COMPOSITION D'INFORMATIQUE n°3

Sujet 1 : Algorithmique et programmation

(Durée : 4 heures)

L'utilisation de la calculatrice **n'est pas autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Jeu de mots et langages continuables.

Le sujet traite d'un jeu à deux joueurs sur la construction d'un mot autour d'un langage rationnel et contient cinq parties. La première partie présente le jeu et introduit quelques exemples. La deuxième partie aborde le cas d'un langage fini. La troisième partie, indépendante du reste, étudie des langages particuliers, appelés continuables, et leur lien avec les mots primitifs. La quatrième partie reprend le jeu pour un langage non continuable. Enfin, la cinquième et dernière partie concerne une légère variante du jeu, dans le cas d'un langage continuable.

Consignes

Les questions de programmation doivent être traitées en langage C uniquement. On supposera que les bibliothèques `stdlib.h`, `stdbool.h` et `string.h` ont été chargées.

Lorsque le candidat écrira une fonction, il pourra faire appel à des fonctions définies dans les questions précédentes, même si elles n'ont pas été traitées. Il pourra également définir des fonctions auxiliaires, mais devra préciser leurs rôles ainsi que les types et significations de leurs arguments. Les candidats sont encouragés à expliquer les choix d'implémentation de leurs fonctions lorsque ceux-ci ne découlent pas directement des spécifications de l'énoncé. Si les paramètres d'une fonction à coder sont supposés vérifier certaines hypothèses, il ne sera pas utile dans l'écriture de cette fonction de tester si les hypothèses sont bien satisfaites.

On identifiera une même grandeur écrite dans deux polices de caractères différentes, en italique du point de vue mathématique (par exemple n) et en Computer Modern à chasse fixe du point de vue informatique (par exemple `n`).

Sans précision supplémentaire, lorsqu'une question demande la complexité d'une fonction, il s'agira de la complexité temporelle dans le pire des cas. La complexité sera exprimée sous la forme $\mathcal{O}(f(n,m))$ où n et m sont les tailles des arguments de la fonction, et f une expression la plus simple possible. Les calculs de complexité seront justifiés succinctement.

1 Présentation du jeu

On s'intéresse dans ce sujet à un jeu à deux joueurs. Le principe est le suivant : pour Σ un alphabet et $L \subseteq \Sigma^*$ un langage rationnel, en partant d'un mot initialement vide, les joueurs vont tour à tour ajouter une lettre de Σ à la fin du mot. Si le mot obtenu devient un mot de L, le joueur 1 remporte la partie. Si le joueur 2 a la garantie que, même en passant ses tours, le joueur 1 ne pourra pas obtenir un mot de L, c'est lui qui remporte la partie. Si la partie est infinie, elle est considérée comme un match nul.

Formellement, on considère L un langage rationnel donné sous la forme d'un automate fini déterministe **complet** $A = (Q, \delta, q_0, F)$, où Q désigne l'ensemble des **états**, $\delta : Q \times \Sigma \to Q$ est une fonction totale appelée **fonction de transition**, $q_0 \in Q$ est **l'état initial** et $F \subseteq Q$ est **l'ensemble des états finaux**.

Une **partie** est une suite infinie $(a_k)_{k\in\mathbb{N}^*} \in \Sigma^{\mathbb{N}^*}$ de lettres. On définit par récurrence une suite de mots $(u_k)_{k\in\mathbb{N}}$ par :

– $u_0 = \varepsilon$;
– pour $k \in \mathbb{N}^*$, $u_k = u_{k-1}a_k$.

À une partie, on associe également une suite d'états $(q_k)_{k\in\mathbb{N}}$ telle que pour $k \in \mathbb{N}$, $q_{k+1} = \delta(q_k, a_{k+1})$ (et le premier état de la suite est l'état initial q_0).

Question 1 Avec les définitions précédentes, montrer que pour tout $k \in \mathbb{N}$, $q_k = \delta^*(q_0, u_k)$.

Une partie est dite :
– gagnée par 1 s'il existe $k \in \mathbb{N}$ tel que $u_k \in L$, c'est-à-dire tel que $q_k \in F$;
– gagnée par 2 s'il existe $k \in \mathbb{N}$ tel que pour tout $v \in \Sigma^*$, $u_k v \notin L$, c'est-à-dire tel que q_k n'est pas co-accessible ;
– nulle sinon.

Une **stratégie** est une fonction $f : Q \to \Sigma$. Si f est une stratégie pour le joueur $j \in \{1, 2\}$ et g est une stratégie pour le joueur $3-j$, la partie $(a_k)_{k\in\mathbb{N}^*} \in \Sigma^{\mathbb{N}^*}$ jouée selon f et g et commencée par le joueur j, appelée (f, g)-**partie**, est la partie vérifiant :
– si k impair, alors $a_k = f(q_{k-1})$;
– si k pair, alors $a_k = g(q_{k-1})$.

où la suite (q) est celle définie précédemment.

On dit qu'une stratégie f est **gagnante** pour le joueur j si pour toute stratégie g :
– si j commence à jouer, alors la (f, g)-partie est gagnante pour j ;
– si $3 - j$ commence à jouer, alors la (g, f)-partie est gagnante pour j.

