D.S. 1+

Arborescences

1 Première partie

Question 1
On compte $N_t(1,1) = 3$, $N_t(1,2) = 4$, $N_t(1,3) = 0$, $N_t(2,2) = 1$, $N_t(2,3) = 1$, $N_t(1) = 10$, $N_t(2) = 7$ et $N_t(3) = 2$. La matrice de ramification est donc $\begin{pmatrix} 1 & 0 & 0 \\ \frac{4}{7} & \frac{3}{7} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}$.

Question 2
Les arêtes d'épaisseur k parviennent à un nœud qui peut être de bi-épaisseur $(k-1, k-1)$ ou (i, k) avec $i < k$ pour $k \geqslant 1$. On a donc $N_t(k) = \sum_{i=1}^{k-1} N_t(i,k) + N_t(k,k)$ ce qui implique que $\sum_{i=0}^{ep(t)} R_{k,i} = 1$ pour $k > 1$; l'égalité pour $k = 1$ est évidente.

Question 3
La question est ambiguë : que signifie "remplis" ?
Si on interprète "le niveau de profondeur h est rempli" par "tous les nœuds de niveau $h-1$ ont deux fils c'est-à-dire sont de degré 2 alors cela est équivalent au fait qu'il n'y a pas de feuille au niveau $h-1$ (car les nœuds ont 0 ou 2 pour degré) donc que toutes les feuilles sont au niveau maximal.
Dans ce cas on montre par récurrence que tous les nœuds de profondeur k on une bi-épaisseur $(h-k, h-k)$ car alors les arêtes entre les nœuds de profondeur k et ceux de profondeur $k-1$ ont pour épaisseur $h-k+1$. On a ainsi $N_t(i,j) = 0$ pour $i \neq j$.
On en déduit que $ep(t) = h+1$ car l'épaisseur de la racine est (h,h) :
la matrice de bifurcation est I_{h+1}.

Question 4
Un arbre-peigne de hauteur h a h nœuds internes, chacun ayant un fils feuille (2 pour le dernier). La bi-épaisseur du dernier nœud et $(1,1)$ tandis que les autres ont la bi-épaisseur $(2,1)$. On a donc $N_2 = h$, $N_{2,1} = h-1$ et $N_{1,1} = 1$ donc la matrice d'adjacence vaut $\begin{pmatrix} 1 & 0 \\ \frac{h-1}{h} & \frac{1}{h} \end{pmatrix}$.

Question 5
On remarque d'abord que l'épaisseur d'un arbre semble mesurer la densité des nœuds de degré 2 : plus l'arbre est effilé moins l'épaisseur est importante On voit aussi que l'apparition d'arêtes d'épaisseur importante demande des nœuds d'épaisseur de la forme (k,k) : la matrice aura des termes diagonaux dominants

Question 6
Un arbre de taille $n \geqslant 1$ admet au moins un nœud terminal qui aura donc une bi-épaisseur $(1,1)$ ainsi il existe toujours une arête d'épaisseur 2 On a vu que les arbres peignes avaient une épaisseur 2 ; ainsi $\lambda(n) = 2$.
Pour obtenir une épaisseur 2 il faut que tous les nœuds aient $(1,1)$ ou $(1,2)$ comme bi-épaisseur. En particulier un arbre de hauteur n admet au moins un fils d'épaisseur 1 donc une feuille, l'autre étant un arbre de hauteur $n-1$ et d'épaisseur 2 au plus. Si on note $\mu(n)$ le nombre d'arbres de hauteur n et d'épaisseur 2 on a donc $\mu(1) = 1$ et $\mu(n) = 2\mu(n-1)$ car on choisit le fils de hauteur $n-1$. On aboutit à $\mu(n) = 2^{n-1}$, c'est en fait le nombre d'arbres-peignes.

