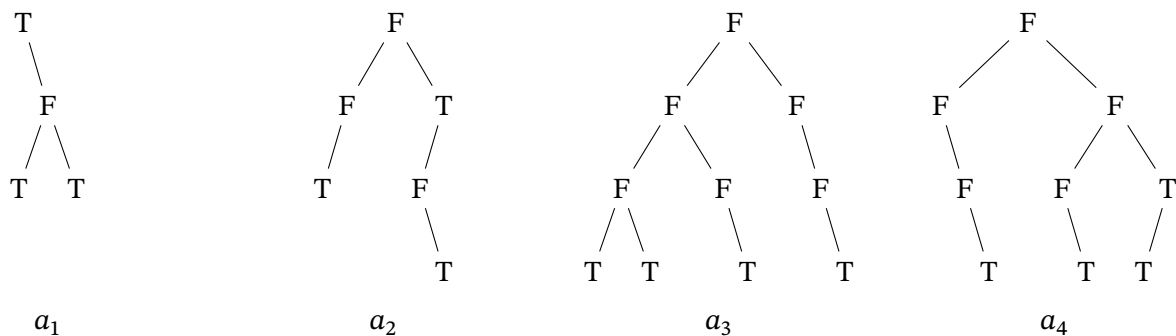


FIGURE 1 – Des arbres de mots

Ici,  $a_1$  est l'arbre  $N(T, V, N(F, N(T, V, V), N(T, V, V)))$ . On a trois  $T$  dans l'arbre  $a_1$ ,  $\mathbb{M}(a_1)$  contient donc trois mots : dans l'ordre préfixe, ces mots sont  $\varepsilon$ , 10 et 11.

► **Question 26** Donner sans justifier l'ensemble de mots représenté par l'arbre  $a_2$  et sa représentation par le type arbre en OCaml.

On remarque qu'une infinité d'arbres de mots peuvent représenter un même ensemble de mots. En effet, il suffit de remplacer un arbre vide dans un arbre de mots par  $N(F, V, V)$  pour obtenir un arbre de mots représentant le même ensemble de mots. On appelle **arbre de mot réduit** un arbre de mot ne contenant aucune feuille  $N(F, V, V)$ .

► **Question 27** Dessiner sans justifier un arbre de mots réduit représentant l'ensemble de mots suivant :

$$M = \{\varepsilon, 00, 010, 1, 110, 111\}.$$

► **Question 28** Montrer que tout arbre de mots  $a$  contenant un nœud étiqueté  $T$  vérifie  $\mathbb{M}(a) \neq \emptyset$ .

► **Question 29** Montrer que, pour tout ensemble fini de mots  $M$ , il existe un unique arbre de mots réduit représentant  $M$ . Cette question est difficile : on pourra montrer séparément l'existence et l'unicité, et il ne faut pas hésiter à proposer une stratégie de preuve même si l'on n'arrive pas à la mener à terme.

Dans la suite, on note  $\mathbb{A}(M)$  l'unique arbre de mots réduit représentant l'ensemble de mots  $M$ . On a donc  $\mathbb{M}(\mathbb{A}(M)) = M$  pour tout ensemble de mots  $M$  et  $\mathbb{A}(\mathbb{M}(a)) = a$  pour tout arbre de mots  $a$ .

► **Question 30** Écrire une fonction `zeros_puis_uns` : `int`  $\rightarrow$  arbre qui prend en argument un entier  $n \geq 0$  et qui renvoie l'arbre de mots réduit représentant l'ensemble des mots de la forme  $0^p 1^q$  avec  $p + q = n$ . Indication : l'arbre  $a_3$  de la Figure 1 est l'arbre `zeros_puis_uns 3`.

► **Question 31** Écrire une fonction `sN` : arbre  $\rightarrow$  arbre qui prend en argument un arbre de mots  $a$  et qui renvoie  $V$  si  $a = N(F, V, V)$  et qui renvoie  $a$  sinon.

