

# Mines 2005 — épreuve d'informatique

## Partie I — PROBLÈME SUR LES AUTOMATES

### Question I.1

On nomme les états de l'automate proposé par le reste modulo 3 de la longueur d'un mot lu par l'automate.

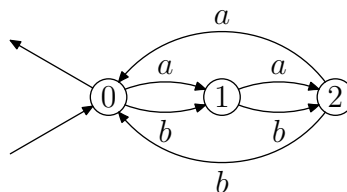


Figure 1 Un automate pour  $L_3$

### Question I.2

Bien sûr une expression régulière admissible est  $((a|b)(a|b)(a|b))^*$ .

### Question I.3

Montrons que  $\Phi(L_3) = \{a^n, n \in \mathbb{N}\}$ .

L'inclusion  $\subset$  est une conséquence immédiate de la définition de  $\Phi$ .

Réciproquement, si  $n \in \mathbb{N}$ , soit  $p$  le reste modulo 3 de  $n$  :  $a^n b^{3-p}$  est clairement dans  $L_3$  donc  $a^n \in \Phi(L_3)$ .

### Question I.4

Soit  $L = \{a^n b^n, n \in \mathbb{N}\}$ . On a  $\Phi(L) = \{a^n, n \in \mathbb{N}\}$  qui est un langage rationnel décrit par  $a^*$  alors que  $L$  n'est pas rationnel (c'est un résultat classique).

### Question I.5

Quitte à l'émonder, on peut supposer l'automate  $\mathcal{A}$  co-accessible.

Construisons l'automate  $\mathcal{B} = \langle \Sigma, Q \cup \{\omega\}, T', I, F' \rangle$ , où  $\omega$  est un nouvel état, de la façon suivante :

- ▷  $\mathcal{B}$  a les mêmes états initiaux que  $\mathcal{A}$  ;
- ▷ ses états sont les états de  $\mathcal{A}$ , auxquels on adjoint un nouvel état  $\omega$  ;
- ▷ ses états finals sont les mêmes  $F' = F$ , auxquels on ajoute les états  $p$  tels qu'il existe dans  $T$  une transition de la forme  $(p, b, q)$  (on remarquera donc que  $\omega$  n'est pas final) ;
- ▷ ses états initiaux sont les mêmes :  $I' = I$  ;
- ▷ toute transition  $(p, a, q) \in T$  est conservée identique dans  $T'$  ;
- ▷ toute transition  $(p, b, q) \in T$  se transforme en  $(p, b, \omega)$  dans  $T'$  ;
- ▷ on ajoute les transitions  $(\omega, a, \omega)$  et  $(\omega, b, \omega)$ , qui font de  $\omega$  un état-puits.

Montrons que  $\mathcal{B}$  reconnaît  $\Phi(L)$ . Soit  $M$  le langage reconnu par  $\mathcal{B}$ .

Observons tout d'abord que toute lecture d'un  $b$  place l'automate dans l'état  $\omega$  d'où on ne peut sortir, et qui est un état-puits. Donc les mots de  $M$  sont tous de la forme  $a^n$ .

Les transitions étiquetées par  $a$  sont reprises telles quelles de l'automate  $\mathcal{A}$ .

Un mot  $a^n$  est donc reconnu si et seulement si il existe un calcul de l'automate  $\mathcal{A}$  étiqueté par  $a^n$ , passant d'un état initial  $p_0$  à un certain état  $p$  qui est final dans  $\mathcal{B}$  : ou bien  $p$  était déjà final dans  $\mathcal{A}$ , et c'est dire que  $a^n \in L$  ; ou bien il existe dans  $\mathcal{A}$  une transition étiquetée par  $b$  qui conduisait à un état  $p'$ . Comme on suppose l'automate  $\mathcal{A}$  co-accessible, il existe un mot  $x$  qui fait passer l'automate  $\mathcal{A}$  de  $p'$  à un état final : c'est dire que  $a^n b x \in L$ . On a bien retrouvé la caractérisation de  $\Phi(L)$ .

