SESSION 2021 **MP7IN**

ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE MP

INFORMATIQUE

Durée : 4 heures

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- *Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.*
- *Ne pas utiliser de correcteur.*
- *Écrire le mot FIN à la fin de votre composition.*

Les calculatrices sont interdites.

Le sujet est composé de trois parties indépendantes.

Partie I - Étude des partitions non croisées

Dans cette partie, on introduit et on étudie de façon élémentaire les partitions non croisées, objets combinatoires apparaissant dans divers domaines des mathématiques, notamment dans la théorie des probabilités libres et des matrices aléatoires.

Le langage de programmation utilisé dans cette partie est le langage Python.

Dans la suite, pour tout couple $(i,n) \in \mathbb{N}^2$, les notations $[\![n]\!]$ et $[\![i,n]\!]$ désignent respectivement les ensembles $\{1,2,\dots,n\}$ et $\{i,i+1,\dots,n\}$. Par convention, $[\![0]\!] = \emptyset$ et si $i > n$ alors $[\![i,n]\!] = \emptyset$.

Définition 1 (Partition)
Soit A un sous-ensemble non vide de $\mathbb{N}$. Une **partition** $\mathcal{P}$ de A est un ensemble de parties de A tel que :

1. $\forall P \in \mathcal{P}, P \neq \emptyset$; 2. $\forall (P,Q) \in \mathcal{P}^2, (P \neq Q) \implies (P \cap Q = \emptyset)$; 3. $\bigcup_{P\in\mathcal{P}} P = A$.

Par exemple, $\{\{2,4,5\},\{7,9\},\{8\}\}$ est une partition de l'ensemble $\{2,4,5,7,8,9\}$.

Définition 2 (Classe)
Soit $\mathcal{P}$ une partition d'un sous-ensemble A non vide de $\mathbb{N}$. Soit i un élément de A. La **classe** de i est l'unique élément P de $\mathcal{P}$ tel que $i \in P$. Elle est notée $\mathrm{Cl}(i)$.

Par exemple, pour $\mathcal{P} = \{\{2,4,5\},\{7,9\},\{8\}\}$, on a $\mathrm{Cl}(4) = \{2,4,5\}$.

Définition 3 (Partition non croisée)
Soit $\mathcal{P}$ une partition d'un sous-ensemble A non vide de $\mathbb{N}$. On dit que $\mathcal{P}$ est **non croisée** si pour tout quadruplet $(a,b,c,d) \in A^4$ tel que $a < b < c < d$ on a :

$$(\mathrm{Cl}(a) = \mathrm{Cl}(c), \mathrm{Cl}(b) = \mathrm{Cl}(d)) \implies (\mathrm{Cl}(a) = \mathrm{Cl}(b) = \mathrm{Cl}(c) = \mathrm{Cl}(d)).$$

Par exemple, $\mathcal{P} = \{\{2,4,5\},\{7,9\},\{8\}\}$ est bien une partition non croisée de $\{2,4,5,7,8,9\}$. En effet, aucun quadruplet $(a,b,c,d) \in \{2,4,5,7,8,9\}^4$ tel que $a < b < c < d$ ne vérifie la condition $\mathrm{Cl}(a) = \mathrm{Cl}(c), \mathrm{Cl}(b) = \mathrm{Cl}(d)$.

De même, $\mathcal{Q} = \{\{2,7,8,9\},\{4,5\}\}$ en est bien une. En effet, $(a,b,c,d) = (2,7,8,9)$ est l'unique quadruplet tel que $a < b < c < d$ vérifiant $\mathrm{Cl}(a) = \mathrm{Cl}(c), \mathrm{Cl}(b) = \mathrm{Cl}(d)$ et ces quatre éléments sont bien dans la même classe.

