Option info - Centrale 2000 **Corrigé**

1 Arbres bicolores

Partie I - B

1. (a) La rotation gauche est appliquée si et seulement si l'arbre est non vide et son fils droit non plus :

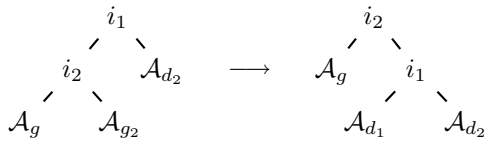
```
let rotation_gauche=fun
  if est_vide a then a
  else let fd=(noeud a).droit in
       if est_vide fd then a
       else begin
              (noeud a).droit<-(noeud fd).gauche;
              (noeud fd).gauche<-a;
              fd
            end;;
```

   (b) Supposons que $\mathcal{A}$ est un arbre binaire de recherche de la forme $(i_1, \mathcal{A}_{g_1}, (i_2, \mathcal{A}_{g_2}, \mathcal{A}_{d_2}))$; notons $\mathcal{A}'_g = (i_1, \mathcal{A}_{g_1}, \mathcal{A}_{g_2})$ et $\mathcal{A}' = (i_2, \mathcal{A}'_g, \mathcal{A}_{d_2}) = rotation-gauche(\mathcal{A})$. Soient n un nœud de $\mathcal{A}'$, de fils droits $\mathcal{A}_g$ et $\mathcal{A}_d$; soient également $n_g \in \mathcal{A}_g$ et $n_d \in \mathcal{A}_d$.

   - Si $n \in \mathcal{A}_{g_1}$ ou $n \in \mathcal{A}_{g_2}$ ou $n \in \mathcal{A}_{d_2}$, ses sous-arbres n'ont pas été modifiés dans la rotation, et la condition $e(n_g) \leq e(n) \leq e(n_d)$ est vérifiée car $\mathcal{A}$ est un ABR.
   - Si $n = i_1 : n_g \in \mathcal{A}_{g_1}$ et $n_d \in \mathcal{A}_{g_2}$, donc $e(n_g) \leq e(i_1) \leq e(n_d)$ vient du fait que n_g et n_d sont dans les sous-arbres gauches et droits de l'ABR $\mathcal{A}$ de racine i_1.
   - Si $n = i_2 : n_d \in \mathcal{A}_{d_2}$ donc $e(n_d) \geq e(i_2)$. n_g est soit égal à i_1, et alors $e(i_1) \leq e(i_2)$; soit dans $\mathcal{A}_{g_1}$, et alors $e(n_g) \leq e(i_1) \leq e(i_2)$; soit enfin il est dans $\mathcal{A}_{g_2}$, et alors $e(n_g) \leq e(i_2)$.

   Une solution plus efficace consiste à utiliser la CARACTERISATION suivante des ABR : "un arbre binaire est de recherche si et seulement si le parcours infixe de ses étiquettes est croissant." Ici, on en déduit imédiatement le résultat puisque le parcours infixe est inchangé après rotation.

2. D'après la définition :

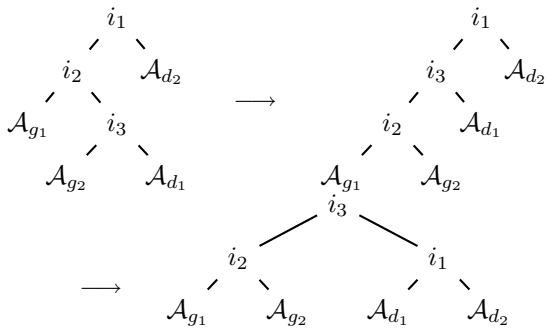


   D'où la fonction :

```
let rotation_droite a=
  if est_vide a then a
```

```
else let fg=(noeud a).gauche in
     if est_vide fg then a
     else begin
            (noeud a).gauche<-(noeud fg).droit;
            (noeud fg).droit<-a;
            fg
          end;;
```

