centrale 2013 : corrigé

I.A -

```
let monAD = [|
  Decision("examen",1,2);
  Feuille true;
  Decision("assiduite",3,4);
  Decision("rattrapage",5,6);
  Feuille false;
  Feuille true;
  Feuille false |];;
```

I.B -

```
let eval_var variable liste =
  let rec aux = function
    | [] -> false
    | h::t -> variable = h || aux t
  in
  aux liste;;
```

I.C -

```
let eval arbre liste =
  let rec aux i = match arbre.(i) with
    | Feuille b -> b
    | Decision(v,g,d) -> aux (if eval_var v liste then g else d)
  in
  aux 0;;
```

II.A -

```
  let redirige diagramme v w =
     let n = vect_length diagramme in
     diagramme.(v) <- Vide;
     for j = 0 to n-1 do
       match diagramme.(j) with
         | Decision (variable,g,d) ->
             diagramme.(j) <- Decision (variable, (if g=v then w else g), (if d=v then w else d))
         | _ -> ()
       done;;
```

II.B -

```
let trouve_elimination diagramme =
  let n = vect_length diagramme in
  let i = ref 0 in
  let result = ref (-1) in
  while !result = -1 && !i < n do
    match diagramme.(!i) with
      | Decision (_,g,d) ->  if g <> d then i := !i+1 else result := !i
      | _ -> i:= !i+1
  done;
  !result;;
```

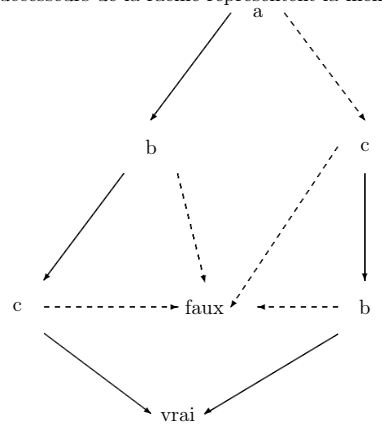

II.C -

```
let trouve_isomorphisme diagramme =
  let n = vect_length diagramme in
  let result = ref (-1,-1) in
  let i = ref 0 in
  while !result = (-1,-1) && !i < n do
    let j = ref (!i + 1) in
    while !result = (-1,-1) && !j < n do
      if
       (match (diagramme.(!i),diagramme.(!j)) with
         | (Feuille a,Feuille b) -> a=b
         | (Decision(v,g,d), Decision(w,l,r)) ->
               v=w && g=l && d=r
         | _ -> false)
      then
        result := (!i,!j)
      else
        j:= !j+1
      done;
    i:= !i+1
    done;
  !result;;
```

II.D -

L'énoncé semble dire que si v est une variable et p une formule, les **formules** p et $(v \wedge p) \vee (\neg v \wedge p)$ sont identiques, alors qu'**elles ne sont qu'équivalentes**.

Or la réduction proposée n'empèchera pas des noeuds distincts de correspondre à la même **fonction** logique, comme le montre l'exemple suivant auquel aucune des deux règles ne peut être appliquée bien que les deux successeurs de la racine représentent la même **fonction** logique $b \wedge c$.



Démontrons néanmoins que des noeuds distincts correspondent à des **formules** distinctes si et seulement si la règle d'isomorphisme ne peut être appliquée (ou, suivant les notations de la partie III, une formule est soit une constante booléenne, soit de la forme $v \to p, q$ où v est une variable et p, q des formules ne faisant pas intervenir v ; cette hypothèse est nécessaire pour éviter des cycles dans les diagrammes)

Si la règle d'isomorphisme peut être appliquée, alors deux noeuds distincts représentent la même formule logique.