Question 7
Un arbre complet de hauteur $h-1$ admet 2^h-1 nœuds et son épaisseur est$h$. De plus, une épaisseur h n'est possible que s'il existe un nœud de bi-épaisser $(h-1, h-1)$ donc deux nœuds d'épaisseur $h-1$ d'où, par récurrence, 2^{h-1} nœuds terminaux de bi-épaisseur $(1,1)$. Ainsi un arbre d'épaisseur h admet au moins $\displaystyle\sum_{k=0}^{h-1} 2^k = 2^h - 1$ nœuds : on a $n(h) \geqslant 2^h - 1$ avec la borne atteinte.
Si on considère un arbre-peigne de taille p et que l'on remplace une feuille par un arbre complet de hauteur $h-1$ on obtient un arbre d'épaisseur h et de taille $p+2^h-1$ donc il n'y a pas de majorant possible pour $n(h)$.
Les valeurs possibles pour $n(h)$ sont donc les entiers supérieurs ou égaux à $2^h - 1$.

Question 8
Pour stocker une structure arborescente on doit stocker, pour chaque nœud, les informations relatives au nœud ainsi que qu'un moyen de déterminer ses fils. L'encombrement en mémoire est donc proportionnel au nombre de nœuds, indépendamment du langage.
En Caml, un typage minimal possible est

```
type forme_a = Vide|Noeud of forme_a  * forme_a;;
```

Question 9

```
let rec epaisseur arbre =
  match arbre with
  |Vide -> 1
  Noeud(g, d) -> let eg = epaisseur g in
                    let ed = epaisseur d in
                    if eg = ed then eg + 1 else max eg ed ;;
```

Question 10
La fonction existe en OCaml : c'est `Random.random`. On commence par une fonction de choix, une variable aléatoire suivant une loi de Bernoulli de paramètre $0,5$.

```
let choix () = Random.random() < 0.5;;
```

On écrit la fonction d'ajout d'un nœud :

```
let rec ajout arbre =
  match arbre with
  |Vide -> Noeud(Vide, Vide)
  Noeud(g, d) -> if choix()
                 then Noeud(ajout g, d)
                 else Noeud(g, ajout d);;
```

On peut enfin ajouter n nœuds à un arbre vide :

```
let creation n =
  let arbre = ref Vide in
  for i = 1 to n do
    arbre := ajout !arbre done;
  !arbre;;
```

Au pire, lors de la création d'un arbre-peigne, la complexité, en nombre d'appels récursifs sera $\sum_{k=0}^{n-1} k$, c'est un $\mathcal{O}(n^2)$.

Question 11
La question est mal rédigée :

- elle n'utilise pas la matrice
- elle ne précise rien sur $r2_k$, qui devrait s'appeler plutôt $r2(k)$.

Un algorithme possible serait alors

```
entrée : une matrice M
s est le nombre de ligne de M
t est un arbre avec 1 nœud, d'épaisseur s
tant qu'il reste des feuille d'épaisseur différente de 1
    choisir une telle feuille f
    k est son épaisseur
    i = r2(k)
    si i = k
        alors a = k, b = k
        sinon si r1() < 0.5
            alors a = i, b =k
            sinon a = k, b = 1
    ajouter une feuille d'épaisseur a comme fils gauche de f
    ajouter une feuille d'épaisseur b comme fils droit de f
```

On a tiré au sort les tailles des fils pour ne pas déséquilibre l'arbre.
Si la probabilité de $r2(k) = 1$ n'est pas nulle l'algorithme pourrait ne pas terminer mais la probabilité que le nombre d'étape soit supérieur à N tend vers 0 quand N grandit sauf si $r2(k) = 1$ est de probabilité 1.

Question 12
L'hypothèse faite est que $r2$ dépende aussi de M avec $P\big(r2(k, M) = i\big) = M_{k,i}$.
Dans ce cas on peut imaginer que l'arbre construit (si l'algorithme termine) aura une matrice de ramification proche de M.