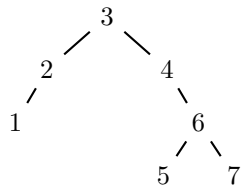
3. (a) La transformation demandée est la suivante :



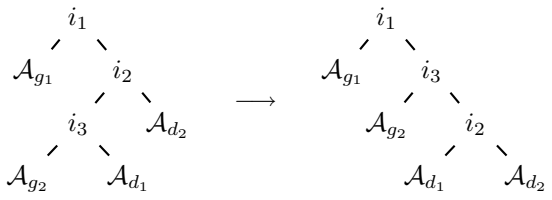
Il faut vérifier AVANT de faire la rotation gauche que la rotation droite est possible (il se pourrait que la rotation droite soit impossible bien que la rotation gauche du fils gauche le soit).

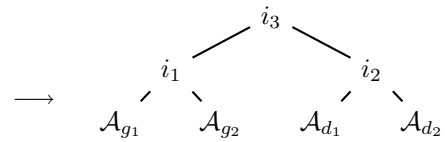
```
let rotation_gauche_droite a=
  if (est_vide a)||(est_vide (noeud a).gauche)
    ||(est_vide (noeud (noeud a).gauche).droit)
    then a else
      begin
        (noeud a).gauche<-rotation_gauche (noeud a).gauche;
        rotation_droite a
      end;;
```

(b) Sans problème :



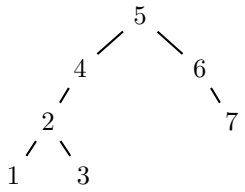
(c) La transformation demandée est la suivante :






```
let rotation_droite_gauche a=
  if (est_vide a)||(est_vide (noeud a).droit)
    ||(est_vide (noeud (noeud a).droit).gauche)
    then a else
      begin
        (noeud a).droit<-rotation_droite (noeud a).droit;
        rotation_gauche a
      end;;
```

(d) En suivant à nouveau la définition :



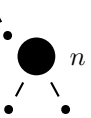
Partie I - C

1. (a) Pour $n \in \mathcal{N}$, notons $\mathcal{A}_n$ le sous-arbre de $\mathcal{A}$ de racine n. On va montrer le résultat par récurrence sur $h(\mathcal{A}_n)$ (et non sur $hn(n)$; danger ...).
   - Si $h(\mathcal{A}_n) = 0$: n est une feuille, donc est de hauteur noire nulle, et $\mathcal{A}_n$ ne possède pas de nœud interne : $0 \leq 2^0 - 1$.
   - Supposons le résultat acquis lorsque $h(\mathcal{A}_n) \leq H$ avec $H \in \mathbb{N}$ donné, et soit n un nœud tel que $h(\mathcal{A}_n) = H + 1$: n est donc un nœud interne, de la forme $\mathcal{A}_n = (n, \mathcal{A}_g, \mathcal{A}_d)$. Si f est l'un des fils de n, on a $hn(n) = hn(f)$ si n est noir et f est blanc, et $hn(n) = hn(f) + 1$ sinon, et donc : $hn(f) \geq hn(n) - 1$. Par ailleurs, les nœuds internes de $\mathcal{A}_n$ sont n, ainsi que ceux de $\mathcal{A}_g$ et $\mathcal{A}_d$. Par "hypothèse de récurrence", ceux-ci sont donc au moins $1 + 2(2^{hn(n)-1} - 1) = 2^{hn(n)} - 1$. ■

   (b) Dans un chemin de la racine à une feuille (en excluant la racine), on trouve $hn(\mathcal{A})$ nœuds noirs et $b \leq hn(\mathcal{A})$ nœuds blancs (car tout nœud blanc est SUIVI d'un nœud noir, d'après (A-2) et (A-4)), donc la profondeur des feuilles est $\leq 2hn(n)$, donc $h(\mathcal{A}) \leq 2hn(\mathcal{A})$. Par ailleurs, grâce à la question précédente : $l \geq 2^{hn(n)} - 1$, donc $hn(n) \leq \log_2(l+1)$: gagné.

