ENSI 2011 — Option informatique

Partie I — Logique et calcul des propositions

Question I.1
Les règles de politesse s'énoncent $(A_1 \Rightarrow B_1) \wedge (B_1 \Rightarrow C_1)$.

Question I.2
L'affirmation de A est $A_1 = D \vee G$; de même $B_1 = \neg D$ et $C_1 = P \wedge G$.

Question I.3
La première conversation conduit à la proposition $X = ((D \vee G) \Rightarrow \neg D) \wedge (\neg D \Rightarrow (P \wedge G))$.
On sait qu'on a l'équivalence $(p \Rightarrow q) \equiv (\neg p \vee q)$, ce qui permet de réécrire X sous la forme

$$\begin{aligned} X &\equiv (\neg(D \vee G) \vee \neg D) \wedge (D \vee (P \wedge G)) \\ &\equiv ((\neg D \wedge \neg G) \vee \neg D) \wedge (D \vee P) \wedge (D \vee G) \\ &\equiv \neg D \wedge (D \vee P) \wedge (D \vee G). \end{aligned}$$

On a trouvé la forme normale conjonctive : il est maintenant facile de conclure que D est fausse, P vraie et G vraie. Autrement dit, un kjalt n'a pas de dard, mais est doté de pinces et de griffes.

Question I.4
Les règles de politesses s'écrivent ici $Y = (A_2 \Rightarrow B_2) \wedge (B_2 \Rightarrow A_3) \wedge (A_2 \Leftrightarrow A_3)$ (car A est constant dans le mensonge ou la vérité).

Question I.5
Bien sûr, $A_2 = M \wedge \neg J$, $B_2 = \neg V$ et $A_3 = V \Rightarrow J$. (En fait, l'affirmation A_2 n'est pas très claire : doit-on plutôt l'interpréter comme *s'il est mauve, il n'est pas jaune* ? cela s'écrirait $A_2 = M \Rightarrow \neg J$.)

Question I.6
On dresse une table de vérité.

J	M	V	A_2	B_2	A_3	$A_2 \Rightarrow B_2$	$B_2 \Rightarrow A_3$	$A_2 \Leftrightarrow A_3$	Y
1	1	1	0	0	1	1	1	0	0
1	1	0	0	1	1	1	1	0	0
1	0	1	0	0	1	1	1	0	0
1	0	0	0	1	1	1	1	0	0
0	1	1	1	0	0	0	1	0	0
0	1	0	1	1	1	1	1	1	1
0	0	1	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1	0	0

Tableau 1 la table de vérité de Y

On en déduit qu'un lyop est ou bien mauve mais ni jaune ni vert, ou bien vert mais ni jaune ni mauve. Dans le cas où $A_2 = M \Rightarrow \neg J$, on trouve trois solutions : jaune, non mauve et non vert ; mauve, non jaune et non vert ; ni jaune, ni mauve, ni vert.

Partie II — Automates et langages

Question II.1
L'automate $\mathcal{E}$ reconnaît le langage L décrit par l'expression régulière `ab(cb|a)*`.

Question II.2

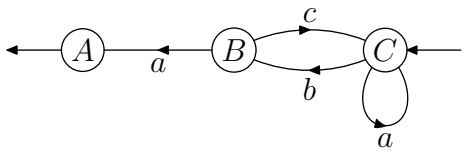

Figure 1 l'automate $\mathcal{E}^{-1}$

Question II.3
Une expression régulière dénotant le langage reconnu par $\mathcal{E}^{-1}$ est alors `(bc|a)*ba`.
On observe que le langage en question est le *miroir* $\bar{L}$ du langage L reconnu par $\mathcal{E}$. (L'énoncé, dans la définition II.3, parle plutôt *d'inverse* du langage L.)

Question II.4
Montrons, par récurrence sur la longueur du mot m, que

$$(P) \qquad \forall x \in X, \forall (o,d) \in Q^2, \quad d \in \delta^\star(o, mx) \iff \exists q \in Q, q \in \delta^\star(o,m) \wedge d \in \delta(q,x).$$

Comme $\delta^\star(o,\Lambda) = \{o\}$, dans le cas où $m = \Lambda$, $d \in \delta^\star(o,x) \Leftrightarrow d \in \delta(o,x)$ est vrai d'après la définition de $\delta^\star$. Cela garantit (P) quand $|m| = 0$.