2 Deuxième partie

Question 13
Évaluation de gauche à droite de $a + (b - c) * (d - e)$.

instruction	R_0	R_1	R_2	R_3
$R_0 \leftarrow a$	a			
$R_1 \leftarrow b$	a	b		
$R_2 \leftarrow c$	a	b	c	
$R_1 \leftarrow R_1 - R_2$	a	$b - c$	c	
$R_2 \leftarrow d$	a	$b - c$	d	
$R_3 \leftarrow e$	a	$b - c$	d	e
$R_2 \leftarrow R_2 - R_3$	a	$b - c$	$d - e$	e
$R_1 \leftarrow R_1 * R_2$	a	$(b - c) * (d - e)$	$d - e$	e
$R_0 \leftarrow R_0 + R_1$	$a + (b - c) * (d - e)$	$(b - c) * (d - e)$	$d - e$	e

On utilise 4 registres

Question 14
Évaluation de droite à gauche de $a + (b - c) * (d - e)$.

instruction	R_0	R_1	R_2	R_3
$R_0 \leftarrow e$	e			
$R_1 \leftarrow d$	e	d		
$R_0 \leftarrow R_1 - R_0$	$d - e$	d		
$R_1 \leftarrow c$	$d - e$	c		
$R_2 \leftarrow b$	$d - e$	c	b	
$R_1 \leftarrow R_2 - R_1$	$d - e$	$b - c$	b	
$R_0 \leftarrow R_1 * R_0$	$(b - c) * (d - e)$	$b - c$	b	
$R_1 \leftarrow a$	$(b - c) * (d - e)$	a	b	
$R_0 \leftarrow R_0 + R_1$	$a + (b - c) * (d - e)$	a	b	

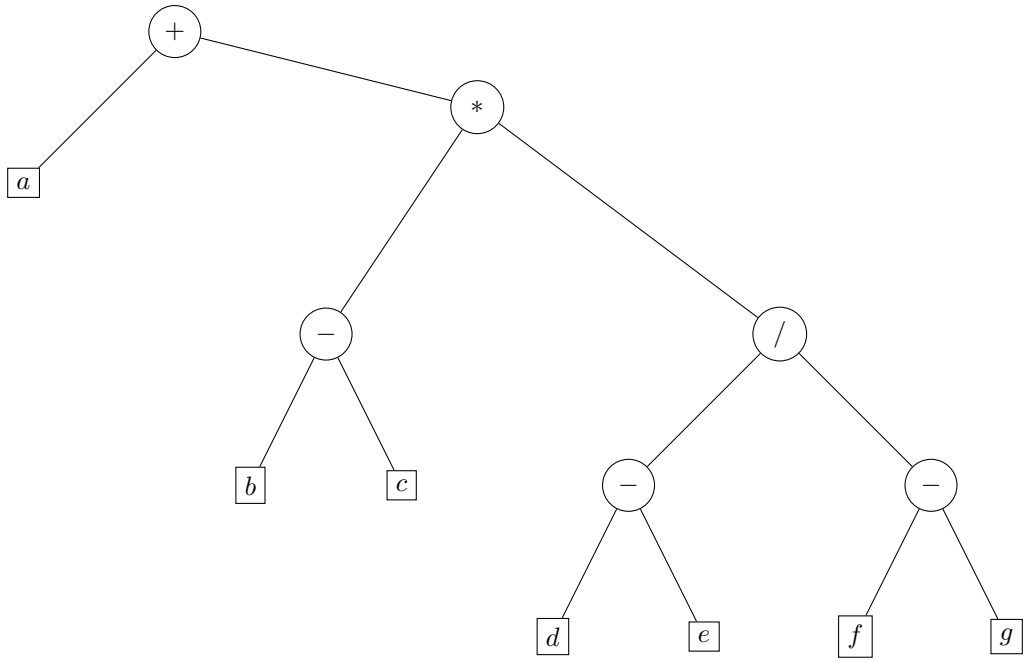
On utilise 3 registres

Question 15
Dans la somme le second terme demande plus de registres, de même dans le produit.
Dans le quotient les deux termes demandent 2 registres.