Par contre, $\mathcal{R} = \{\{2,5\},\{4,7,9\},\{8\}\}$ n'en est pas une. En effet, $\mathrm{Cl}(2) = \mathrm{Cl}(5)$ et $\mathrm{Cl}(4) = \mathrm{Cl}(7)$ mais $\mathrm{Cl}(2) \neq \mathrm{Cl}(4)$.

Le terme « non croisée » découle naturellement des représentations picturales des partitions (**figure 1**).

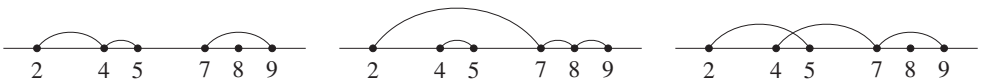

Figure 1 - Représentations picturales de $\mathcal{P}, \mathcal{Q}, \mathcal{R}$

Dans la suite, on s'intéresse uniquement aux partitions d'un sous-ensemble fini de $\mathbb{N}$. On les représente en Python à l'aide de listes de listes croissantes d'entiers triées dans l'ordre lexicographique.

Par exemple, les partitions $\mathcal{P} = \{\{2,4,5\},\{7,9\},\{8\}\}$ et $\mathcal{R} = \{\{2,5\},\{4,7,9\},\{8\}\}$ sont respectivement représentées par les listes `P = [[2,4,5],[7,9],[8]]` et `R = [[2,5],[4,7,9],[8]]`.

I.1 - Exemples et fonctions élémentaires (Informatique Pour Tous)

Q1. Justifier brièvement que $\mathcal{P} = \{\{1,7\},\{2\},\{3,4,5\},\{6\}\}$ est une partition non croisée de $[\![7]\!]$ et que $\mathcal{Q} = \{\{1,6\},\{2\},\{3,4,5,7\}\}$ n'en est pas une.

Q2. Parmi les ensembles suivants, indiquer sans justification lesquels sont des partitions de $[\![5]\!]$. Parmi les partitions, préciser sans justification lesquelles sont non croisées.

1. $\mathcal{P}_1 = \{\{1,3\},\{2,4,5\}\}$,
2. $\mathcal{P}_2 = \{\{1,3\},\{1,2\},\{4,5\}\}$,
3. $\mathcal{P}_3 = \{\{1,3\}\{2,4\}\}$,
4. $\mathcal{P}_4 = \{\{1,4,5\},\{2,3\}\}$.

Q3. Décrire l'ensemble des partitions non croisées de $[\![4]\!]$.

Pour **Q4**, **Q5** et **Q6**, on se fixe un ensemble fini $A \subset \mathbb{N}$. Cet ensemble est représenté en Python par la liste croissante d'entiers `A`, définie au préalable. On suppose également que pour ces questions, la variable `P` désigne une liste de listes croissantes d'entiers triée dans l'ordre lexicographique.

On définit la fonction Python `mystere(P)` par :

```
def mystere(P) :
    L = [False for _ in range(len(A))]
    for i in range(len(P)) :
        if P[i] == [] :
            return False
        else :
            for j in range(len(P[i])) :
                if P[i][j] not in A :
                    return False
                else :
                    # A.index(a) renvoie l'indice de a dans A
                    k = A.index(P[i][j])
                    if L[k] :
                        return False
                    else :
                        L[k] = True
    return not (False in L)
```

Q4. Uniquement dans cette question, on suppose que `A = [1,2,3,4,5]`.
Expliciter sans justification les valeurs de :

1. `mystere([[1,3],[1,5],[2,4]])`
2. `mystere([[1,3,4],[2]])`
3. `mystere([[1,3,5],[2,4]])`
4. `mystere([[],[1,2,3,4,5]])`.

Q5. Écrire une fonction Python `classe(P,i)` prenant en arguments une variable `P` représentant une partition $\mathcal{P}$ de A et un entier $i \in A$ qui renvoie une liste L représentant $\mathrm{Cl}(i)$.