Supposons la propriété (P) vérifiée pour tout mot m de longueur inférieure ou égale à un entier $n \geqslant 0$.
Soit m un mot de longueur $n+1$, qu'on écrit $m = ym'$ où $|m'| \leqslant n$ et $y \in X$, soit $x \in X$ et $(o,d) \in Q^2$.
On a : $d \in \delta^\star(o,mx) \iff d \in \delta^\star(o,ym'x) \iff \exists q_1 \in Q, q_1 \in \delta(o,y) \wedge d \in \delta^\star(q_1, m'x)$ d'après la définition de $\delta^\star$. On applique l'hypothèse de récurrence pour écrire :

$$\begin{aligned} d \in \delta^\star(o,mx) &\iff \exists (q_1,q_2) \in Q^2, q_1 \in \delta(o,y) \wedge q_2 \in \delta^\star(q_1,m') \wedge d \in \delta(q_2,x) \\ &\iff \exists q_2 \in Q, q_2 \in \delta^\star(o,ym') \wedge d \in \delta(q_2,x) \\ &\iff \exists q_2 \in Q, q_2 \in \delta^\star(o,m) \wedge d \in \delta(q_2,x), \end{aligned}$$

où on reconnaît (P) pour le mot m de longueur $n+1$, ce qui enclenche la récurrence. (Le passage de la première à la deuxième ligne utilise la définition de $\delta^\star$.)

Question II.5
On procède là encore par récurrence sur la longueur du mot m.

La propriété est triviale quand $m = \Lambda$, car elle se réécrit $d \in \{o\} \iff o \in \{d\}$.

Supposant la propriété vérifiée pour un mot m, on montre qu'elle est également vraie pour un mot de la forme xm avec $x \in X$.
Or, pour tous états o et d, on dispose bien de :

$$\begin{aligned} d \in \delta^\star(o,mx) &\iff \exists q \in Q, q \in \delta^\star(o,m) \wedge d \in \delta(q,x) \\ &\iff \exists q \in Q, o \in \delta^{-1\star}(q,m^{-1}) \wedge q \in \delta^{-1}(d,x) \\ &\iff o \in \delta^{-1\star}(d, xm^{-1}) \\ &\iff o \in \delta^{-1\star}(d, (mx)^{-1}). \end{aligned}$$

(La première équivalence provient de II.4, la seconde de l'hypothèse de récurrence, la troisième de la définition du I.3, la dernière de la définition II.3.)
La récurrence s'enclenche…

Question II.6
Tout cela se résume à dire que $\mathcal{A}$ et $\mathcal{A}^{-1}$ reconnaissent des langages miroirs l'un de l'autre.

Partie III — Algorithmique et programmation en Caml

Préliminaire : arbre binaire d'entiers

Question III.1

On a bien sûr $|a| = \begin{cases} -1, & \text{si } a = \emptyset \text{ ;} \\ 1 + \max(|\mathcal{G}(a)|, |\mathcal{D}(a)|), & \text{sinon.} \end{cases}$

Question III.2

Pour un jeu fixé de n étiquettes, l'arbre de profondeur minimale est un arbre bien tassé (un arbre dont toutes les feuilles sont à profondeur $|a|$ ou $|a| - 1$) ; l'arbre de profondeur maximale est l'arbre-ligne où par exemple tous les fils gauches sont vides, qui est de profondeur $|a| = n - 1$.

Question III.3

Il est évident que si il y a k nœuds à un certain niveau, il y en a $2k$ au niveau suivant. Comme il y a 1 nœud (la racine) au niveau 0, il y en a bien 2^p à profondeur p.

Question III.4

Alors $n = \sum_{k=0}^{p} 2^k = 2^{p+1} - 1$.

Question III.5

Autrement dit : $p = \lg(n+1) - 1$.