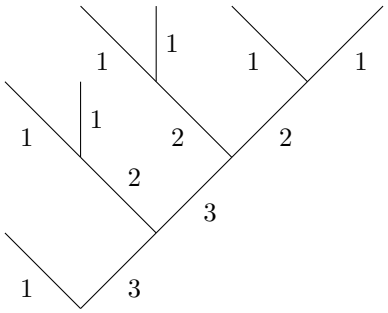
instruction	R_0	R_1	R_2	R_3
$R_0 \leftarrow d$	d			
$R_1 \leftarrow e$	d	e		
$R_0 \leftarrow R0 - R_1$	$d - e$	e		
$R_1 \leftarrow f$	$d - e$	f		
$R_2 \leftarrow g$	$d - e$	f	g	
$R_1 \leftarrow R_1 + R_2$	$d - e$	$f + g$	g	
$R_0 \leftarrow R_0 / R_1$	$(d - e)/(f + g)$	$f + g$	g	
$R_1 \leftarrow b$	$(d - e)/(f + g)$	b	g	
$R_2 \leftarrow c$	$(d - e)/(f + g)$	b	c	
$R_1 \leftarrow R1 + R2$	$(d - e)/(f + g)$	$b + c$	c	
$R_0 \leftarrow R1 * R0$	$(b - c) * (d - e)/(f + g)$	$b + c$	c	
$R_1 \leftarrow a$	$(b - c) * (d - e)/(f + g)$	a	c	
$R_0 \leftarrow R1 + R_0$	$a + (b - c) * (d - e)/(f + g)$	a	c	

On utilise 3 registres.

Question 16



Question 17



On montre par induction structurelle que l'épaisseur de la forme arborescente associée à une expression arithmétique est égale au nombre de registres utilisés pour l'évaluation de cette expression avec la stratégie exposée à la question 15.

- C'est vrai si l'expression arithmétique est réduite à une valeur car, dans ce cas, la forme associée est une feuille, dont l'épaisseur vaut 1, et un registre suffit pour charger une valeur.
- Pour une expression $e_1 \heartsuit e_2$ avec e_1 demandant p registres et e_2 demandant q registres, on suppose que les expressions sont associées à deux arbres d'épaisseurs respectives p et q.
  - Si $p < q$ on commence par calculer e_2, ce qui demande q registres. On stocke le résultat dans R_0. Il reste alors les registre R_1 à R_{q-1} avec $q-1 \geqslant p$ dans lesquels on peut effectuer le calcul de e_2. On met ensuite le résultat dans $R81$ et il reste à calculer $R_0 \heartsuit R_1$ que l'on place dans R_0. On utilise donc q registres, ce qui est l'épaisseur de l'arbre associé car ses deux fils ont p et q comme épaisseur avec $p < q$.
  - De même, si $p > q$, on utilise p registres et l'épaisseur de l'arbre est p.
  - Si $p = q$ alors on utilisera $p + 1$ registres.
    En effet on utilise p registre pour e_1 dont on place le résultat dans R_0 et on a besoin des p registres R_1 à R_p pour calculer e_2 dont la valeur est placée dans R_1. On finit par $R_0 \leftarrow R_0 \heartsuit R_1$. On a donc besoin de $p + 1$ registres.
    $p + 1$ est aussi de l'épaisseur de l'arbre associé car ses deux fils ont p comme épaisseur.

3 Troisième partie

L'énoncé est défaillant.

- Il ne signale pas quel coté est utilisé pour être considéré comme une branche.
- Les mots bien emboîtés ne correspondent pas à ce qui est attendu. Par exemple $[[a]]$ est bien emboîté alors qu'il ne correspond pas à un arbre. Je remplace la définition par la suivante.
  1. a est bien emboîté.
  2. Si u et v sont bien emboîtés alors uv est bien emboîté.
  3. Si u, v et w sont bien emboîtés alors $u[v]w$ est bien emboîté.

  On en déduit (avec 1. et 2.) que tout mot non vide de Σ^* est bien emboîté.

Question 18
Une forme n'a pas de codage unique, seule une représentation pourrait avoir une unicité si on suppose qu'on a toujours un des fils qui se dessine dans le prolongement. Il est probable que le sujet attendait le codage qui consiste à remplacer chaque lettre par un a dans l'exemple fourni : $aa[aa]aa[a[a]a]a[aaa]aa$.
Un autre codage possible pourrait être $aa[aa[a[aaa]aa]a[a]a]aa$

Question 19
$[w]$ est une branche si et seulement si w est un arbre.
La question demande donc de reconnaître les codages des arbres.
Prouvons que les codages des arbres non vides sont les mots bien emboîtés.