On définit la fonction Python au code incomplet `est_nc(P)` suivante :

```
def est_nc(P) :
    # On rappelle que A est une liste triée dans l'ordre croissant
    N = len(A)
    for i in range(N) :
        for j in range(i+1,N) :
            for k in range(j+1,N) :
                for l in range(k+1,N) :
                    . . .
```

Q6. Recopier et compléter le code de la fonction `est_nc(P)`, sachant qu'elle prend en argument une variable `P` représentant une partition $\mathcal{P}$ de A et qu'elle renvoie `True` si $\mathcal{P}$ est non croisée et `False` sinon.

I.2 - Nombre de partitions non croisées

Pour tout $n \geq 1$, on note C_n le nombre de partitions non croisées d'un ensemble $A \subset \mathbb{N}$ ayant exactement n éléments et $\mathrm{NC}(A)$ l'ensemble des partitions non croisées de A. Par convention, $C_0 = 1$ et $\mathrm{NC}(\emptyset) = \{\emptyset\}$.

Définition 4 (Partition non croisée emboîtée)
Soient A un ensemble fini de $\mathbb{N}$ de cardinal n et $\mathcal{P}$ une partition non croisée de A. On dit que $\mathcal{P}$ est **emboîtée** si le minimum m et le maximum M de A sont dans la même classe ($\mathrm{Cl}(m) = \mathrm{Cl}(M)$). L'ensemble des partitions non croisées et emboîtées de A et son cardinal sont respectivement notés $\mathrm{NCE}(A)$ et D_n.

Par exemple, $\mathcal{S} = \{\{1, 2, 5, 9\}, \{3, 4\}, \{6, 7, 8\}\}$ est une partition non croisée et emboîtée de $[\![9]\!]$.

Le terme « emboîtée » découle naturellement des représentations picturales des partitions (**figure 2**).

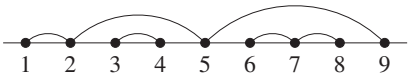

Figure 2 - Représentations picturale de $\mathcal{S}$

Q7. Montrer que pour tout $n \in \mathbb{N}$, on a $C_n = D_{n+1}$.

Q8. Soit $n \in \mathbb{N}$. On définit l'application Φ_n comme suit :

$$\begin{array}{rccc} \Phi_n : & \displaystyle\bigcup_{i=1}^{n+1} \mathrm{NCE}([\![i]\!]) \times \mathrm{NC}([\![i+1, n+1]\!]) & \to & \mathrm{NC}([\![n+1]\!]) \\ & (Q_1, Q_2) & \mapsto & Q_1 \cup Q_2. \end{array}$$

En admettant que Φ_n est une application bijective, montrer que $C_{n+1} = \displaystyle\sum_{k=0}^{n} C_k C_{n-k}$.

Q9. (Informatique Pour Tous) Écrire une fonction Python `calcul_C(n)` qui prend en argument un entier naturel `n` et qui renvoie la valeur C_n.

Partie II - Logique et étude du problème Horn-Sat

Dans cette partie, $\wedge, \vee, \neg$, désignent respectivement les connecteurs de conjonction, de disjonction et de négation. Les symboles V, F désignent respectivement les valeurs de vérité VRAI et FAUX du calcul propositionnel.

Dans la suite, on s'intéresse au problème Horn-Sat, qui peut être décrit de la façon suivante : étant donnée une formule de Horn P, existe-t-il une interprétation I telle que $[P]_I = \mathrm{V}$?

L'objectif est d'expliciter un algorithme permettant de résoudre ce problème.

Définition 5 (Formule propositionnelle)
Soit X un ensemble de symboles appelés variables propositionnelles. Les **formules propositionnelles** sur X sont alors définies de façon inductive comme suit :

- V, F sont des formules propositionnelles ;
- tout élément x de X est une formule propositionnelle ;
- si P et Q sont des formules propositionnelles, alors $\neg P, (P \wedge Q), (P \vee Q)$ sont des formules propositionnelles.