   (c) Le recherche dans un ABR s'effectue "en descente" dans l'arbre; sa complexité en temps dans le pire des cas est donc de l'ordre de la hauteur de l'arbre, donc est un $O(\ln l)$, alors que pour un ABR dégénéré, cette hauteur peut-être de l'ordre de l.

2. (a) Si l'arbre initial est vide, on obtient après insertion :

. Il suffit de colorier la racine en noir pour retrouver l'arbre bicolore 

   (b) Si le père de n avant l'insertion initiale est noir, le remplacement de la feuille noire f par l'arbre

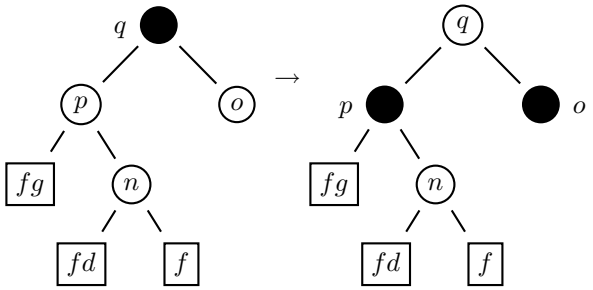
maintient la structure d'arbre bicolore : les contraintes (A-i) sont clairement vérifiées pour $1 \leq i \leq 4$. Pour la cinquième condition, on considère un chemin aboutissant à une feuille : s'il s'agit d'une ancienne feuille, il correspond à un chemin dans l'arbre initial; s'il aboutit à une des nouvelles feuilles, il a rencontré exactement autant de nœuds noirs que dans le chemin dans l'arbre initial aboutissant à l'ancienne feuille f. Les différents chemins issus d'un nœud du nouvel arbre aboutissant à une feuille contiennent donc toujours le même nombre de nœuds noirs : (A-5) reste vérifiée.

   (c) Le couple (f, p) ne viole pas la condition (A-4) : f est donc noir. Il en va d'ailleurs de même pour fg et fd.

   (d) Pour vérifier les conditions (A-3) et (A-4), il est déjà nécessaire que q ait un père noir. Réciproquement, si q a effectivement un père noir : le nouvel arbre vérifie les conditions (A-1), (A-2), (A-3) et (A-4). Mais les chemins aboutissant aux feuilles rencontrent toujours le même nombre de nœuds noirs qu'avant transformation (que les feuilles soient dans fg, fd, f ou o), sauf pour les chemins issus de q, qui rencontrent TOUS un nœud noir de plus. La condition (A-5) reste donc vérifiée, et l'arbre obtenu est bicolore.

   (e) Si q est la racine, il suffit de la colorier en noir. Sinon, q a un père blanc : il devient le nœud courant, et on retourne en c. La condition (A-5) est vérifiée par tous les nœuds de l'arbre (l'explication précédente reste valable), seuls q et r violent la contrainte (A-4), et la profondeur du nœud courant a diminué strictement de 2, ce qui assurera la terminaison.

   (f) Le cas 3 (resp. 4) étant symétrique du cas 2 (resp. 1), il suffit de traiter le cas 2. On garde la même structure, en changeant la couleur de p, q, et o :

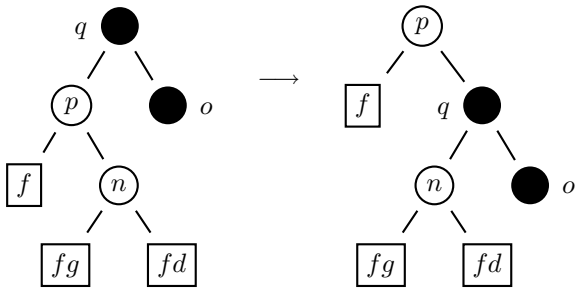


   Le même raisonnement qu'en (d) assure que (A-5) est vérifiée, et (A-4) ne peut être violée que par q et son père : on va donc prendre q comme nœud courant, et retourner en c ...