Remarquons qu'on a montré l'implication :  $L$  rationnel  $\Rightarrow \Phi(L)$  rationnel.

### Question I.6

Soit  $M_3$  le langage constitué des mots dont la longueur est de la forme  $3p + 1$  avec  $p$  entier et qui contiennent au moins un  $b$ .

Tout mot de  $M_3$  est donc de la forme  $xyb$  où  $|x| + |y| = 3p$  donc  $xy \in L_3$  : on a prouvé  $M_3 \subset \Gamma(L_3)$ .

Réciproquement, si  $v \in \Gamma(L_3)$ , il s'écrit  $v = xyb$  avec  $xy \in L_3$  donc  $|v| = 1 + |xy|$  est bien de la forme  $3p + 1$ .

**Question I.7**

On construit un nouvel automate  $\mathcal{A}' = \langle \Sigma, Q \times \{0, 1\}, T', I \times \{0\}, F \times \{1\} \rangle$  de la façon suivante :

- ▷ ses états sont des couples  $(q, 0)$  ou  $(q, 1)$  où  $q \in Q$  : la deuxième composante vaut 1 si l'insertion d'un **b** a été réalisée ;
- ▷ ses états initiaux sont les couples  $(q, 0)$  où  $q \in I$ , ses états finals les  $(q, 1)$  où  $q \in F$  ;
- ▷ pour toute transition  $(p, c, q) \in T$ , on construit les transitions  $((p, 0), c, (q, 0))$  et  $((p, 1), c, (q, 1))$  ;
- ▷ pour tout état  $p \in Q$ , on ajoute une transition  $((p, 0), b, (p, 1))$  marquant l'insertion d'un **b**.

Il est alors clair que  $\mathcal{A}'$  reconnaît  $\Gamma(L)$ .

**Question I.8**

Les mots de  $\Psi(L_3)$  ont évidemment tous une longueur de la forme  $3p + 2$ , où  $p \geq 0$ . Inversement tout mot  $u$  ayant une telle longueur est bien dans  $\Psi(L_3)$  car il provient de l'effacement du **b** initial dans le mot  $bu \in L_3$ .

**Question I.9**

Soit  $L = \{b^n u, n \geq 0, |u|_a \geq |u|_b\}$  le langage constitué des mots constitués d'un nombre quelconque (éventuellement nul) de **b** initiaux suivis d'un mot contenant plus de **a** que de **b**.

I.9.a Montrons que  $L$  n'est pas un langage rationnel.

En effet, soit sinon  $\alpha$  un automate déterministe reconnaissant  $L$ , sans état inutile (c'est-à-dire que tout état est accessible et co-accessible). Soit  $p_n$  l'état auquel on aboutit depuis l'état initial  $p_0$  à la lecture de  $a^n$  ( $p_n$  est bien défini car  $a^n \in L$ ).

Comme  $\alpha$  a un nombre fini d'états, il existe  $k < \ell$  tels que  $p_k = p_\ell$ . Alors  $a^\ell b^\ell$  étant dans  $L$ , il existe un chemin dans  $\alpha$ , étiqueté par  $b^\ell$ , le faisant passer de  $p_\ell$  à un état final  $q$ . Alors  $a^k b^\ell$  sera accepté par  $\alpha$ , qui passe de  $p_0$  à  $p_k = p_\ell$  en lisant  $a^k$ , puis de  $p_k = p_\ell$  à  $q$  en lisant  $b^\ell$ . C'est la contradiction espérée puisque  $a^k b^\ell \notin L$ .

I.9.b Montrons que  $\Psi(L) \subset L$ .

En effet, supprimer la première occurrence de **b** dans  $b^n u$  amène à  $b^{n-1} u$  si  $n \geq 1$ , et il s'agit bien d'un mot de  $L$  ; et si  $n = 0$ , la suppression du premier **b** de  $u$  amène à un mot  $v$  tel que  $|v|_b = |u|_b - 1 < |u|_b \leq |u|_a = |v|_a$  donc  $v \in L$ .