Dans la suite, les variables propositionnelles et formules propositionnelles considérées sont construites à l'aide d'un ensemble X qui ne sera pas explicité.

Définition 6 (Littéral)
Soit x une variable propositionnelle. Un **littéral** est la formule x ou la formule $\neg x$. Le littéral x (respectivement $\neg x$) est un littéral positif (respectivement négatif).

Définition 7 (Clause disjonctive)
Soient $n \in \mathbb{N}$, $x_1, x_2, \ldots, x_n$ des littéraux. La formule $x_1 \vee x_2 \vee \ldots \vee x_n$ est appelée **clause disjonctive** à n littéraux (ou de taille n). Par convention, () désigne la **clause disjonctive vide**, c'est-à-dire la clause sans littéral.

Une **clause de Horn** est une clause disjonctive contenant au plus un littéral positif.

Une **clause unitaire** est une clause composée d'un unique littéral.

Définition 8 (Forme normale conjonctive)
Soit P une formule propositionnelle. On dit que P est une **forme normale conjonctive** s'il existe un entier $n \in \mathbb{N}$, $C_1, C_2, \ldots, C_n$ des clauses disjonctives vérifiant $P = C_1 \wedge C_2 \wedge \ldots \wedge C_n$. Par convention, une forme normale conjonctive sans clause est représentée par $\emptyset$.

Définition 9 (formule de Horn)
Soit P une formule propositionnelle. On dit que P est une **formule de Horn** s'il existe $n \in \mathbb{N}, C_1, C_2, \ldots, C_n$ des clauses de Horn vérifiant $P = C_1 \wedge C_2 \wedge \ldots \wedge C_n$.

Définition 10 (Interprétation)
Soient $n \in \mathbb{N}$ et $x_1, x_2, \ldots x_n$ des variables propositionnelles. Une **interprétation** est une application $I : \{x_1, x_2, \ldots, x_n\} \to \{\mathrm{V}, \mathrm{F}\}$.

Définition 11 (Évaluation)
Soient P une formule propositionnelle, $x_1, x_2, \ldots, x_n$ les variables propositionnelles apparaissant dans P et I une interprétation sur $\{x_1, x_2, \ldots, x_n\}$. **L'évaluation** de P suivant l'interprétation I que l'on note $[P]_I$ est définie par récurrence de la façon suivante :

- si $P \in \{\mathrm{V}, \mathrm{F}\}$, alors $[P]_I = P$;
- si $P \in \{x_1, x_2, \ldots, x_n\}$, alors $[P]_I = I(P)$;
- si $P = \neg Q$ où Q est une formule propositionnelle, alors $[P]_I = \neg[Q]_I$;
- si $P = A \circ B$ où A, B sont des formules propositionnelles et $\circ \in \{\wedge, \vee\}$, alors $[P]_I = [A]_I \circ [B]_I$.

Par convention, $[()]_I = \mathrm{F}$ et $[\emptyset]_I = \mathrm{V}$.

Définition 12 (Satisfiable)
Soit P une formule propositionnelle. On dit que P est **satisfiable** s'il existe une interprétation I telle que $[P]_I = \mathrm{V}$.

Q10. Pour chacune des formules suivantes, montrer qu'elle est satisfiable ou qu'elle est non satisfiable.

1. $P_1 = (x \vee y) \wedge (\neg x \vee \neg y) \wedge (x \vee \neg y)$,
2. $P_2 = (x) \wedge (\neg x \vee \neg y) \wedge (\neg y \vee z) \wedge (z)$,
3. $P_3 = ()$,
4. $P_4 = (x \vee \neg y \vee \neg t) \wedge (z \vee \neg t \vee \neg x \vee \neg y)$.

Q11. Parmi les fomules de la **Q10**, lesquelles sont des formules de Horn ? On ne justifiera pas la réponse.