(g) Structurellement, il s'agit d'une rotation droite. On a par ailleurs changé les couleurs de p et q.

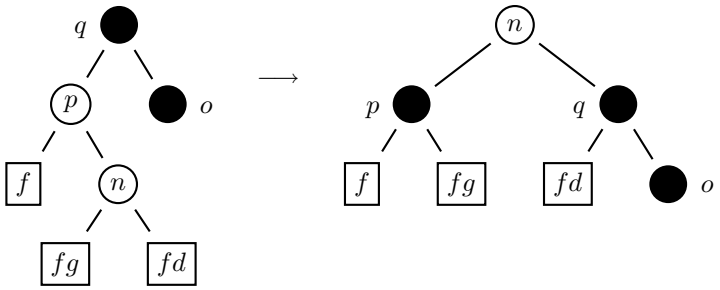
(h) La condition (A-4) est cette fois vérifiée globalement. Par ailleurs, f, fg et fd ont leur racine noire et la même hauteur noire (regarder les chemins de p aux feuilles avant transformation); on en déduit que les chemins issus d'un nœud donné et atteignant une feuille visitent tous le même nombre de nœuds noirs (que l'on parte de n, de p, de q ou d'un ascendant strict de p). La condition (A-5) est ainsi vérifiée, et l'arbre obtenu est bicolore.

(i) Après rotation droite, on obtient :



o et les racines de f, fg et fd ont même hauteur noire avant transformation. Après, les chemins de p aux feuilles de fg rencontreront les nœuds noirs de fg (autant que de nœuds noirs rencontrés dans les chemins vers les feuilles de f), plus l'un des nœuds q ou n, puisqu'ils ne peuvent pas être blancs tous les deux d'après la condition (A-4) : la condition (A-5) ne pourra donc pas être respectée en même temps que la condition (A-4).

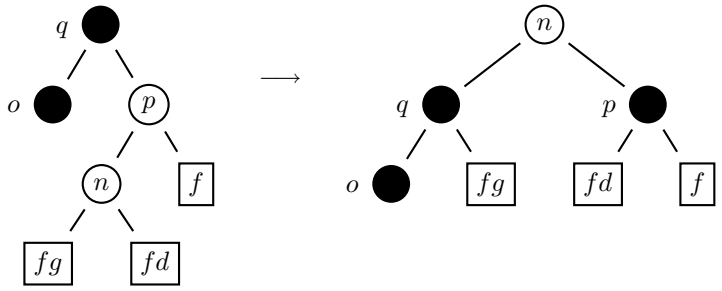
(j) Il suffit de faire une rotation gauche-droite, puis de colorier p en noir.



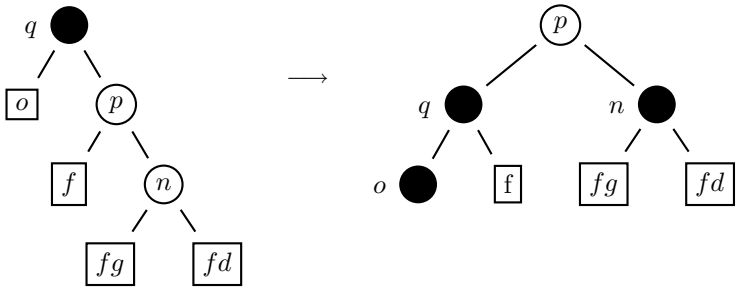
Il est aisé de voir que la condition (A-5) est vérifiée pour les nœuds de ce sous-arbre, et que de plus le nombre de nœuds rencontrés depuis la nouvelle racine (incluse) jusqu'aux feuilles est égal à $2 + nh(f) = 2+nh(fg) = 2+nh(fd)$, ce qui était déjà le cas avant transformation : la contrainte (A-5) restera donc vérifiée par les nœuds extérieurs à $\mathcal{A}_n$.