I.9.c Montrons que  $\Psi(L) \supset L$ .

En effet, le mot  $b^n u$  avec  $n \geq 0$  et  $|u|_a \geq |u|_b$  s'obtient en effaçant la première occurrence de **b** dans  $b^{n+1} u$  qui appartient bien à  $L$ .

**Question I.10**

On propose un automate  $\mathcal{A}'$  pouvant comporter des  $\varepsilon$ -transitions, ce qui ne devrait pas poser de problème : on sait comment les éliminer si on le souhaite.

L'automate  $\mathcal{A}' = \langle \Sigma, Q \times \{0, 1\}, T', I', F' \rangle$  est défini de la façon suivante :

- ▷ les états de  $\mathcal{A}'$  sont les états de la forme  $(p, 0)$  ou  $(p, 1)$  où  $p$  est un état de  $\mathcal{A}$  ;
- ▷ les états initiaux de  $\mathcal{A}'$  sont les états  $(p, 0)$  où  $p \in I$  ;
- ▷ les états finals de  $\mathcal{A}'$  sont les états  $(p, 1)$  où  $p \in F$  ;
- ▷ pour chaque transition  $(p, a, q) \in T$ , on ajoute à  $T'$  les transitions  $((p, 0), a, (q, 0))$  et  $((p, 1), a, (q, 1))$  ;
- ▷ pour chaque transition  $(p, b, q) \in T$ , on ajoute à  $T'$  les transitions  $((p, 0), \varepsilon, (q, 1))$  (marquant la suppression d'une première occurrence de **b**) mais aussi une transition  $((p, 1), b, (q, 1))$  (on garde les occurrences suivantes).

# Problème d'algorithmique, logique et programmation

## Partie II — DESCENDANTS D'UN SOMMET D'UN GRAPHE ORIENTÉ

### Question II.1

On marque 1. Ses fils non marqués sont 0 et 2. Sont marqués :  $\{1\}$ .

On marque 0. Son seul fils non marqué est 5. Sont marqués :  $\{0, 1\}$ .

On marque 5. Il n'a pas de fils non marqué. Sont marqués :  $\{0, 1, 5\}$ . Fin de `marque(5)`. Fin de `marque(0)`.

On marque 2. Son seul fils non marqué est 6. Sont marqués  $\{0, 1, 2, 5\}$ .

On marque 6. Il n'a pas de fils non marqué. Sont marqués  $\{0, 1, 2, 5, 6\}$ . Fin de `marque(6)`. Fin de `marque(2)`. Fin de `marque(1)`.

### Question II.2

---

```
type Graphe = { nb_sommets: int ; fils: int list vect } ;;

let Gex = { nb_sommets = 7 ; fils = [| [5]; [0;2]; [6]; []; [0;1]; [0;1] ; [] |] } ;;

let calcul_descendants G r =
  let marques = make_vect G.nb_sommets false in
  let rec marque s =
    marques.(s) <- true ;
    do_list (function x -> if not marques.(x) then marque x) G.fils.(s)
  in
  marque r ;
  marques ;;
```

---

### Programme 1 La fonction `calcul_descendants`

*Remarque :* On a utilisé la fonction `do_list`, qui fait partie de la bibliothèque standard Caml, mais que chaque candidat doit savoir fournir le code dans sa copie. Il en est de même pour d'autres fonctions habituelles ci-dessous, comme `it_list`, `mem` ou `subtract`.

### Question II.3

Pour tout ensemble  $X$  de sommets et tout entier  $n \in \mathbb{N}$ , définissons  $V_0(X) = X$  puis  $V_{n+1}(X)$  comme la réunion de  $V_n(X)$  et de l'ensemble des fils des sommets de  $V_n(X)$ . On dira que  $V_n(X)$  est l'ensemble des descendants de  $n$ -ème génération (au plus) des éléments de  $X$ .

