OPTION INFORMATIQUE
Devoir surveillé 3
Corrigé

1 Cherchez les Clés du Paradis (d'après CCP 2018)

1.1 Préparation

1. $A \vee \bot \equiv A$
   $A \vee \top \equiv \top$
   $A \wedge \bot \equiv \bot$
   $A \wedge \top \equiv A$
   $A \vee \overline{A} \equiv \top$
   $A \wedge \overline{A} \equiv \bot$
   $A \Rightarrow \bot \equiv \overline{A}$
   $A \Rightarrow \top \equiv \top$
   $\bot \Rightarrow A \equiv \top$
   $\top \Rightarrow A \equiv A$

1.2 Première épreuve

1. $B_1 = P_1 \vee P_2$
   $B_2 = \overline{P_1}$
2. On note ϕ l'affirmation de l'animateur.

$$\begin{aligned} \phi &= (B_1 \wedge B_2) \vee (\overline{B_1} \wedge \overline{B_2}) \\ &\equiv ((P_1 \vee P_2) \wedge \overline{P_1}) \vee (\overline{P_1} \wedge \overline{P_2} \wedge P_1) \\ &\equiv (P_1 \wedge \overline{P_1}) \vee (P_2 \wedge \overline{P_1}) \\ &\equiv P_2 \wedge \overline{P_1} \end{aligned}$$

3. D'après le calcul précédent, il faut donc ouvrir la boîte numéro 2.

1.3 Deuxième épreuve

1. $B_1 = \overline{P_1} \oplus P_2$
   $B_2 = P_1$
2. On a $\phi = (B_1 \wedge B_2) \vee (\overline{B_1} \wedge \overline{B_2})$
   On dresse la table de vérité de ϕ :

P_1	P_2	B_1	B_2	ϕ
F	F	V	F	F
F	V	F	F	V
V	F	F	V	F
V	V	V	V	V

   Les valuations satisfaisant ϕ sont donc $\{P_1 \mapsto F, P_2 \mapsto V\}$ et $\{P_1 \mapsto V, P_2 \mapsto V\}$.
   Il faut donc ouvrir la boîte numéro 2.

1.4 Troisième épreuve

1. On remarque qu'on peut obtenir $\bot \equiv A \oplus A$
   On peut donc obtenir la négation : $\overline{A} \equiv A \Rightarrow \bot \equiv A \Rightarrow (A \oplus A)$.
   Enfin, on peut écrire une disjonction comme une implication :
   $A \vee B \equiv \overline{A} \Rightarrow B$
   On sait que $\{\vee, \neg\}$ est un système complet de connecteurs, il ne reste plus qu'à démontrer par induction structurelle que toute formule ϕ n'utilisant que les connecteurs $\vee$ et $\neg$ est équivalente à une formule n'utilisant que $\oplus$ et $\Rightarrow$:
   - Si ϕ est une variable, c'est immédiat ;
   - Si $\phi = \neg\phi'$.
     Supposons que ϕ' soit équivalente à ϕ'' composée de $\oplus$ et $\Rightarrow$.
     On a alors $\phi \equiv \phi'' \Rightarrow (\phi'' \oplus \phi'')$ composée de $\oplus$ et $\Rightarrow$.
   - Si $\phi = \phi_1 \vee \phi_2$.
     Supposons que ϕ_1, ϕ_2 sont équivalentes à ϕ_1', ϕ_2' composées de $\oplus$ et $\Rightarrow$.
     On a $\phi \equiv \overline{\phi_1'} \Rightarrow \phi_2' \equiv (\phi_1' \Rightarrow (\phi_1' \oplus \phi_1')) \Rightarrow \phi_2'$.composée de $\oplus$ et $\Rightarrow$.

   En conclusion, toute formule composée de $\vee$ et $\neg$ est équivalente à une formule composée de $\oplus$ et $\Rightarrow$, donc $\{\oplus, \Rightarrow\}$ est un système complet de connecteurs.