(k) Le cas 3 est parfaitement symétrique : il s'agit d'une rotation droite-

gauche suivie d'une coloration de p en noir.



(l) La situation est parfaitement symétrique au cas 1 : on fait une rotation gauche, et on change la couleur de p et q.



(m) Le principe du programme est le suivant : on commence par insérer le nouvel élément aux feuilles grâce à la fonction `inserer_arbre_binaire`. Le nœud retourné devient le nœud courant. Tant que celui-ci est blanc et qu'il n'est pas la racine : on observe son père. Si le père est noir, c'est fini. S'il est blanc, on est dans l'un des quatre cas de l'énoncé. Dans chacun de ces cas, on fait une transformation qui remonte le nœud courant de 2 niveaux, ce qui assure la terminaison de l'algorithme (à la fin, il faut d'ailleurs retourner non pas le nœud courant, mais son aïeul).

Le validité vient du fait qu'à chaque étape, la condition (A-5) est vérifiée, et que la condition (A-4) n'est violée que par le nœud courant et son père. Comme à la sortie de la boucle, le nœud courant est noir ou bien a un père noir, on retrouve bien un arbre bicolore.

La complexité en terme d'opérations élémentaires est dans le pire des cas de l'ordre de la profondeur du nœud initial, donc est un $O(\ln l)$, où l est le nombre de nœuds de l'arbre.

On commence par une procédure donnant l'aïeul d'un nœud (la racine de l'arbre dont il fait partie), et par une fonction `coul` donnant la couleur d'un arbre (en traitant le cas de l'arbre vide conformément aux conventions) :

```
let rec adam a=if (noeud a).pere=ArbreVide
               then a
               else adam (noeud a).pere;;

let coul=function
    | ArbreVide  -> Noir
```

```
    | Pointeur a -> a.couleur;;

let inserer_arbre_bicolore a e=
  let n=ref (inserer_arbre_binaire a e) in
  if est_vide a then (noeud !n).couleur<-Noir (* b *)
  else while (noeud !n).couleur=Blanc
  do let p=(noeud !n).pere in
    if p=ArbreVide then (noeud !n).couleur<-Noir
    else if (noeud p).couleur=Noir then n:=p (* pour sortir *)
    else let q=(noeud p).pere in
    match ((noeud q).gauche=p),((noeud p).gauche= !n) with
    | true,true ->    (* Cas 1 *)
       if coul (noeud q).droit=Blanc
         then begin
                (noeud q).couleur<-Blanc;
                (noeud p).couleur<-Noir;
                (noeud (noeud q).droit).couleur<-Noir;
                n:=q
              end
         else begin
                (noeud q).couleur<-Blanc;
                n:=rotation_droite q;
                (noeud !n).couleur<-Noir
              end
    | false,false ->   (* Cas 4, symétrique du cas 1 *)
        ...
| true,false ->    (* Cas 2 *)
       if coul (noeud q).droit=Blanc
         then begin
                (noeud q).couleur<-Blanc;
                (noeud p).couleur<-Noir;
                (noeud (noeud q).droit).couleur<-Noir;
                n:=q
              end
         else begin
                (noeud p).couleur<-Noir;
                n:=rotation_gauche_droite q
              end
    | false,true ->    (* Cas 3, symétrique du cas 2 *)
       ...
  done;
  adam !n;;
```

2 Logique et automates

Partie II - B

1. • $\sigma_i \in p$ pour tout $i \in [\![4, 7]\!]$, donc $(\sigma, 4) \models \Box p$.

- $(\sigma, 4) \not\models q$, donc $(\sigma, 3) \not\models \Box q$, puis : $(\sigma, 2) \not\models \bigcirc(\Box q)$.
- $(\sigma, 4) \models \Box p$, donc $(\sigma, 1) \models \Diamond(\Box p)$.
- $(\sigma, 1) \models p$, $(\sigma, 2) \models p$ et $(\sigma, 3) \models q$, donc $(\sigma, 1) \models p \mathcal{J} q$.