Remarquons que, dans notre définition,  $V_n(X) \subset V_{n+1}(X)$ , donc que la suite  $(V_n(X))$  est croissante pour l'inclusion, et, comme le graphe est fini, elle est stationnaire. Notons  $V_\infty(X)$  la limite, et observons qu'il s'agit de l'ensemble de tous les descendants des sommets qui figurent dans  $X$ .

Venons en à la preuve demandée.

II.3.a L'algorithme termine. Chaque appel récursif de `marque` marque en effet un **nouveau** sommet, et l'ensemble des sommets est fini.

II.3.b L'algorithme ne marque que des descendants de  $s$ . C'est évident, par récurrence sur le nombre d'appels à `marque`.

II.3.c L'algorithme marque tous les descendants de  $s$ . En effet, si on suppose que l'algorithme marque tous les descendants de  $n$ -ème génération, appliqué à  $s$ , il marquera  $s$  et les descendants de  $n$ -ème génération des fils de  $s$ , donc finalement il aura bien marqué tous les descendants de  $(n+1)$ -ème génération de  $s$ . Or l'algorithme marque bien sûr  $s$  lui-même, donc  $V_0(\{s\})$  donc, par récurrence,  $V_n(\{s\})$  pour tout entier  $n$ .

### Question II.4

L'algorithme tourne en  $O(\min(n, m))$ , car chaque arête n'est utilisée qu'une fois au plus et chaque sommet marqué une fois au plus.

Dans le cas d'un graphe linéaire (où  $m = n - 1$ ), la complexité vaut  $m$  ; dans le cas d'un graphe complet (où  $m = n(n - 1)$ ), elle vaut  $n$ .

### Partie III — UN PROBLÈME DE SATISFIABILITÉ

#### Question III.1

On dresse la table de vérité de la formule.

$x$	$y$	$z$	$x \vee y$	$\bar{x} \vee z$	$\bar{y} \vee z$	$\bar{x} \vee \bar{z}$	$F_1$
V	V	V	V	V	V	F	F
V	V	F	V	F	F	V	F
V	F	V	V	V	V	F	F
V	F	F	V	F	V	V	F
F	V	V	V	V	V	V	V
F	V	F	V	V	F	V	F
F	F	V	F	V	V	V	F
F	F	F	F	V	V	V	F

**Table 1** Table de vérité de  $F_1$

$F_1$  est satisfiable :  $(x, y, z) = (F, V, V)$  est un contexte solution.

#### Question III.2

On dresse la table de vérité de la formule.

$x$	$y$	$z$	$t$	$x \vee y$	$\bar{x} \vee z$	$\bar{y} \vee \bar{z}$	$t \vee \bar{z}$	$y \vee \bar{t}$	$x \vee \bar{y}$	$F_2$
V	V	V	V	V	V	F	V	V	V	F
V	V	V	F	V	V	F	F	V	V	F
V	V	F	V	V	F	V	V	V	V	F
V	V	F	F	V	F	V	V	V	V	F
V	F	V	V	V	V	V	V	F	V	F
V	F	V	F	V	V	V	F	V	V	F
V	F	F	V	V	F	V	V	F	V	F
V	F	F	F	V	F	V	V	V	V	F
F	V	V	V	V	V	F	V	V	F	F
F	V	V	F	V	V	F	F	V	F	F
F	V	F	V	V	V	V	V	V	F	F
F	V	F	V	V	V	V	V	V	F	F
F	F	V	V	F	V	V	V	F	V	F
F	F	V	F	F	V	V	F	V	V	F
F	F	F	V	F	V	V	V	F	V	F
F	F	F	F	F	V	V	V	V	V	F

**Table 2** Table de vérité de  $F_2$

$F_2$  n'est pas satisfiable.