Réciproquement, supposons que la règle d'isomorphisme ne s'applique pas et qu'il existe deux noeuds d'indices distincts i, j représentent la même formule logique. Parmi les paires de tels noeuds, choisissons-en une dont la somme des hauteurs est minimale, où nous définissons la hauteur d'un noeud comme la plus grande des longueurs des chemins de ce noeud à une feuille (l'énoncé a négligé de définir ce qu'était un "diagramme de décision" ; il faut les définir comme acycliques pour que cette définition de la hauteur fasse sens). Alors de deux choses l'une :
- ces deux noeuds sont des feuilles, auquel cas on peut leur appliquer la règle d'isomorphisme, ce qui est contradictoire.
- ces deux noeuds sont des noeuds internes. Leurs variables sont les mêmes, $succ_T(i)$ et $succ_T(j)$ représentent la même formule logique et il en va de même pour $succ_F(i)$ et $succ_F(j)$. Comme la règle d'isomorphisme ne s'applique pas, on peut supposer (par exemple) que $succ_T(i)$ et $succ_T(j)$ sont distincts. Comme la somme de leurs hauteurs est strictement inférieur à la somme des hauteurs des noeuds d'indices i et j, ceci est contradictoire.

II.E -

```
let reduit diagramme =
  let n = vect_length diagramme in
  let termine = ref false in
  while not !termine do
    let v = trouve_elimination diagramme in
    if v <> -1 then
      match diagramme.(v) with
       | Decision(_,w,_) -> redirige diagramme v w
       | _ -> failwith "impossible"
    else begin
       let (v,w) = trouve_isomorphisme diagramme in
       if (v,w) <> (-1,-1) then redirige diagramme v w else termine := true
       end
    done
;;
```

III.A - On a les égalités logiques (et non syntaxiques) suivantes :

$$\neg x = x \to 0, 1 \qquad x \vee y = x \to 1, y \qquad x \wedge y = x \to y, 0$$

III.B - On peut au choix dresser une table de vérité (comportant 32 lignes...) ou utiliser les lois de Morgan :

$$\begin{aligned}(a \to b, c) \to d, e &= ((a \wedge b) \vee (\neg a \wedge c)) - > d, e \\ &= (((a \wedge b) \vee (\neg a \wedge c)) \wedge d) \vee ((\neg a \vee \neg b) \wedge (a \vee \neg c) \wedge e) \\ &= (a \wedge b \wedge d) \vee (\neg a \wedge c \wedge d) \vee (\neg a \wedge \neg c \wedge e) \vee (a \wedge \neg b \wedge e) \vee (\neg b \wedge \neg c \wedge e) \\ &= (a \wedge b \wedge d) \vee (\neg a \wedge c \wedge d) \vee (\neg a \wedge \neg c \wedge e) \vee (a \wedge \neg b \wedge e) \vee (a \wedge \neg b \wedge \neg c \wedge e) \vee (\neg a \wedge \neg b \wedge \neg c \wedge e)\end{aligned}$$

Or si la cinquième des conjonctions ci-dessus est vraie, la quatrième l'est aussi. Si la sixième est vraie, la troisième l'est aussi. Donc on peut supprimer les deux dernières conjonctions et il vient :

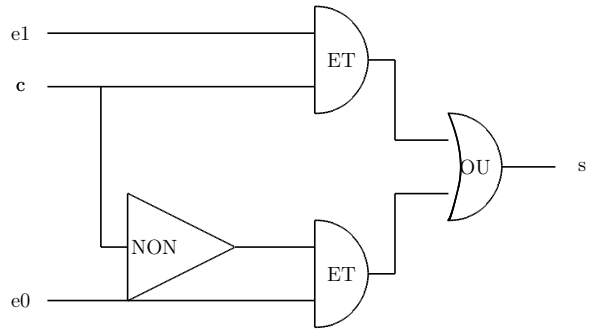
$$\begin{aligned}(a \to b, c) \to d, e &= (a \wedge ((b \wedge d) \vee (\neg b \wedge e))) \vee (\neg a \wedge ((c \wedge d) \vee (\neg c \wedge e))) \\ &= (a \wedge (b \to d, e)) \vee (\neg a \wedge (c \to d, e)) \\ &= a \to (b \to d, e), (c \to d, e)\end{aligned}$$

III.C - Par la question III.A, on peut écrire une formule logique en utilisant uniquement les constantes 0, 1 et l'opérateur $. \to ., .$ et des parenthèses.
Tant qu'il reste à gauche d'une flèche $\to$ une expression non réduite à une variable, on applique la question III.B. Cet algorithme termine car la somme des longueurs des expressions à gauche des flèches diminue strictement à chaque étape. La formule obtenue correspond à un arbre de décision.