Question 2 On suppose pour cette question uniquement que $|\Sigma| = 1$. Quel joueur possède une stratégie gagnante ? On distinguera selon le langage L.

Question 3 On suppose que c'est au joueur 2 de commencer à jouer. Montrer que, quitte à modifier l'automate A, on peut se ramener à l'étude d'un jeu où c'est le joueur 1 qui commence à jouer.

Dans toute la suite du sujet, on suppose que c'est au joueur 1 de commencer à jouer.

Pour les trois questions suivantes, on suppose $\Sigma = \{a, b\}$.

Question 4 Dans cette question, on considère L_1 comme l'ensemble des mots commençant par aa ou terminant par ba. Représenter graphiquement un automate fini déterministe complet reconnaissant L_1, puis montrer qu'il existe une stratégie gagnante pour le joueur 1, qu'on détaillera.

Question 5 Dans cette question, on considère L_2 comme l'ensemble des mots commençant par aab. Représenter graphiquement un automate fini déterministe complet reconnaissant L_2, puis montrer qu'il existe une stratégie gagnante pour le joueur 2, qu'on détaillera.

Question 6 Dans cette question, on considère L_3 comme l'ensemble des mots terminant par aab. Représenter graphiquement un automate fini déterministe complet reconnaissant L_3, puis montrer qu'il n'existe de stratégie gagnante pour aucun des deux joueurs.

2 Jeu dans un langage fini

Question 7 Montrer que si L est un langage fini, l'un des deux joueurs possède une stratégie gagnante.

On représente en C un mot par une chaîne de caractères. Un automate fini déterministe complet $A = (Q, \delta, q_0, F)$ est représenté par un objet `A` du type suivant :

```
struct AFD {
    int Q;
    int Sigma;
    bool* finaux;
    int** delta;
};

typedef struct AFD afd;
```

tel que $Q = [\![0, n-1]\!]$, avec $n =$ `A.Q`, $|\Sigma| = m =$ `A.Sigma`, `A.finaux` est un tableau de booléens de taille n tel que pour $q \in Q$, `A.finaux[q]` vaut `true` si et seulement si $q \in F$ et `A.delta` est un tableau de n tableaux de m entiers tel que pour $q \in Q$ et $a \in \Sigma$, $\delta(q, a) =$ `A.delta[q][phi(a)]`, où `int phi(char c)` est une bijection de Σ dans $[\![0, m-1]\!]$ qu'on suppose créée et qu'on ne demandera pas d'expliciter. On supposera de plus que l'état initial q_0 est l'état 0.

Question 8 Écrire une fonction `int delta_etoile(afd A, int q, char* u)` qui calcule $\delta^*(q, u)$ étant donné un automate fini déterministe complet A.

Question 9 En déduire une fonction `bool reconnu(afd A, char* u)` qui détermine si le mot u est reconnu par l'automate A.

Pour les questions suivantes dans cette partie, on suppose que le langage L reconnu par l'automate A est un langage fini. On suppose de plus que l'automate A est de taille minimale.

Question 10 Montrer que tous les états de A sont accessibles et que A possède un seul état non co-accessible qu'on notera q_ω.

Question 11 Soit $q \in Q$. Montrer qu'il existe $u \in \Sigma^+$ tel que $\delta^*(q, u) = q$ si et seulement si $q = q_\omega$.

Pour $q \in Q$ on appelle **issue de** q, notée $\iota(q)$, un entier $i \in \{0, 1, 2, 3\}$ tel que :
- si $\iota(q) = 0$, c'est le joueur dont ce n'est pas le tour de jouer depuis q qui a une stratégie gagnante ;
- si $\iota(q) = 1$, c'est le joueur 1 qui a une stratégie gagnante depuis q, indépendamment du joueur dont c'est le tour ;
- si $\iota(q) = 2$, c'est le joueur 2 qui a une stratégie gagnante depuis q, indépendamment du joueur dont c'est le tour ;
- si $\iota(q) = 3$, c'est le joueur dont c'est le tour de jouer depuis q qui a une stratégie gagnante.

Pour un automate `A` de type `afd`, le tableau des issues est un tableau `issue` de taille $n =$ `A.Q` tel que pour $q \in Q$, `issue[q]` vaut $\iota(q)$.

Question 12 Représenter graphiquement, sans justifier, un automate fini déterministe complet minimal à 4 états reconnaissant un langage fini, dont le tableau des issues est `{0, 1, 2, 3}`.

Pour $q \in Q$, on appelle **voisin de** q un état $q' \in \delta(q, \Sigma)$.

Question 13 Soit $q \in Q$ un état non final différent de q_ω. Montrer, en justifiant brièvement, les propriétés suivantes :
(a) $\iota(q) = 0$ si et seulement si q ne possède que des voisins d'issue 3 ;
(b) $\iota(q) = 1$ si et seulement si q possède au moins un voisin d'issue 1 et aucun voisin d'issue 0 ou 2 ;
(c) $\iota(q) = 2$ si et seulement si q possède au moins un voisin d'issue 2 et aucun voisin d'issue 0 ou 1 ;
(d) $\iota(q) = 3$ si et seulement si q possède soit au moins un voisin d'issue 0, soit au moins un voisin d'issue 1 et un d'issue 2.