#### Question III.3

La phrase *La personne X dit : Z est fiable* se traduit par  $x \Rightarrow z$  ou encore  $\bar{x} \vee z$ .

La deuxième phrase par  $y \Rightarrow (\bar{z} \wedge t)$  ou encore  $(\bar{y} \vee \bar{z}) \wedge (\bar{y} \vee t)$ .

La troisième phrase par  $z \Rightarrow (y \wedge t)$  ou encore  $(\bar{z} \vee y) \wedge (\bar{z} \vee t)$ .

La quatrième phrase par  $t \Rightarrow (\bar{x} \wedge y)$  ou encore  $(\bar{t} \vee \bar{x}) \wedge (\bar{t} \vee y)$ .

La cinquième peut se réécrire  $x \vee y$  et la sixième  $z \vee t$ .

Finalement on obtient la formule  $F_3 = (\bar{x} \vee z) \wedge (\bar{y} \vee \bar{z}) \wedge (\bar{y} \vee t) \wedge (\bar{z} \vee y) \wedge (\bar{z} \vee t) \wedge (\bar{t} \vee \bar{x}) \wedge (\bar{t} \vee y) \wedge (x \vee y) \wedge (z \vee t)$ .

#### Question III.4

On laisse le soin au lecteur de remplir la table de vérité de  $F_3$  et de constater que le seul contexte où elle soit satisfaite est :  $X$  et  $Z$  ne sont pas fiables,  $Y$  et  $T$  sont fiables (ou encore  $(x, y, z, t) = (F, V, F, V)$ ).

### Question III.5

---

```
type Clause = { a: int ; b: int } ;;

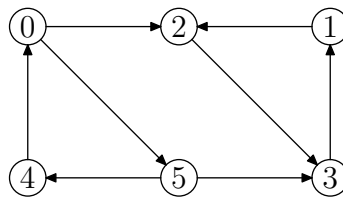
type Formule = { nb_var: int ; clauses: Clause list } ;;

let barre alpha p =
  if alpha < p then alpha + p else alpha - p ;;
```

---

**Programme 2** La fonction barre

### Question III.6



**Figure 2** Le graphe  $G(F_1)$

### Question III.7

L'énoncé garantit qu'il n'y a pas deux fois la même clause dans la formule proposée, ce qui permet d'écrire la version suivante de **transformer** :

---

```
let transformer F =
  let p = F.nb_var in
  let G = { nb_sommets = 2*p ; fils = make_vect (2*p) [] } in
  let ajoute_clause { a = x ; b = y } =
    let x' = barre x p and y' = barre y p in
    G.fils.(x') <- y :: G.fils.(x') ;
    G.fils.(y') <- x :: G.fils.(y')
  in
  do_list ajoute_clause F.clauses ;
  G ;;
```

---

**Programme 3** La fonction transformer

### Question III.8

La complexité de la fonction **transformer** est clairement  $O(p + q)$ .

### Question III.9

On a écrit une fonctionnelle **filtre** telle que l'appel **filtre prédicat l** retourne la liste constituée des éléments de **l** : 'a list qui satisfont **prédicat** : 'a -> bool. On écrit alors le programme 4, page 6, qui ne pose pas de difficulté particulière.

### Question III.10

La complexité de la fonction **retirer** est clairement  $O(q)$ .

### Question III.11

Soit  $\alpha_0 = \alpha \rightarrow \alpha_1 \rightarrow \dots \rightarrow \alpha_k = \beta$  un chemin dans le graphe  $G(F)$ . La présence, pour  $0 \leq j < k$ , de l'arc  $\alpha_j \rightarrow \alpha_{j+1}$  montre que la formule  $F$  contient la clause  $\bar{\alpha}_j \vee \alpha_{j+1}$  qui est logiquement équivalente à  $\alpha_j \Rightarrow \alpha_{j+1}$ . Finalement  $F$  entraîne la chaîne d'implications  $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_k$  donc aussi  $\alpha \Rightarrow \beta$ .