- Une feuille est codée par a, bien emboîté.
- Si un arbre f est codé par un mot bien emboîté non vide u alors l'arbre qui a f pour seul fils est codé par au donc est bien emboîté d'après le cas 2.
- Si g et d sont deux arbres codés par u et v respectivement avec u et v bien emboîtés alors l'arbre de fils g et d est codé par $a[u]v$ qui est bien emboîté d'après le cas 3.

Ainsi, par induction structurelle, tout codage d'un arbre est bien emboîté.

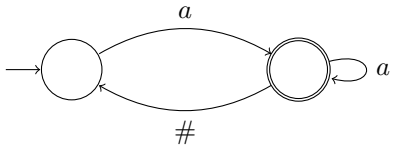
On prouve ensuite que tout mot bien emboîté code un arbre ; la démonstration se fait aussi par induction structurelle.
On commence par remarquer que si u code un arbre, il finit par a qui correspond à une feuille, on nommera feuille finale cette feuille.

- a code un arbre réduit à une feuille.
- u et v, mots bien emboîtés, codent les arbres a_1 et a_2 alors uv code l'arbre obtenu en ajoutant a_2 comme unique fils de la feuille finale (pour le codage u) de a_1.
- u, v et w, mots bien emboîtés, codent les arbres a_1, a_2 et a_3 alors $u[v]w$ code l'arbre obtenu en ajoutant a_2 et a_3 comme fils de la feuille finale (pour le codage u) de a_1.

Question 20
Ici encore, on se ramène aux arbres (en enlevant les crochets externes).
On notera que l'énoncé est faux car il autorise des branches vides.
D'après la question précédente, il suffit de prouver que tout mot bien emboîté peut s'écrire sous la forme $u_0[v_1]u_1[v_2]\cdots u_{n-1}[v_n]u_n$ avec $u_i \in \Sigma^+$, u_i non vide et v_i bien emboîté.
La démonstration peut se faire par induction structurelle, comme ci-dessus.

Question 21
Les axes marqués sont les mots de l'ensemble AM défini par

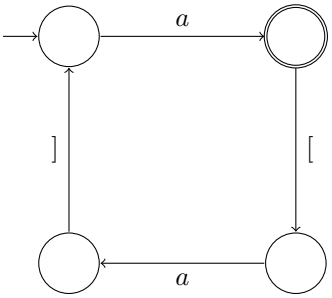
1. $a^n \in AM$
2. Si u et v appartiennent à AM, alors $u\#v \in AM$.



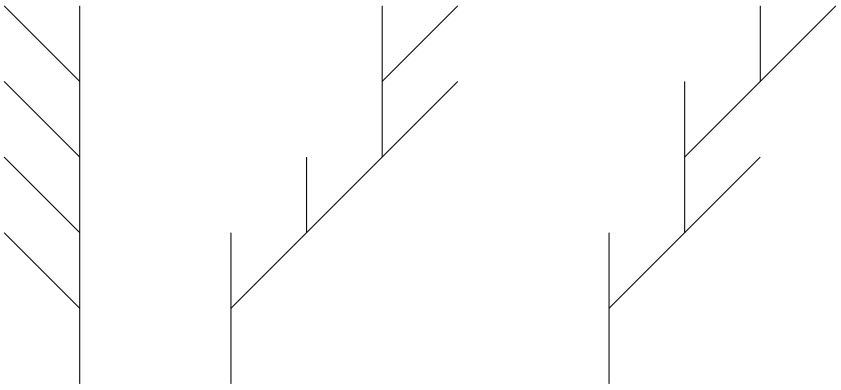
Question 22
Comme à chaque étape les deux fils d'un arbre parfait sont égaux, un arbre parfait d'épaisseur n admet un codage unique w_n avec $w_1 = a$ et $w_{n+1} + a[w_n]w_n$. On peut en déduire la fonction

```
let rec code_parfait w =
  match String.length w with
  |1 -> w.[0] = 'a'
  |n when n mod 2 = 0 -> false
  |n -> let p = (n - 3)/2 in
        let w1 = String.sub w 2 p in
        let w2 = String.sub w (p+3) p in
        w.[0] = 'a' || w.[1] = '[' || w.[p+2] = ']' || w1 = w2
                    || code_parfait w1 || code_parfait w2;;
```