Définition 13 (Propagation unitaire)
Soit P une forme normale conjonctive contenant une clause unitaire (x). On construit une formule P' à partir de P de la façon suivante :

1. supprimer de P toutes les clauses où x apparaît ;
2. enlever toutes les occurrences du littéral $\neg x$.

Cette procédure de simplification est appelée **propagation unitaire**.

Par exemple, si $P = (x \vee \neg y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x)$, on a alors $P' = (y \vee z)$. Si $P = (x) \wedge (\neg x)$, on a alors $P' = ()$.

Dans la suite, on note $\Pi(P)$ une formule sans clause unitaire obtenue en itérant la propagation unitaire sur P. On admet que si $\Pi(P)$ ne contient pas de clause vide $()$ alors $\Pi(P)$ est unique et que si $\Pi(P)$ contient au moins une clause vide alors toute formule sans clause unitaire obtenue par itération de la propagation unitaire sur P contient au moins une clause vide.

Q12. On pose :

$$\begin{aligned} P \;=\; & (x_1) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_2 \vee \neg x_1 \vee x_3) \\ & \wedge (x_3 \vee \neg x_4 \vee x_5) \wedge (\neg x_1 \vee x_2 \vee x_5). \end{aligned}$$

Calculer $\Pi(P)$. On pourra donner le résultat directement sans détailler les calculs.

Q13. Soit P une formule de Horn. Montrer que $\Pi(P)$ est une formule de Horn.

Q14. Soit P une forme normale conjonctive. Montrer que P est satisfiable si et seulement si $\Pi(P)$ est satisfiable.

Q15. Soit P une forme normale conjonctive. Montrer que si $()$ apparaît dans $\Pi(P)$, alors P n'est pas satisfiable.

Q16. Soit C une clause de Horn. Montrer que si C n'est ni la clause vide ni une clause unitaire positive, alors C est satisfiable.

Q17. Soit P une formule de Horn ne contenant ni de clause vide ni de clause unitaire positive. Montrer que P est satisfiable.

Q18. Soit P une formule de Horn. Montrer que P n'est pas satisfiable si et seulement si $()$ apparaît dans $\Pi(P)$.

Partie III - Étude des classes sylvestres

Étant donné un arbre binaire de recherche T, sa classe sylvestre est l'ensemble des mots qui donnent l'arbre T après insertion dans l'arbre vide. L'objectif de cette partie est de donner une caractérisation et une description de cet ensemble. On commence par rappeler la structure des arbres binaires de recherche ainsi que leurs propriétés usuelles. Puis, on introduit la notion de S-équivalence sur les mots qui permet de caractériser la classe sylvestre d'un arbre T. Enfin, à l'aide du produit de mélange, on présente un algorithme calculant la classe sylvestre d'un arbre donné.

Dans toute la suite, Σ désigne un alphabet fini totalement ordonné. Le symbole ε désigne le mot vide.

III.1 - Algorithme d'insertion dans un arbre binaire

Définition 14 (Arbre binaire)
Un **arbre binaire** T (étiqueté par les éléments de Σ) est récursivement soit :

- l'arbre vide que l'on note $\circ$;
- un triplet (T_g, r, T_d) où r est un élément de Σ, T_g et T_d des arbres binaires. Les éléments r, T_g et T_d sont respectivement appelés racine, sous-arbre gauche et sous-arbre droit de T.

Définition 15 (Arbre binaire de recherche)
Un **Arbre Binaire de Recherche** (abrégé en **ABR**) T est récursivement soit :

- l'arbre vide ;
- un triplet (T_g, r, T_d) où r est un élément de Σ, T_g et T_d des **ABR**. De plus, toute valeur apparaissant dans T_g est strictement inférieure à r et toute valeur apparaissant dans T_d est supérieure ou égale à r.