2. Si $(\sigma, 1) \models \Box(\bigcirc p)$, alors $|\sigma| > 1$ et pour tout $j \in [\![1, |\sigma| - 1]\!]$, $(\sigma, j) \models \bigcirc p$. En particulier pour $j = |\sigma| - 1 \geq 1$: $(\sigma, |\sigma|) \models p$, ce qui est exclu car $|\sigma| \geq |\sigma| \ldots$

   Finalement aucune suite σ ne vérifie $(\sigma, 1) \models \Box(\bigcirc p)$.

3. $(\sigma, i) \models \bigcirc V$ si et seulement si $(\sigma, i+1) \models V$, c'est-à-dire $i + 1 < |\sigma|$, ce qui revient à dire que σ_i n'est pas le dernier élément de σ. On peut donc prendre : $Dern = \neg(\bigcirc V)$.

4. La formule recherchée est telle que $\sigma \models Pair$ si et seulement si :

   - $(\sigma, 0) \models p$;
   - pour tout $i \in [\![0, |\sigma| - 2]\!]$, on a ($(\sigma, i) \models p$) et $(\sigma, i+1) \not\models p$) ou bien ($(\sigma, i) \not\models p$) et $(\sigma, i+1) \models p$).

   Dans le sens direct, c'est trivial, et pour la réciproque, c'est une récurrence pas bien méchante (elle l'est plus à rédiger qu'à comprendre !)

   La condition $(\forall i \in [\![0, |\sigma| - 2]\!],\ \varphi)$ peut se traduire par : $\Box(Dern \vee \varphi)$, et ainsi, on peut prendre :

$$Pair = p \bigwedge \Box\Big(Dern \bigvee \big(p \wedge \bigcirc(\neg p)\big) \bigvee \big((\neg p) \wedge \bigcirc p\big)\Big).$$

5. - $\sigma \models \Box\Diamond\varphi$ si et seulement si $\forall j \in [\![0, |\sigma| - 1]\!]$, $(\sigma, j) \models \Diamond\varphi$, ce qui revient à dire que pour tout $j \in [\![0, |\sigma| - 1]\!]$, il existe $k \in [\![j, |\sigma| - 1]\!]$ tel que $(\sigma, k) \models \varphi$. Une condition nécessaire ($j = |\sigma| - 1$) mais aussi suffisante ($k = |\sigma| - 1$) est : $(\sigma, |\sigma| - 1) \models \varphi$.

     Il est facile de voir que cette condition est également équivalente à : $\sigma \models \Diamond\Box\varphi$. Ainsi, on a bien :

$$\Box\Diamond\varphi \iff \Diamond\Box\varphi.$$

   - L'équivalence est vérifiée (simple jeu d'écriture ...). Si $\sigma \models \Diamond\varphi$, alors $(\sigma, 0) \models \Diamond\varphi$ (définition de $\sigma \models$), donc il existe $i \in [\![0, |\sigma| - 1]\!]$ tel que $(\sigma, i) \models \Diamond\varphi$, donc $\sigma \models \Diamond\Diamond\varphi$.

     Réciproquement, si $\sigma \models \Diamond\Diamond\varphi$, il existe $i \in [\![0, |\sigma| - 1]\!]$ tel que $(\sigma, i) \models \Diamond\varphi$. Il existe alors $j \in [\![i, |\sigma| - 1]\!]$ tel que $(\sigma, j) \models \varphi$, et donc $\sigma \models \Diamond\varphi$. Ouf ...

$$\Diamond\varphi \iff \Diamond\Diamond\varphi$$

   - On prend $\varphi = \{e_1\}$, $\psi = \{e_2\}$ (simples propriétés) avec $e_1 \neq e_2$, et $\sigma = (e_1, e_2)$: $\sigma \models \Box(\varphi \vee \psi)$ mais $\sigma \not\models (\Box\varphi) \vee (\Box\psi)$.