---

```

let rec filtre prédicat = function
| [] -> []
| t :: q -> if prédicat t then t :: (filtre prédicat q) else filtre prédicat q ;;

let retirer F alpha =
  let p = F.nb_var in
  let alpha' = barre alpha p in
  let contrôle { a = x ; b = y } = x <> alpha && x <> alpha' && y <> alpha && y <> alpha'
  in
  { nb_var = p ; clauses = filtre contrôle F.clauses } ;;

```

---

#### Programme 4 La fonction retirer

##### Question III.12

Dans la construction de  $G(F)$ , la création de l'arête  $\alpha_j \rightarrow \alpha_{j+1}$  s'accompagne toujours de la création de l'arête  $\overline{\alpha_{j+1}} \rightarrow \overline{\alpha_j}$ , donc, de façon évidente, l'existence d'un chemin de  $\alpha$  à  $\beta$  implique celle d'un chemin de  $\beta$  vers  $\alpha$ .

##### Question III.13

L'implication  $\alpha \Rightarrow \bar{\alpha}$  est équivalente à  $\bar{\alpha}$ , donc l'existence dans  $G(F)$  d'un chemin de  $\alpha$  vers  $\bar{\alpha}$  montre que  $\bar{\alpha}$  doit être vraie si l'on veut satisfaire  $F$ . De même l'existence d'un chemin de  $\bar{\alpha}$  vers  $\alpha$  montre que  $\alpha$  doit être vraie si l'on veut satisfaire  $F$ . Finalement, si ces deux chemins existent simultanément, la formule  $F$  ne peut être satisfaite : c'est une contradiction.

##### Question III.14

Raisonnons par l'absurde : si  $\beta$  et  $\bar{\beta}$  sont des descendants de  $\alpha$ , c'est qu'il existe un chemin de  $\alpha$  vers  $\beta$  et un chemin de  $\alpha$  vers  $\bar{\beta}$ , donc aussi de  $\beta$  vers  $\bar{\alpha}$  d'après III.12. Mais alors, il y aurait un chemin de  $\alpha$  vers  $\bar{\alpha}$ , ce qu'on a exclu.

##### Question III.15

L'énoncé aurait dû préciser que dans le cas où toutes les clauses de  $F$  sont effacées, on choisit  $F' = V$ . Supposons  $F$  satisfiable : il existe un  $p$ -uplet de valeurs booléennes  $(b_0, \dots, b_{p-1})$  qui correspond à un contexte satisfaisant  $F$ . Autrement dit, si on remplace la variable numéro  $k$  par le booléen  $b_k$  et son complément (la variable numéro  $p + k$ , donc) par  $\bar{b}_k$ , la formule  $F$  prend la valeur VRAI, donc chacune des clauses  $\beta \vee \gamma$  également. Mais alors, le même contexte satisfait nécessairement  $F'$  également, puisqu'elle comporte moins de clauses.

La réciproque est plus difficile. Supposons donc que  $F'$  est satisfiable, et soit  $\omega = (b_0, \dots, b_{p-1})$  un contexte qui la satisfait : notons que dans ce contexte, les valeurs booléennes qui correspondent aux variables n'apparaissant pas dans  $F'$  sont indifférentes.

On construit un nouveau contexte  $\Omega$ , dont on va montrer qu'il satisfait  $F$ , de la façon suivante :

- ▷ À toute variable apparaissant (elle ou son complément) dans  $F'$ , on associe la valeur booléenne que lui associe le contexte  $\omega$ .
- ▷ Si  $\beta$  est un littéral qui apparaît dans  $F$  mais pas dans  $F'$ , c'est que  $\beta$  ou  $\bar{\beta}$  est un descendant de  $\alpha$  dans  $G(F)$ . Mais alors, la question précédente certifie : c'est ou bien  $\beta$ , ou bien  $\bar{\beta}$  qui est descendant de  $\alpha$  dans  $G(F)$ , pas les deux à la fois. Si c'est  $\beta$ , on lui associe la valeur VRAI, si c'est  $\bar{\beta}$ , on associe la valeur FAUX à  $\beta$  (ou VRAI à  $\bar{\beta}$ ).