Définition 16 (Insertion dans un arbre binaire)
L'**insertion** d'un élément a de Σ dans un arbre binaire T est une opération que l'on note $T \leftarrow a$ définie récursivement comme suit :

- si $T = \circ$, alors $T \leftarrow a = (\circ, a, \circ)$;
- si $T = (T_g, r, T_d)$ et $r \leq a$, alors $T \leftarrow a = (T_g, r, T_d \leftarrow a)$;
- si $T = (T_g, r, T_d)$ et $r > a$, alors $T \leftarrow a = (T_g \leftarrow a, r, T_d)$.

On définit récursivement alors l'insertion d'un mot w dans un arbre binaire T noté également $T \leftarrow w$ comme suit :

- si $w = \varepsilon$, alors $T \leftarrow w = T$;
- si $w = av$ avec $a \in \Sigma$, alors $T \leftarrow w = (T \leftarrow a) \leftarrow v$.

Dans le cas où T est un **ABR**, on admet que $T \leftarrow a$ et $T \leftarrow w$ sont également des **ABR**.

Étant donné un mot w sur Σ, **l'arbre binaire de recherche associé** à w est l'arbre $\circ \leftarrow w$.

Définition 17 (Classe sylvestre)
Soit T un arbre binaire de recherche. La **classe sylvestre** de T que l'on note $\mathrm{Syl}(T)$ est l'ensemble des mots w vérifiant : $(\circ \leftarrow w) = T$.

Par exemple, si $T = ((\circ, a, \circ), b, (\circ, c, \circ))$, on a $\mathrm{Syl}(T) = \{bac, bca\}$.

Et si $T = \circ$, alors $\mathrm{Syl}(T) = \{\varepsilon\}$.

Pour simplifier la représentation des arbres binaires, on peut utiliser des graphes.

Par exemple, l'arbre $T = ((\circ, a, ((\circ, b, \circ), a, \circ)), f, (\circ, c, \circ))$ est représenté dans la **figure 3**.

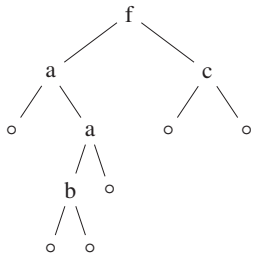

Figure 3 - Représentation de T

Dans la suite, les lettres de Σ sont représentées en Ocaml par des `char` et les mots sur l'alphabet Σ par des `char list`. Ainsi, le mot w = "arbre" est représenté par la liste `w = ['a';'r';'b';'r';'e']`. On représente les arbres binaires en Ocaml à l'aide de la structure `arbre` suivante :

```
type 'a arbre = Vide | Noeud of ('a arbre) * 'a * ('a arbre).
```

Ainsi, l'arbre $T = (\circ, a, (\circ, b, \circ))$ est représenté en Ocaml par :

```
Noeud(Vide, 'a', Noeud(Vide, 'b', Vide)).
```

Q19. Représenter le graphe de l'**ABR** associé au mot "fantastique", l'ordre sur les lettres étant l'ordre alphabétique.

Q20. Écrire une fonction récursive en Ocaml de signature :

```
insertion_lettre : char -> char arbre -> char arbre
```

et telle que `insertion_lettre a t` est l'arbre binaire obtenu en insérant la lettre `a` dans l'arbre binaire `t`.

Q21. Écrire une fonction récursive en Ocaml de signature :

```
insertion_mot : char list -> char arbre -> char arbre
```

et telle que `insertion_mot w t` est l'arbre binaire obtenu en insérant le mot `w` dans l'arbre binaire `t`.

Définition 18 (Lecture préfixe)
Soit T un arbre binaire. La **lecture préfixe** de T que l'on note w_T est le mot défini récursivement par :

- si $T = \circ$, alors $w_T = \varepsilon$;
- si $T = (T_g, r, T_d)$, alors $w_T = r w_g w_d$ où w_g et w_d sont les lectures préfixes respectives de T_g et de T_d.