III.D - $t \to e, e$ vaut bien sûr e, c'est la règle d'élimination de la partie II.
III.E - $t \to f[t = 1], f[t = 0]$ vaut f.
Remarque : le vocabulaire flotte entre "formule logique" et "expression booléenne"'.
III.F - On construit un diagramme ordonné récursivement : supposant construits des diagrammes ordonnés d_1 et d_0 de $f[x_1 = 1]$ et de $f[x_1 = 0]$ (qu'on peut considérer comme des fonctions de $n - 1$ variables), on forme le diagramme de racine x_1 qui a pour successeurs les racines de d_1 et d_0.
Le cas de base est celui d'une fonction de zéro variables, c'est-à-dire d'une constante booléenne. Dans ce cas le diagramme ordonné est une feuille.
Puis on applique la fonction `reduit` de la question II.E.
III.G - L'existence de u a été établie à la question précédente.
On retrouve a priori la même ambiguité qu'en II.D concernant les diagrammes réduits. Disons que ce sont ceux tels que 2 noeuds distincts correspondent à des formules **syntaxiquement** distinctes et ne comportant pas de noeuds internes dont les successeurs sont identiques. Mais (c'est le sens de la question III.H suivante), des noeuds distincts correspondront **dans le cas des diagrammes réduits ordonnés** à des **fonctions** distinctes, c'est-à-dire à des formules **sémantiquement** distinctes.
Remarquons que parler d'unicité est ici quelque peu abusif : il n'y aura unicité qu'à renumérotation près des noeuds non vides autres que la racine (et le nombre de noeuds égaux à `Vide` est arbitraire) .
On prouve l'unicité de u par récurrence sur le nombre de variables : pour une fonction constante, on ne peut avoir qu'un diagramme réduit à une feuille, donc unique.
Supposant l'unicité établie pour les fonctions d'au plus $n - 1$ variables, tout diagramme réduit ordonné correspondant à une fonction f de n variables non constante doit avoir pour racine le noeud :
-de variable la première (au sens de l'ordre sur les variables) variable x_i vérifiant $f[x_i = 1] \neq f[x_i = 0]$
-de successeurs les racines (renumérotées) des diagrammes réduits ordonnés (qui sont uniques par hypothèse) correspondant aux fonctions $f[x_i = 1]$ et $f[x_i = 0]$.
De plus les "isomorphismes" dans le diagramme ainsi obtenu doivent être simplifiés tant que cela est possible (nous admettrons que le résultat ne dépend pas, à renumérotation près des noeuds non vides, de l'ordre des simplifications). Le diagramme réduit ordonné de f est donc unique (à renumérotation près).
III.H - Il suffit de construire les diagrammes réduits ordonnés et de les comparer **à renumérotation près des noeuds non vides autres que la racine**. On peut aussi former l'arbre de l'équivalence de ces deux fonctions et vérifier que c'est une tautologie, cette vérification est plus facile que la comparaison à renumérotation près.
III.I - Une formule logique est une tautologie si et seulement si son diagramme réduit ordonné est la feuille vraie (c'est-à-dire est décrite par un tableau dont le terme d'indice 0 est la feuille vraie et dont les autres termes valent `Vide`).
IV.A - Encore une question floue. Le nombre de portes est aussi fonction du nombre de *mintermes*. Si on note n le nombre de variables et k le nombre de mintermes, il faudra
$k(n-1)$ portes "et", $k-1$ portes "ou" et on peut majorer le nombre de portes "non" par kn (majoration un peu grossière si k est grand).
Si on cherche une majoration indépendante de k, on majore k par 2^n et le nombre de portes "non" par $\sum_{p=0}^{n} p\binom{n}{p} = n2^{n-1}$. Le pire des cas est celui de la tautologie pour lequel il faut effectivement $n2^{n-1}$ portes "non".
IV.B -