► **Question 32** Écrire une fonction `compter` : arbre  $\rightarrow$  `int` qui prend en argument un arbre de mots  $a$  et qui renvoie le cardinal de  $\mathbb{M}(a)$ . On attend une complexité en  $\mathcal{O}(t(a))$  sans avoir besoin de la justifier.

► **Question 33** Écrire une fonction `recherche` : mot  $\rightarrow$  arbre  $\rightarrow$  `bool` qui prend en argument un mot  $m$  et un arbre de mots  $a$  et qui renvoie `true` si  $m \in \mathbb{M}(a)$ , `false` sinon. On attend une complexité en  $\mathcal{O}(\ell(m))$  sans avoir besoin de la justifier.

► **Question 34** Écrire une fonction `ajouter` : mot  $\rightarrow$  arbre  $\rightarrow$  arbre qui prend en argument un mot  $m$  et un arbre de mots  $a$  et qui renvoie l'arbre de mots réduit représentant l'ensemble de mots  $\mathbb{M}(a) \cup m$ . Déterminer la complexité de cette fonction.

► **Question 35** Écrire une fonction `supprimer` : mot  $\rightarrow$  arbre  $\rightarrow$  arbre qui prend en argument un mot  $m$  et un arbre de mots  $a$  et qui renvoie l'arbre de mots réduit représentant l'ensemble de mots  $\mathbb{M}(a) \setminus \{m\}$ . Sans justifier, donner la complexité de cette fonction.

Dans la suite, on remarque que pour un ensemble de mots  $M$ , si on pose  $a'$  l'arbre de mots  $\mathbb{A}(M)$  dans lequel on remplace toutes les étiquettes  $F$  par  $T$ , alors  $\mathbb{M}(a') = \{m[:j] \mid m \in M \text{ et } 0 \leq j \leq \ell(m)\}$  l'ensemble des préfixes d'au moins un mot de  $M$ .

## II.C Arbres des suffixes

Dans cette partie, pour un mot  $m$ , on définit l'**arbre des suffixes de  $m$** , noté  $\mathbb{AS}(m)$ , comme l'arbre de mots réduit représentant l'ensemble des suffixes de  $m$ , c'est-à-dire  $\mathbb{AS}(m) = \mathbb{A}(S(\{m\}))$ .

► **Question 36** Déterminer l'arbre des suffixes de  $m = 1101$ .

► **Question 37** Déterminer en la justifiant  $h(\mathbb{AS}(m))$  pour tout mot  $m$ . En déduire une fonction `retrouver_mot` : arbre  $\rightarrow$  mot qui retrouve le mot  $m$  à partir de l'arbre  $\mathbb{AS}(m)$  en entrée de complexité linéaire en la taille de cet arbre. Il n'est pas demandé de la justifier.

► **Question 38** Montrer que pour un mot  $m$ , l'arbre  $a'$  obtenu en remplaçant toutes les étiquettes des nœuds de  $\mathbb{AS}(m)$  par `true` est l'arbre  $\mathbb{A}(F(m))$ .

► **Question 39** En déduire que la taille de l'arbre  $\mathbb{AS}(m)$  est au moins linéaire et au plus quadratique en  $\ell(m)$ . Montrer que ces bornes sont atteintes, en exhibant pour tout  $\ell \in \mathbb{N}$  un mot  $m$  tel que  $\mathbb{AS}(m)$  ait une taille  $\ell(m) + 1$ , et un mot  $m'$  tel que  $\mathbb{AS}(m')$  ait une taille au moins égale à  $C\ell^2$  pour une constante  $C$  à déterminer. Pour la borne supérieure, on pourra se restreindre aux longueurs paires.

Le second problème dans la Section II est tiré de l'épreuve d'option informatique A des écrits de l'ENS 2021. Les élèves intéressés y trouveront une représentation compacte de l'arbre des facteurs d'un mot binaire, permettant notamment de calculer en temps linéaire en  $\ell(m)$  la longueur du plus long facteur répété de  $m$ , ou encore une méthode pour construire les mots de taille  $\ell$  comprenant le plus grand nombre de facteurs distincts en utilisant les graphes eulériens.