Exercice : arbre binaire de recherche

Question III.6

La fonction auxiliaire `aux1` appelle successivement `ajouter` pour chaque élément de la liste `s`, insérant dans un arbre initialement vide les éléments de la liste.
La fonction auxiliaire `aux2` liste alors récursivement l'arbre obtenu, en ordre symétrique.
On dessine ci-dessous les arbres binaires construits par insertions successives.

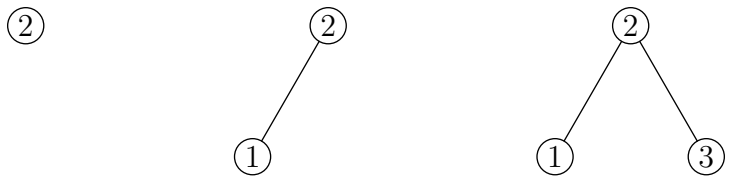

Figure 2 les insertions successives

Question III.7

On démontre par récurrence sur la taille de l'arbre de recherche p que si r est le résultat de l'appel `ajouter e p`, r est également un arbre de recherche dont l'ensemble (avec répétition) d'étiquettes est égal à celui de p plus l'étiquette e.
Si p est l'arbre vide, alors r est un arbre réduit à sa racine d'étiquette e, qui est bien de recherche.
Sinon, soit x la racine de l'arbre de recherche p.

- ▷ Si $x = e$, l'arbre r a pour fils gauche un arbre r' de racine x, sans fils droit et de fils gauche $\mathcal{G}(p)$. Mais les éléments de $\mathcal{G}(p)$ sont inférieurs ou égaux à $x = e$, donc r' est bien de recherche et tous ses éléments sont inférieurs ou égaux à e. r a pour racine e, pour fils gauche r' dont on vient de parler et pour fils droit $\mathcal{D}(p)$ qui est bien de recherche et dont tous les éléments sont supérieurs strictement à $x = e$: on a bien prouvé que r est de recherche et que ses étiquettes sont e et les étiquettes qui figuraient déjà dans p.
- ▷ Si $e < x$, la racine de r est e, strictement inférieure à tous les éléments de son fils droit qui est l'arbre de recherche $\mathcal{D}(p)$. Le fils gauche de r est l'arbre obtenu par l'appel `ajouter e g`, où $g = \mathcal{G}(p)$. Par hypothèse de récurrence, cet arbre est de recherche, et ses étiquettes sont e et les étiquettes de $\mathcal{G}(p)$, toutes inférieures ou égales à x. Bref, r est de recherche et ses étiquettes sont bien e et les étiquettes qui figuraient déjà dans p.
- ▷ Si $e > x$, la racine est e, supérieure ou égale à tous les éléments de son fils gauche qui est l'arbre de recherche $\mathcal{G}(p)$. Le fils droit de r est l'arbre obtenu par l'appel `ajouter e d`, où $d = \mathcal{D}(p)$. Par hypothèse de récurrence, cet arbre est de recherche, et ses étiquettes sont e et les étiquettes de $\mathcal{D}(p)$, toutes strictement supérieures à x. Bref, r est de recherche et ses étiquettes sont bien e et les étiquettes qui figuraient déjà dans p.

Question III.8
Le fait que les séquences p et r soient de même longueur $m = n$ et que les ensembles d'étiquettes sont égaux (la propriété b) découle de la démonstration précédente.
La propriété c traduit simplement que dans le parcours symétrique de l'arbre réalisé par `aux2`, on liste d'abord les étiquettes du fils gauche, avant la racine, puis les étiquettes du fils droit, ce qui pour un arbre de recherche, revient à lister les étiquettes en ordre croissant.

Question III.9
Soit n le nombre d'étiquettes à trier.
L'appel `aux1 l r` réalise $|\ell|$ appels de `ajouter`.
`ajouter v a` termine car les appels récursifs se font toujours sur des arbres binaires de taille strictement plus petite que la taille de a.
`aux2 a` termine pour la même raison : la taille de l'arbre argument diminue strictement à chaque appel récursif.