c	e_0	e_1	s
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

IV.C -



IV.D - Etant donné un diagramme de décision, on remplace chaque feuille par une sortie 0 ou 1 et chaque noeud interne par un multiplexeur dont la commande est branchée sur la variable correspondante et dont les entrées sont branchées sur les sorties de leurs successeurs. La sortie de la racine donne la valeur de la fonction booléenne (en fonction des valeurs des variables).

V.A - Le déterminant du système est 1 et les formules de Cramer donnent l'unique solution : (13,30).

Les écritures binaires de 13 et 30 sont 1101 et 11110 donc **les** mots correspondant à la solution sont ceux de la forme :

$$\binom{1}{0}\binom{0}{1}\binom{1}{1}\binom{1}{1}\binom{0}{1}\binom{0}{0}^n \quad (n \in \mathbb{N})$$

V.B - $\vec{v} = \vec{b} + 2\vec{v'}$. Ainsi :

$$(\vec{a}|\vec{v}) = k \Leftrightarrow (\vec{a}|\vec{v'}) = \frac{k - (\vec{a}|\vec{v'})}{2}$$

V.C - L'état initial est indexé par k et l'unique état final par 0.

En effet un mot correspond à une solution si et seulement si lorsqu'on lui suffixe un mot v', on a $(\vec{a}|\vec{v'}) = 0$, et le concaténé du mot vide et d'un mot v' correspond à une solution si et seulement si $(\vec{a}|\vec{v'}) = k$.

V.D - L'ensemble $\{|(\vec{a}|\vec{b})|, b \in A\}$ est fini. Soit M un majorant de cet ensemble.

Dans les notations précédentes, on a $|k_j| \leqslant \frac{|k_i|+M}{2}$.

Ainsi $|k_j| \leqslant max(M, |k_i|)$.

Par induction, on en déduit que tous les indices p d'états accessibles vérifient $|p| \leqslant max(M, |k|)$.

Comme ces indices sont entiers, ils sont en nombre fini.

V.E - On peut utiliser la programmation dynamique :

-On note M la somme des valeurs absolues des coefficients de $\vec{a}$ (sa norme "un") et on pose $K = max(M, |k|)$.

-on crée un tableau de booléens de longueur $2K + 1$ (chaque entier de $[|-K, K|]$ est indexé par un entier de $[|0, 2K|]$) dont toutes les cases valent initialement false.

-on crée une fonction auxiliaire récursive de paramètre un entier p de $[|-K, K|]$; cette fonction calcule chacun des $p - (\vec{a}|\vec{b})$ $(b \in A)$ et lorqu'elle en trouve un qui est pair et dont la moitié p' n'est pas marquée par un "true" dans le tableau, inscrit un "true" dans le tableau et se rappelle elle-même avec le paramètre p'. Dans tous les cas, elle renvoie ().

-on appelle cette fonction avec pour argument k.

-on teste si la case du tableau correspondant à 0 contient "true" ou non.