     Les deux propositions ne sont donc pas équivalentes.

   - On prend $\varphi = \{e_1, e_2\}$, $\psi = \{e_2\}$, et $\sigma = (e_1, e_2)$: $(\sigma, 1) \models \varphi \wedge \Box\psi$, donc $\sigma \models \Diamond(\varphi \wedge \Box\psi)$, mais $(\sigma, 0) \not\models \psi$, donc $\sigma \not\models \Box\psi$, et a fortiori $\sigma \not\models (\Diamond\varphi) \wedge (\Box\psi)$.

     Les deux propositions ne sont donc pas équivalentes.

- On prend $\varphi = \{e_1\}$, $\psi = \{e_1, e_2\}$, et $\sigma = (e_1, e_2)$: $\sigma \models (\Diamond\varphi) \wedge (\Box\psi)$, mais $(\sigma, 1) \not\models \varphi$, donc $(\sigma, 1) \not\models \Diamond\varphi$, puis $(\sigma, 1) \not\models (\Diamond\varphi \wedge \psi)$, et en conséquence : $\sigma \not\models \Box(\Diamond\varphi \wedge \psi)$.

  Les deux propositions ne sont donc pas équivalentes.

- Si $\sigma \models (\varphi \mathcal{J} \psi)\mathcal{J}\psi$: il existe $j \in [\![0, |\sigma| - 1]\!]$ tel que $(\sigma, j) \models \psi$ et $(\sigma, k) \models \varphi\mathcal{J}\psi$ pour tout $k \in [\![0, j-1]\!]$.

  Si $j = 0$, on a $(\sigma, 0) \models \psi$ donc $\sigma \models \varphi\mathcal{J}\psi$. Sinon, pour $k = 0$, on obtient : $\sigma \models \varphi\mathcal{J}\psi$.

  Réciproquement : Supposons : $\sigma \models \varphi\mathcal{J}\psi$: il existe $j \in [\![0, |\sigma| - 1]\!]$ tel que $(\sigma, j) \models \psi$ et $(\sigma, k) \models \varphi$ pour tout $k \in [\![0, j-1]\!]$. Si $k \in [\![0, j-1]\!]$ et $l \in [\![k, j-1]\!]$, on a a fortiori $l \in [\![0, j-1]\!]$, donc $(\sigma, l) \models \varphi$; mais $(\sigma, j) \models \psi$, donc $(\sigma, k) \models \varphi\mathcal{J}\psi$. Ceci est vrai pour tout $k \in [\![0, j-1]\!]$, et de plus $(\sigma, j) \models \psi$, donc $\sigma \models (\varphi\mathcal{J}\psi)\mathcal{J}\psi$.

$$(\varphi\mathcal{J}\psi)\mathcal{J}\psi \quad \Longleftrightarrow \quad \varphi\mathcal{J}\psi.$$

6. La formule $\Diamond\varphi$ est équivalente à $V\mathcal{J}\varphi$ (cf définitions !). On va donc prouver le résultat demandé par induction structurelle sur les formules :

   - Le résultat est vrai pour les propriétés p.
   - S'il est vérifié pour φ et ψ : on a $\varphi \Leftrightarrow \widetilde{\varphi}$ et $\psi \Leftrightarrow \widetilde{\psi}$ où $\Diamond$ n'apparaît ni dans $\widetilde{\varphi}$ ni dans $\widetilde{\psi}$. On a alors les relations suivantes, qui fournissent des formules équivalentes à celles construites à partir de φ et ψ, mais où n'apparaît pas $\Diamond$:
     - $(\varphi \wedge \psi) \Leftrightarrow (\widetilde{\varphi} \wedge \widetilde{\psi})$;
     - $(\varphi \vee \psi) \Leftrightarrow (\widetilde{\varphi} \vee \widetilde{\psi})$;
     - $(\neg\varphi) \Leftrightarrow (\neg\widetilde{\varphi})$;
     - $(\bigcirc\varphi) \Leftrightarrow (\bigcirc\widetilde{\varphi})$;
     - $(\varphi\mathcal{J}\psi) \Leftrightarrow (\widetilde{\varphi}\mathcal{J}\widetilde{\psi}$;
     - $(\Diamond\varphi) \Leftrightarrow (V\mathcal{J}\varphi) \Leftrightarrow (V\mathcal{J}\widetilde{\varphi})$;
     - $(\varphi\mathcal{J}\psi) \Leftrightarrow (\widetilde{\varphi}\mathcal{J}\widetilde{\psi})$.