Question III.10
Le coût d'un appel `ajouter v a` est la longueur de la branche suivie lors des appels récursifs. Le coût de `aux1 l r` est celui de $|\ell|$ appels de `ajouter`. `aux2 a` visite tous les nœuds de l'arbre et coûte donc le nombre de ces nœuds.
Le cas le pire est donc celui où la profondeur de l'arbre construit est maximale : c'est le cas quand la liste s est une liste strictement monotone d'entiers. Dans ce cas, le coût total est $O(n^2)$.
Le cas le plus favorable est celui où l'arbre construit est de profondeur minimale, donc quasi-complet à chaque instant : le coût total est alors $O(n \lg n)$. Comme l'insertion se fait aux feuilles, on peut par exemple, pour $n = 15$, considérer la séquence $s = (15, 13, 14, 3, 5, 11, 1, 2, 4, 6, 9, 10, 12)$.

Problème : représentation de systèmes creux

Question III.11
On raisonne par récurrence sur p, le résultat étant évident si $p = 0$.
Supposons donc qu'on ait numéroté les nœuds à profondeur p avec les entiers de 2^p à $2^{p+1} - 1$.
Notons que cet intervalle d'entiers est exactement égal à $I_p = \{i \in \mathbb{N}^*, p \leqslant \lg i < p + 1\}$.
Quand i décrit I_p, $2^p + i$ décrit $J_p = [2^{p+1}, 2^{p+1} + 2^p - 1]$ et $2^{p+1} + i$ décrit $K_p = [2^{p+1} + 2^p, 2^{p+2} - 1]$. Les intervalles J_p et K_p sont disjoints et $J_p \cup K_p = I_{p+1}$: cela permet d'assurer l'hérédité de la récurrence.

Question III.12
Bien sûr, les intervalles I_p sont eux-mêmes deux à deux disjoints, ce qui garantit l'unicité de la numérotation.

Question III.13
On raisonne là encore par récurrence sur p.
Si $p = 0$, on considère la racine, de numéro $n = 1 = 2^p$ et d'occurrence $\langle\rangle$.
Si $p = 1$, le nœud d'occurrence $\langle 0\rangle$ porte le numéro $n = 2 = 2^1 + 0.2^0$ et le nœud d'occurrence $\langle 1\rangle$ porte le numéro $n = 3 = 2^1 + 1.2^0$.
Supposons que le résultat soit acquis pour tout nœud de profondeur au plus égale à p. Considérons un nœud numéroté n, de profondeur $p+1$, et notons $\langle c_1, c_2, \ldots, c_p\rangle$ l'occurrence de son père et m le numéro de ce père. Alors :

- ▷ si le nœud considéré est fils gauche de son père, $c_{p+1} = 0$; donc

$$n = m + 2^p = 2^p + \sum_{i=0}^{p-1} c_{i+1}2^i + 2^p = 2^{p+1} + \sum_{i=0}^{p-1} c_{i+1}2^i = 2^{p+1} + \sum_{i=0}^{p} c_{i+1}2^i;$$

- ▷ si c'est le fils droit, $c_{p+1} = 1$; donc

$$n = m + 2^{p+1} = 2^p + \sum_{i=0}^{p-1} c_{i+1}2^i + 2^{p+1} = 2^{p+1} + \sum_{i=0}^{p-1} c_{i+1}2^i + 2^p = 2^{p+1} + \sum_{i=0}^{p} c_{i+1}2^i.$$

On a bien réussi à enclencher la récurrence dans les deux cas.

Question III.14
D'après le calcul précédent, c_i n'est autre que le reste dans la division par 2 de $\left\lfloor \frac{n}{2^{i-1}} \right\rfloor$.