Question 23
Les arbres peignes sont de la forme $a[a]a[a]a\cdots[a]a[a]a$; on peut les écrire $a([a]a)^n$ avec $n \in \mathbb{N}$. Un automate possible est donc



En fait, ceci est peut être la solution attendue mais elle n'est pas exacte. En effet, pour écrire le code ci-dessus, on a supposé qu'on mettait dans la branche le fils réduit à une feuille mais ceci n'est pas spécifié. Un même arbre peigne peut être vu de plusieurs façons
Voici des arbres peignes de hauteur 5 :



Si considère que les branches sont les fils qui ne sont pas dans le prolongement dessiné ils sont codés respectivement $a[a]a[a]a[a]a[a]a$, $a[a[a]a[a[a]a]a]a$ et $a[a[a[a]a]a]a]a$.
Les codes forment l'ensemble AP défini par

1. $a \in AP$
2. Si $u \in AP$ alors $a[a]u$ et $a[u]a$ appartiennent à AP.

Cet ensemble n'est pas rationnel donc ne peut pas être défini par un automate.

Question 24
En recherchant le premier nœud à deux fils dans un arbre on peut écrire le code d'un arbre sous a forme $a^n[u]v$ avec u et v qui sont les codage d'un arbre (les deux fils) On va donc chercher l'apparition d'un caractère différent de a, qui doit être '['. On cherche ensuite le ']' qui lui correspond. On finit récursivement en testant les parties délimitées par ces crochets.

```
let rec est_code w =
  let n =  String.length w in
  if n = 0
  then false
  else let p = ref 0 in
    while !p < n && w.[!p] = 'a' do p := !p + 1 done;
    if !p = n
    then true
    else let nb = ref 1 in
      let q = ref (!p + 1) in
      while !nb > 0 && !q < n do
        if w.[!q] = '[' then nb := ! nb + 1;
        if w.[!q] = ']' then nb := !nb - 1;
        q := !q +1 done;
      q := !q - 1;
      if !q = n - 1
      then false
      else let u = String.sub w (!p + 1) (!q - !p - 1) in
        let v = String.sub w (!q + 1) (n - !q - 1) in
        est_code u && est_code v;;
```

Question 25
On peut commencer par la séparation du code sous la forme $a^p[u]w$, on reprend le code ci-dessus.

```
let structure w =
  let n =  String.length w in
  if n = 0
  then 0, "", ""
  else let p = ref 0 in
    while !p < n && w.[!p] = 'a' do p := !p + 1 done;
    if !p = n
    then n, "", ""
    else let nb = ref 1 in
      let q = ref (!p + 1) in
      while !nb > 0 && !q < n do
        if w.[!q] = '[' then nb := ! nb + 1;
        if w.[!q] = ']' then nb := !nb - 1;
        q := !q +1 done;
      q := !q - 1;
      if !q = n - 1
      then faiwith "Ce n'est pas un code valide
      else let u = String.sub w (!p + 1) (!q - !p - 1) in
        let v = String.sub w (!q + 1) (n - !q - 1) in
        p, u, v;;
```

On peut alors dessiner les arbres.

```
let dessin w =
  open_graph " 800x900";
  let d = 40 in
  let pi_sur_2 = 2.0 *. (atan 1.0) in
  let rec aux w x0 y0 a da =
    if w <> ""
    then begin
      moveto x0 y0;
      let dda = ref da in
      let x1 = ref x0 in
      let y1 = ref y0 in
      let dx = int_of_float ((float_of_int d) *. (cos a)) in
      let dy = int_of_float ((float_of_int d) *. (sin a)) in
      let p, u, v = structure w in
      for i = 1 to p do
        x1 := !x1 + dx;
        y1 := !y1 + dy;
        lineto !x1 !y1;
        dda := !dda *. 0.63 done;
      aux u !x1 !y1 (a +. !dda) !dda;
      aux v !x1 !y1 (a -. !dda) !dda; () end in
  aux w 400 100 pi_sur_2 pi_sur_2;
  let _ = wait_next_event [Button_down] in
  close_graph();;
```