En particulier, on associe à  $\alpha$  la valeur VRAI.

Montrons maintenant que le contexte  $\Omega$  satisfait bien  $F$ .

Soit  $\beta \vee \gamma$  une clause de  $F$ . Si elle apparaît dans  $F'$ , le contexte  $\Omega$  prolongeant le contexte  $\omega$ , la clause est bien satisfaite.

Sinon, c'est qu'un des littéraux ou son complémenté est descendant de  $\alpha$  dans  $G(F)$ , par exemple  $\beta$  ou  $\bar{\beta}$ .

Si  $\beta$  est descendant de  $\alpha$ , le contexte  $\Omega$  lui associe la valeur VRAI, donc la clause  $\beta \vee \gamma$  est bien satisfaite. Sinon,  $\bar{\beta}$  est descendant de  $\alpha$  et le contexte  $\Omega$  associe la valeur FAUX à  $\beta$ . Mais le graphe  $G(F)$  contient un arc de  $\bar{\beta}$  vers  $\gamma$  qui est donc aussi un descendant de  $\alpha$  : c'est dire que le contexte  $\Omega$  associe la valeur VRAI à  $\gamma$  et finalement la clause est encore satisfaite.

**Question III.16**

Soit  $F$  une formule sous forme NC2 telle que, pour tout littéral  $\alpha$ , il n'existe pas dans  $G(F)$  à la fois un chemin de  $\alpha$  à  $\bar{\alpha}$  et un chemin de  $\bar{\alpha}$  à  $\alpha$ .

On applique autant de fois que possible les résultats de la question précédente, en supprimant des clauses qui transforment  $F$  en  $F'_1$ , puis  $F'_2$ , etc, jusqu'à une formule  $F'_k$  pour laquelle l'opération est impossible. (Le nombre d'étapes est fini puisque le nombre de clauses qui subsistent décroît strictement.) Notons que  $F$  est satisfiable si et seulement si  $F'_k$  est satisfiable.

Montrons que  $F'_k$  est nécessairement réduite à VRAI (on a supprimé toutes les clauses), donc évidemment satisfiable, ce qui prouvera que  $F$  est satisfiable.

En effet, si un littéral  $\alpha$  apparaissait encore dans  $F'_k$ , comme il n'est plus possible d'appliquer les résultats de la question précédente, c'est qu'il existe un chemin de  $\alpha$  à  $\bar{\alpha}$  dans  $G(F'_k)$ . Mais c'est aussi qu'il existe un chemin de  $\bar{\alpha}$  vers  $\alpha$ . Finalement dans  $G(F'_k)$  et donc aussi dans  $G(F)$ , il doit y avoir un chemin de  $\alpha$  vers  $\bar{\alpha}$  et un chemin de  $\bar{\alpha}$  vers  $\alpha$ , ce qu'exclut l'hypothèse.

**Question III.17**

Voici une description informelle de l'algorithme proposé, qui construit un contexte satisfaisant la formule logique  $F$  sous forme NC2 quand c'est possible :

1. si  $F$  est VRAI,  $F$  est satisfiable et tout contexte est une réponse correcte ;
2. sinon, s'il existe un littéral  $\alpha$  tel qu'il n'y a pas de chemin dans  $G(F)$  de  $\alpha$  vers  $\bar{\alpha}$  ;
  - a. calculer la liste  $L$  des descendants de  $\alpha$  dans  $G(F)$  ;
  - b. remplacer  $F$  par  $F'$ , obtenue en retirant de  $F$  toutes les clauses contenant un littéral figurant lui ou son complémenté dans  $L$  ;
  - c. pour chaque littéral  $\beta$  apparaissant dans  $L$ , associer dans le contexte en construction la valeur VRAI aux littéraux positifs et la valeur FAUX aux littéraux négatifs ;
  - d. continuer à l'étape 1 ;
3. sinon, c'est que pour tout littéral  $\alpha$ , il existe dans  $G(F)$  un chemin de  $\alpha$  vers  $\bar{\alpha}$ , et  $F$  n'est pas satisfiable.