Question III.15
On écrit facilement le programme 1.

```
let rec taille = function
  | Vide -> 0
  | Fourche(g,d) -> (taille g) + (taille d)
  | Noeud(g,_,d) -> 1 + (taille g) + (taille d) ;;
```

Programme 1 la fonction `taille`

Question III.16
Il s'agit ici de parcourir le graphe en mettant à jour le couple $mm = (m, M)$ des indices minimum et maximum rencontrés.
On écrit une fonction auxiliaire `minimax` qui travaille sur ces couples, et une fonction récursive `parcours` qui admet 4 arguments : le couple $mm = (m, M)$, l'indice courant n, et $\pi = 2^p$. On obtient le programme 2.

```
let bornes a =
  let minimax (a,b) (c,d) = (min a c, max b d)
  in
  let rec parcours mm n pi = function
    | Vide -> mm
    | Fourche(g,d) -> parcours (parcours mm (n+pi) (2*pi) g) (n+2*pi) (2*pi) d
    | Noeud(g,_,d) -> parcours (parcours (minimax mm (n,n)) (n+pi) (2*pi) g)
                               (n+2*pi) (2*pi) d
  in
    parcours (max_int,0) 1 1 a ;;
```

Programme 2 la fonction `bornes`

Remarques : `max_int` est le plus grand entier représentable en machine, et est donc l'élément neutre de la fonction `min` ; l'appel à `minimax` n'est effectué que dans le cas d'un `Noeud`, évidemment.

Question III.17
Le résultat de III.13 permet d'observer que si i est le numéro d'un nœud de l'arbre a, qui se trouve dans le fils gauche g de la racine, alors i est pair et, de plus, le même nœud, dans la numérotation du sous-arbre g, porte le numéro $i/2 = \lfloor i/2 \rfloor$.
Le résultat est le même dans le cas où i est impair : le nœud figure alors dans le fils droit d de la racine (sauf si $i = 1$, bien sûr, qui est le cas de la racine elle-même), et, de plus, son numéro dans la numérotation du sous-arbre d est égal à $(i-1)/2 = \lfloor i/2 \rfloor$.
On en déduit l'écriture du programme 3.

```
let rec remplacer i e = function
  | Vide -> Vide
  | Fourche(g,d) ->
      if i = 1 then Noeud(g,e,d) (* ou : failwith "pas de valeur à l'indice i" *)
      else if i mod 2 = 0 then Fourche(remplacer (i/2) e g, d)
      else Fourche(g, remplacer (i/2) e d)
  | Noeud(g,v,d) ->
      if i = 1 then Noeud(g,e,d)
      else if i mod 2 = 0 then Noeud(remplacer (i/2) e g, v, d)
      else Noeud(g, v, remplacer (i/2) e d) ;;
```

Programme 3 la fonction `remplacer`

Question III.18
On peut écrire quelque chose comme

$$(\mathcal{V}_{min}(v) = \min\{\iota(x), x \in \mathcal{N}(\mathcal{V}_a(v))\}) \wedge (\mathcal{V}_{max}(v) = \max\{\iota(x), x \in \mathcal{N}(\mathcal{V}_a(v))\})$$
$$\wedge(\forall x \in \mathcal{N}(\mathcal{V}_a(v)), \mathcal{E}(x) \neq \mathcal{V}_d(v))$$
$$\wedge(\mathcal{V}_t(v) = \text{Card}\{\mathcal{E}(x), x \in \mathcal{N}(\mathcal{V}_a(v))\}) \wedge (\forall x \in \mathcal{F}(\mathcal{V}_a(v)), \mathcal{G}(x) \neq \emptyset \vee \mathcal{D}(x) \neq \emptyset),$$

où on a noté $\iota(x)$ l'indice d'un nœud x.

Question III.19
On écrit le programme 4. La fonction `verification` prend en argument l'indice courant n, $\pi = 2^p$ (où p est la profondeur courante) et l'arbre à analyser : elle déclenche l'exception `Incorrect` dans le cas d'une fourche vide, ou d'un nœud dont la valeur est égale à la valeur par défaut ou dont l'indice n'est pas dans l'intervalle prévu. Si tout va bien, elle renvoie le nombre de nœuds rencontrés.

```
exception Incorrect ;;

let valider (imin,imax,t,v,a) =
  let rec verification n pi = function
    | Vide -> 0
    | Fourche(Vide,Vide) -> raise Incorrect
    | Fourche(g,d) -> (verification (n+pi) (2*pi) g)
                    + (verification (n+2*pi) (2*pi) d)
    | Noeud(g,x,d) -> if n < imin || n > imax || x = v then raise Incorrect
                else (verification (n+pi) (2*pi) g)
                     + 1 + (verification (n+2*pi) (2*pi) d)
  in
    try verification 1 1 a = t with Incorrect -> false ;;
```