Q22. Expliciter w_T pour l'arbre binaire T représenté ci-dessous :

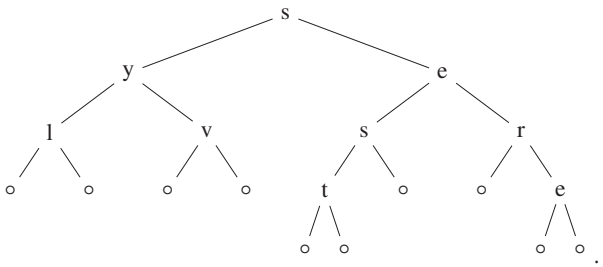

Figure 4 - Représentation de T de **Q22**

Q23. Écrire une fonction récursive en Ocaml de signature :

```
prefixe : char arbre -> char list
```

et telle que `prefixe t` est le mot qui correspond à la lecture préfixe de l'arbre `t`.

Q24. Soit T un **ABR**, soit w_T la lecture préfixe de T. Montrer que T est l'**ABR** associé à w_T.

III.2 - Une relation d'équivalence sur les mots

Définition 19 (S-adjacence)
Soient u et v deux mots sur Σ. On dit que u est **S-adjacent** à v (ou que u et v sont S-adjacents) s'il existe trois lettres $\mathbf{a} < \mathbf{b} \leq \mathbf{c}$ de Σ et trois mots f_1, f_2, f_3 sur Σ vérifiant :

$$(u = f_1\mathbf{b}f_2\mathbf{ac}f_3 \text{ et } v = f_1\mathbf{b}f_2\mathbf{ca}f_3) \text{ ou } (u = f_1\mathbf{b}f_2\mathbf{ca}f_3 \text{ et } v = f_1\mathbf{b}f_2\mathbf{ac}f_3).$$

Définition 20 (S-équivalence)
Soient u et v deux mots sur Σ. On dit que u est **S-équivalent** à v (ou que u et v sont S-équivalents) s'il existe $n \in \mathbb{N}, (w_0, w_1, \ldots, w_n) \in (\Sigma^\star)^{n+1}$ vérifiant :

$$u = w_0, v = w_n, \forall i \in \{0, 1, \ldots, n-1\}, w_i \text{ est S-adjacent à } w_{i+1}\ .$$

Q25. Montrer que la relation " être S-équivalent à " est bien une relation d'équivalence sur $\Sigma^\star$.

Q26. Soient a, b, c trois lettres de Σ vérifiant $a < b \leq c$. Montrer que pour tout arbre binaire de recherche T contenant au moins la lettre b, on a : $T \leftarrow ac = T \leftarrow ca$. On pourra raisonner par récurrence sur le nombre de nœuds de T.

Q27. Montrer que si deux mots u et v sont S-équivalents, alors les **ABR** $(\circ \leftarrow u)$ et $(\circ \leftarrow v)$ sont égaux.

Q28. Soit w un mot de longueur $n \geq 1$ sur Σ, soit r la première lettre de w. On veut montrer qu'il existe un mot $w' = ruv$ où u (respectivement v) est un mot dont toutes les lettres sont plus petites strictement (respectivement plus grandes au sens large) que r et tels que w et w' sont S-équivalents. On note A l'ensemble des mots S-équivalents à w.
Pour tout $a = a_0 a_1 \dots a_{n-1} \in A$, on pose :

$$N(a) = \left\{(i,j) \in [\![1, n-1]\!]^2 \middle| \; i < j, \; a_j < r \leq a_i\right\}.$$

(a) Justifier que l'ensemble A est fini.

(b) Justifier que tous les mots de A commencent par r.

(c) Soit $a \in A$. Montrer que si $N(a) \neq \emptyset$, alors il existe $k \in [\![1, n-2]\!]$ tel que $(k, k+1) \in N(a)$.

(d) Soit $a \in A$. Montrer que si $N(a) = \emptyset$, alors il existe u (respectivement v) un mot dont toutes les lettres sont plus petites strictes (respectivement plus grandes ou égales) que r et tels que $a = ruv$.