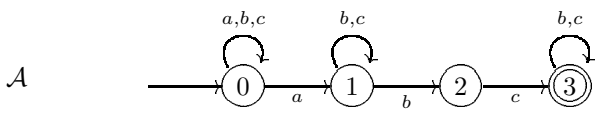
7. Même principe que pour la question précédente, en notant qu'on a l'équivalence :

$$\Box\varphi \quad \Longleftrightarrow \quad \varphi\mathcal{J}(Dern \wedge \varphi),$$

   puis en raisonnant par induction structurelle, ce qui permet d'éliminer les opérateurs $\Box$ sans faire apparaître $\Diamond$ (bien entendu, on a noté que $Dern$ n'utilisait que l'opérateur $\bigcirc$ ...).

Partie II - C

1. La construction d'un automate $\mathcal{A}$ reconnaissant L ne pose guère de problème :



Un mot sera dans L si et seulement si au dela de la dernière occurence de a, il existe un facteur bc. Notons $p = p_a \vee p_b \vee p_c$. La formule $p\mathcal{J}(p_a \wedge O\psi)$ sera vérifiée par les mots de la forme $w_1 a w_2$, avec $w_2 \models \psi$ (et seulement par eux).

ψ doit être une formule satisfaite exactement par les mots sur $\{b, c\}$ contenant au moins un b suivi d'un c. On pourra prendre par exemple :

$$\psi = (\Box\neg p_a) \wedge \Diamond(p_b \wedge Op_c)$$

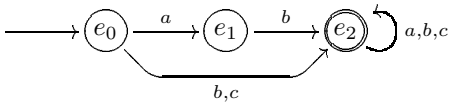
et enfin : $\varphi = p\mathcal{J}(p_a \wedge O\psi)$.

2.
- L_1 est constitué des mots (non vides ...) tels que la première lettre n'est pas un a, ou alors tel qu'il existe une seconde lettre, qui doit être un b. Ce langage est donc décrit par l'expression rationnelle

$$e_1 = (ab + b + c).(a + b + c)^*,$$

et reconnu par l'automate :

$\mathcal{A}_1$

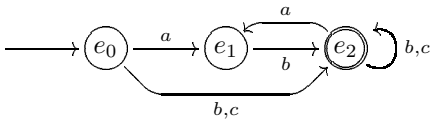


- L_2 est constitué des mots (non vides) tels que **chaque** occurence de a est suivie d'un b. Une expression rationnelle décrivant L_3 est donc

$$e_2 = (ab + b + c).\big(ab + b + c\big)^*,$$

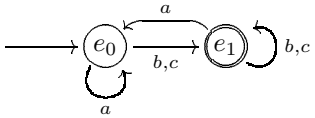
et un automate reconnaissant L_2 est :

$\mathcal{A}_2$



- Un mot non vide σ est dans L_3 si et seulement si il existe un rang au delà duquel toute lettre a est suivie d'un b : tous les mots vérifient cette condition sauf ceux qui se terminent par un a (les autres vérifient la condition ... à partir de la dernière lettre !). L_3 est donc décrit par

$$e_3 = (a + b + c)^*.(b + c)$$

et reconnu par:

$\mathcal{A}_3$