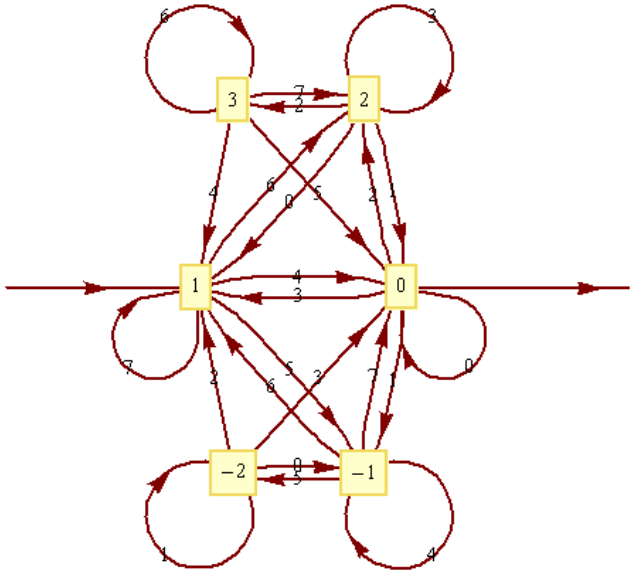
(on peut aussi dans l'écriture de la fonction auxiliaire détecter la valeur 0 du paramètre et lever dans ce cas une exception)

L'algorithme termine car il y a au plus $2K + 1$ appels de la fonction auxiliaire.

V.F -

b	$\binom{0}{0}{0} = 0$	$\binom{0}{0}{1} = 1$	$\binom{0}{1}{0} = 2$	$\binom{0}{1}{1} = 3$	$\binom{1}{0}{0} = 4$	$\binom{1}{0}{1} = 5$	$\binom{1}{1}{0} = 6$	$\binom{1}{1}{1} = 7$
$(\vec{a}\|\vec{b})$	0	2	-4	-2	1	3	-3	-1
k								
1	$\perp$	$\perp$	$\perp$	$\perp$	0	-1	2	1
0	0	-1	2	1	$\perp$	$\perp$	$\perp$	$\perp$
-1	$\perp$	$\perp$	$\perp$	$\perp$	-1	-2	1	0
2	1	0	3	2	$\perp$	$\perp$	$\perp$	$\perp$
-2	-1	0	1	0	$\perp$	$\perp$	$\perp$	$\perp$
3	$\perp$	$\perp$	$\perp$	$\perp$	1	0	3	2

Ci-dessous une représentation de cet automate (merci à la commande GraphPlot de Mathematica) où les lettres de A sont remplacées par des entiers (voir tableau ci-dessus).



V.G - On cherche les solutions s'écrivant avec un mot d'au plus trois lettres de A.
On trouve les solutions suivantes (en identifiant A à $[\![0,7]\!]$) :

mot	4(0)(0)	61(0)	74(0)	57(0)	631	604	625	761	774	757	421	434	417	547	553	564
x_1	1	1	3	3	1	5	5	3	7	7	1	5	5	7	3	7
x_2	0	1	1	2	3	1	3	3	3	5	2	2	4	4	4	2
x_3	0	2	1	3	6	0	4	5	3	7	4	2	6	5	7	1

V.H - Supposons connu un automate $\mathcal{A}$ reconnaissant les mots décrivant les n-uplets vérifiant $f(x_1,\dots,x_n)$, où f est une relation n-aire sur $\mathbb{N}$.
On obtient un automate reconnaissant les mots décrivant les $n-1$-uplets de naturels $(x_1,\dots,x_{i-1},x_{i+1},\dots,x_n)$ vérifiant $\exists x_i \in \mathbb{N}\ f(x_1,\dots,x_n)$ en supprimant la $i^{ème}$ composante des étiquettes des transitions de l'automate précédent.
Pour obtenir un automate reconnaissant les mots décrivant les $n-1$-uplets de naturels $(x_1,\dots,x_{i-1},x_{i+1},\dots,x_n)$ vérifiant $\forall x_i \in \mathbb{N}\ f(x_1,\dots,x_n)$, on commence par former un automate reconnaissant le complémentaire du langage reconnu par $\mathcal{A}$, puis on applique l'opération précédente, et on passe à nouveau au complémentaire.
Remarquons que chaque passage au complémentaire nécessite une déterminisation, l'automate ainsi obtenu a beaucoup plus d'états que l'automate initial...