2 Pile avec oubli

On représente une pile de capacité c par un tableau de taille $c+2$. Les c premières cases contiennent les éléments de la pile, dans l'ordre mais avec possiblement de la circularité. La case d'indice c indique la taille de la pile, et la case d'indice $c+1$ indique la position du sommet.
Par exemple, la pile $(5, 2, 1, 3)$ (où 5 est le sommet) de capacité 10 peut être représentée par le tableau `[|3;1;2;5;0;0;0;0;0;0;4;3|]`, ou également par le tableau `[|2;5;8;2;0;6;4;6;3;1;4;1|]`

```
let c = 30;;

let creer_pile () = Array.make (c+2) 0;;

let est_vide p =
    let c = Array.length p - 2 in p.(c) = 0;;

let empiler p v =
    let c = Array.length p - 2 in
    let i = p.(c+1) in
    let i' = (i + 1) mod c in
    p.(i') <- v;
    p.(c+1) <- i';
    if p.(c) < c then p.(c) <- p.(c) + 1;;

let depiler p =
    if est_vide p then failwith "depilement de pile vide" else
    let c = Array.length p - 2 in
    let i =p.(c+1) in
    let v = p.(i) in
    let i' = (i - 1) mod c in
    p.(c+1) <- i';
    p.(c) <- p.(c) - 1;
    v;;
```

3 Arbres binaires de recherche

1. V est un arbre binaire de recherche.

$N(g,c,x,d)$ est un arbre binaire de recherche si et seulement si g est un arbre binaire de recherche, d est un arbre binaire de recherche, et $g < c < d$.

2. (a)
```
let rec valeur_associee a c =
    match a with
        V -> failwith "clé non présente"
        |N(g,c',v,d) -> if c = c' then v
                        else if c < c' then valeur_associee g c
                        else valeur associee d c;;
```

   (b) Pour chaque noeud, il y a un appel récursif sur un seul fils. La complexité admet donc en fonction de la hauteur la relation de récurrence $C(h) = C(h-1) + O(1)$, d'où $C(h) = O(h)$.

3. (a)
```
let rec maptree f a =
    match a with
        V -> V
        |N(g,c,x,d) -> N(maptree f g, f c ,x , maptree f d);;
```

   (b) Il faut et il suffit que f conserve l'ordre strict, autrement dit soit strictement croissante.

   (c)
   - Supposons f strictement croissante. Montrons par induction structurelle que pour tout arbre binaire a, si a est un arbre binaire de recherche, alors `maptree f a` est un arbre binaire de recherche :
     — Si $a = V$, alors `maptree f a` renvoie V, qui est un arbre binaire de recherche.
     — Si $a = N(g,c,x,d)$ avec g et d vérifiant la propriété, alors g' et d' images de g et d par `maptree f` sont des arbres binaires de recherche. Pour toute clé n de g, on a $n < c$,donc la clé correspondante $f(n)$ de g' vérifie $f(n) < f(c)$ par stricte croissante de f, ie $g' < f(c)$. Similairement, $f(c) < d'$. Le résultat de `maptree f a`, qui est $N(g', f(c), x, d')$, est donc bien un arbre binaire de recherche.
   - Supposons que `maptree f` transforme tout arbre binaire de recherche en arbre binaire de recherche. Soient $n, m \in \mathbb{Z}$ vérifiant $n < m$. L'arbre binaire de recherche $N(V, n, 0, N(V, m, 0, V))$ a pour image par `maptree f` $N(V, f(n), 0, N(V, f(m), 0, V))$, qui doit être un arbre binaire de recherche. On en déduit donc $f(n) < f(m)$. En conclusion, f est donc bien strictement croissante

4. (a)
```
let rec scission a n =
    match a with
        V -> V,V
        |N(g,c,x,d) ->
            if c<=n then
                let d1,d2 = scission d n in
                N(g,c,x,d1),d2
            else
                let g1,g2 = scission g n in
                g1,N(g2,c,x,d)
;;
```

   (b) Pour chaque noeud, il y a un appel récursif sur un seul fils. La complexité admet donc en fonction de la hauteur la relation de récurrence $C(h) = C(h-1) + O(1)$, d'où $C(h) = O(h)$.

5.
```
let rec fusion a b =
    match a with
        V -> b
        |N(g,c,x,d) ->
            let bg,bd = scission b c in
            N(fusion g bg, c,x,fusion d bd);;
```