La seule boucle du programme (étape 2) termine à coup sûr puisque le nombre de clauses décroît strictement...

### Question III.18

Nous commençons, dans le programme 5, par deux fonctions auxiliaires : **intervalle**  $i$   $j$  renvoie la liste des entiers de  $i$  à  $j$  (bornes incluses), et **descendants**  $G$   $r$  renvoie la liste des littéraux descendants du littéral  $r$  dans le graphe  $G$

---

```
let rec intervalle i j =
  if i > j then [] else i :: (intervalle (i+1) j) ;;

let descendants G r =
  let m = calcul_descendants G r in
  filtre (function i -> m.(i)) (intervalle 0 (G.nb_sommets - 1)) ;;
```

---

#### Programme 5 Deux fonctions auxiliaires

Il a semblé plus logique qu'un contexte ne fournisse que les valeurs logiques des seuls littéraux positifs ! En outre, la fonction **satisfiable** présentée dans le programme 6 renvoie le contexte qui satisfait la formule quand c'est possible et déclenche une exception si la formule n'est pas satisfiable. La fonction **candidat** permet de trouver (s'il existe) le littéral  $\alpha$  tel que  $\bar{\alpha}$  ne soit pas un de ses descendants dans le graphe courant. La variable **littéraux** contient la liste des littéraux qui n'ont pas encore été effacés de la formule initiale. La fonction **littéraux\_de** permet de calculer la liste des littéraux qui apparaissent dans une formule.

---

```
let littéraux_de F =
  it_list (fun l c ->
    if mem c.a l then if mem c.b l then l else c.b :: l
    else if mem c.b l then c.a :: l else c.a :: c.b :: l) [] F.clauses ;;

let satisfiable F =
  let contexte = make_vect F.nb_var true
  and p = F.nb_var
  in
  let rec candidat gf = function
    | [] -> failwith "Formule non satisfiable"
    | i :: q -> if mem (barre i p) (descendants gf i) then candidat gf q else i
  in
  let rec boucle littéraux f =
    if f.clauses = [] then contexte
    else
      begin
        let gf = transformer f in
        let alpha = candidat gf littéraux in
        let l = descendants gf alpha in
        do_list (function i -> if i >= p then contexte.(i - p) <- false) l ;
        boucle (subtract littéraux l) (it_list retirer f l)
      end
  in
  boucle (littéraux_de F) F ;;
```

---

#### Programme 6 La fonction satisfiable

On pourrait éviter de recalculer le graphe au fur et à mesure que la formule se simplifie. Mais d'une part l'énoncé a invité à écrire au préalable la fonction **transformer** et a limité la portée de la fonction **retirer** aux formules (et non à leur graphe), d'autre part la formule se simplifie très vite et il n'est pas sûr que la modification du graphe soit plus rapide que son recalcul...



**Question III.19**

Remarquons que le nombre d'arêtes du graphe  $G(F)$  est un  $O(q)$ .

Chaque passage dans la boucle voit décroître le nombre de clauses : il y a donc au plus  $q$  passages dans la boucle.

Chaque passage demande le calcul du graphe (appel à **transformer**), en  $O(p + q)$ , la recherche d'un candidat (il y a  $O(p)$  littéraux à examiner, le calcul de leurs descendants coûte  $O(q)$ , ce qui fait un total de  $O(pq)$ ).

Au total l'algorithme proposé tourne en  $O(pq^2)$  ce qui reste polynomial, alors que le passage par une table de vérité entraînerait un coût de  $O(2^p q)$  ( $2^p$  lignes et  $q$  colonnes dans le tableau).