Programme 4 la fonction `valider`

Question III.20
On s'inspire du programme 3 pour écrire le programme 5 : la fonction récursive `trouver` n'est utilisée que si l'indice proposé sort du domaine $[imin, imax]$.

```
let lire i (imin,imax,t,v,a) =
  let rec trouver i = function
    | Vide -> v
    | Fourche (g,d) ->
       if i = 1 then v
       else trouver (i/2) (if i mod 2 = 0 then g else d)
    | Noeud(g,x,d) ->
       if i = 1 then x
       else trouver (i/2) (if i mod 2 = 0 then g else d)
  in
    if i < imin || i > imax then v
    else trouver i a ;;
```

Programme 5 la fonction `lire`

Question III.21
On propose le programme 6, page 7.
La fonction `modifie` renvoie le sous-arbre courant modifié et `dt` qui vaut 1 si un nouveau nœud a été créé : il s'agit du cas où x remplace la valeur par défaut.

```
let ecrire i x (imin,imax,t,v,a) =
  let rec modifie i = function
    | Vide ->
       if i = 1 then (Noeud(Vide,x,Vide),1)
       else let (a',_) = modifie (i/2) Vide in
         if i mod 2 = 0 then (Fourche(a',Vide),1)
         else (Fourche(Vide,a'),1)
    | Fourche(g,d) ->
       if i = 1 then (Noeud(g,x,d),1)
       else if i mod 2 = 0 then let (g',dt) = modifie (i/2) g in (Fourche(g',d),dt)
       else let (d',dt) = modifie (i/2) d in (Fourche(g,d'),dt)
    | Noeud(g,w,d) ->
       if i = 1 then (Noeud(g,x,d),0)
       else if i mod 2 = 0 then let (g',dt) = modifie (i/2) g in (Noeud(g',w,d),dt)
       else let (d',dt) = modifie (i/2) d in (Noeud(g,w,d'),dt)
  in
  let (a',dt) = modifie i a in
    (min i imin, max i imax, t+dt, v, a') ;;
```

Programme 6 la fonction `ecrire`

Question III.22
Le programme 7 reprend les idées précédentes. On suppose ici, pour simplifier un peu, que la somme de deux éléments des vecteurs arguments ne peut jamais être égale à la valeur par défaut.

```
let somme (im1,iM1,t1,v1,a1) (im2,iM2,t2,v2,a2) =
  let rec ajoute = function
    | (Vide,Vide) -> (Vide,0)
    | (Vide,a) -> (a,taille a)
    | (a,Vide) -> (a,taille a)
    | (Fourche(g1,d1),Fourche(g2,d2)) ->
       let (g,tg) = ajoute (g1,g2) and (d,td) = ajoute (d1,d2) in
         (Fourche(g,d),tg+td)
    | (Fourche(g1,d1),Noeud(g2,x2,d2)) ->
       let (g,tg) = ajoute (g1,g2) and (d,td) = ajoute (d1,d2) in
         (Noeud(g,x2,d),tg+td+1)
    | (Noeud(g1,x1,d1),Fourche(g2,d2)) ->
       let (g,tg) = ajoute (g1,g2) and (d,td) = ajoute (d1,d2) in
         (Noeud(g,x1,d),tg+td+1)
    | (Noeud(g1,x1,d1),Noeud(g2,x2,d2)) ->
       let (g,tg) = ajoute (g1,g2) and (d,td) = ajoute (d1,d2) in
         (Noeud(g,x1 +. x2,d),tg+td+1)
  in
  if v1 <> v2 then failwith "la valeur par défaut doit être commune"
  else
    let (a,t) = ajoute (a1,a2) in
      (min im1 im2, max iM1 iM2, t, v1, a) ;;
```

Programme 7 la fonction `somme`