(e) Soit a un élément de A tel que le nombre d'éléments de $N(a)$ soit le plus petit possible. Montrer par l'absurde que $N(a) = \emptyset$ et en déduire qu'il existe u, v vérifiant les conditions de la question tels que ruv et w sont S-équivalents.

Q29. Montrer que tout mot w est S-équivalent à w_T, où w_T est la lecture préfixe de $T = (\circ \leftarrow w)$. On pourra raisonner par récurrence sur la longueur du mot w et exploiter les résultats de la **Q28**.

Q30. En déduire que deux mots sont S-équivalents si et seulement si ils sont des éléments d'une même classe sylvestre.

III.3 - Construction de classes sylvestres

Pour construire des classes sylvestres, il est utile d'utiliser le produit de mélange.

Définition 21 (Mélange)
Soient u et v deux mots sur Σ. Le **mélange** de u et v, noté $u \sqcup\!\sqcup v$, est l'ensemble récursivement défini comme suit :

- si $v = \varepsilon$, alors $u \sqcup\!\sqcup v = \{u\}$;
- si $u = \varepsilon$, alors $u \sqcup\!\sqcup v = \{v\}$;
- si $u = au'$ et $v = bv'$ où a et b sont des lettres, u' et v' des mots, alors :

$$u \sqcup\!\sqcup v = \{aw | \; \exists w \in u' \sqcup\!\sqcup v\} \cup \{bw | \; \exists w \in u \sqcup\!\sqcup v'\}.$$

Définition 22 (Mélange de langages)
Soient L et M deux langages. Le **mélange des langages** L et M, noté $L \sqcup\!\sqcup M$, est défini par :

$$L \sqcup\!\sqcup M = \bigcup_{u \in L, v \in M} u \sqcup\!\sqcup v.$$

Si au moins un des deux langages est égal à l'ensemble $\emptyset$ ($\neq \{\varepsilon\}$), alors on a $L \sqcup\!\sqcup M = \emptyset$.

Étant donné un **ABR** $T = (T_g, r, T_d)$, on peut montrer que :

$$\mathrm{Syl}(T) = \left\{rw \middle| \; \exists w \in \mathrm{Syl}(T_g) \sqcup\!\sqcup \mathrm{Syl}(T_d)\right\}.$$

On admet ce résultat pour la suite de cette sous-partie. On note que $\mathrm{Syl}(\circ) = \{\varepsilon\}$.

Q31. Expliciter sans justification tous les éléments de l'ensemble $abba \sqcup\!\sqcup ba$.

Q32. Écrire une fonction récursive en Ocaml de signature :

```
ajout_lettre : char -> char list list -> char list list
```

et telle que `ajout_lettre a liste` est la liste de mots de la forme `a::w` où `w` est un élément de `liste`.

Q33. Écrire une fonction récursive en Ocaml de signature :

```
shuffle : char list-> char list -> char list list
```

et telle que `shuffle u v` est une liste contenant exactement tous les mots de $u \sqcup\!\sqcup v$ avec répétitions possibles d'un même mot. On devra utiliser la fonction auxiliaire `ajout_lettre` et l'opérateur de concaténation `@`.

Q34. Écrire une fonction récursive en Ocaml de signature :

```
shuffle_l : char list list -> char list list -> char list list
```

et telle que `shuffle_l l m` est une liste contenant exactement tous les mots de $L \sqcup\!\sqcup M$ où la liste `l` (respectivement `m`) représente le langage L (respectivement M), avec répétitions possibles d'un même mot. Seules les fonctions définies précédemment peuvent être utilisées en fonctions auxiliaires.

Q35. Écrire une fonction récursive en Ocaml de signature :

```
sylvestre_T : char arbre -> char list list
```

et telle que `sylvestre_T t` est une liste contenant exactement tous les éléments de la classe sylvestre de `t`. Seules les fonctions définies précédemment peuvent être utilisées en fonctions auxiliaires.

FIN