Question 14 Écrire une fonction `int* calcul_issues(afd A)` qui prend en argument un automate A et

calcule et renvoie le tableau `issue` décrit précédemment. Cette fonction devra avoir une complexité temporelle en $\mathcal{O}(|Q||\Sigma|)$ et on demande de justifier cette complexité.

Indication : on pourra écrire une fonction auxiliaire récursive `int issue_rec(afd A, int q, int* issue)` *qui modifie la valeur de* `issue` *pour les états accessibles depuis q et renvoie l'issue de l'état q.*

Question 15 En déduire une fonction `char strategie_optimale(afd A, int q, int j, int* issue)` qui, étant donné l'automate A, un état $q \in Q$, un joueur $j \in \{1, 2\}$ et le tableau `issue` calculé par la fonction précédente, renvoie une lettre. La stratégie optimale devra être gagnante pour j s'il existe une stratégie gagnante pour j. On supposera disposer d'une fonction `char psi(int i)` qui est la bijection réciproque de la fonction `phi`.

3 Langages continuables et mots primitifs

Dans toute cette partie, on suppose que $|\Sigma| > 1$.

On dit qu'un langage rationnel $L \subseteq \Sigma^*$ est **continuable** si et seulement si

$$\forall u \in \Sigma^*, \exists v \in \Sigma^*, uv \in L$$

Question 16 Donner un exemple de langage continuable. Donner un exemple de langage infini non continuable.

Question 17 Existe-t-il un langage continuable dont le complémentaire est infini ? Justifier.

Question 18 Montrer que L est continuable si et seulement si L est reconnaissable par un automate fini déterministe complet et émondé.

Soit $u \in \Sigma^* \setminus \{\varepsilon\}$. u est dit **primitif** s'il n'existe pas de mot $v \in \Sigma^*$ et d'entier $p > 1$ tel que $u = v^p$.

Question 19 Le mot *abaaabaa* est-il primitif ? Le mot *ababbaabbbababbab* (de longueur 17) est-il primitif ?

Question 20 Écrire une fonction `bool primitif(char* u)` qui détermine si un mot u est primitif ou non. La fonction devra être de complexité temporelle quadratique en $|u|$ et de complexité spatiale constante.

Question 21 Donner un exemple de langage rationnel infini qui ne contient aucun mot primitif. Donner un exemple de langage rationnel infini qui ne contient que des mots primitifs.

Question 22 Montrer qu'un langage continuable contient une infinité de mots primitifs.

Indication : on pourra s'intéresser à la manière de continuer les mots de la forme ab^i.

Question 23 La réciproque de la question précédente est-elle vraie ? Justifier.

4 Jeu dans un langage non continuable

4.1 Manipulation de listes

On s'intéresse à un type de listes chaînées donné par :

```c
struct Liste {
    int x;
    struct Liste* suivant;
};

typedef struct Liste liste;
```

Pour la suite, lorsqu'on parlera d'une liste, on fera référence à un objet de type `liste*`.

Question 24 Écrire une fonction `liste* cons(int x, liste* lst)` qui renvoie une nouvelle liste ayant `x` en tête et `lst` en queue.

Question 25 Écrire une fonction `bool mem(int x, liste* lst)` qui détermine si l'entier `x` appartient à la liste `lst`.

4.2 Graphes d'automates

Pour un automate fini déterministe complet reconnaissant un langage non continuable, il existe, d'après la partie précédente, des états non co-accessibles. Ainsi, le jeu est symétrique pour les deux joueurs et se ramène à des questions d'accessibilité dans un graphe orienté (seule l'existence des transitions nous intéresse, pas les lettres qui étiquètent ces transitions).

Pour $A = (Q, \delta, q_0, F)$, on appelle **graphe d'automate de** A, noté G_A, le graphe orienté $G_A = (Q, T)$ où $T \subseteq Q^2$ est l'ensemble des arêtes (transitions) défini par :

$$(p, q) \in T \Leftrightarrow \exists a \in \Sigma, \delta(p, a) = q$$

Le jeu se ramène alors à créer un chemin dans G_A depuis l'état initial q_0, en rajoutant une nouvelle arête à chaque tour d'un joueur.

On représente un graphe orienté en C par un objet du type `graphe` défini par :

```c
struct Graphe {
    int Q;
    liste** T;
};

typedef struct Graphe graphe;
```

tel que si $G = (Q, T)$ est représenté par un objet `G` de type `graphe`, alors $Q = [\![0, n-1]\!]$ avec $n =$ `G.Q`, et `G.T` est un tableau de taille n tel que pour $q \in Q$, `G.T[q]` est une liste chaînée contenant les voisins de q, dans un ordre quelconque.

Question 26 Écrire une fonction `graphe graphe_automate(afd A)` qui crée et renvoie le graphe d'automate G_A associé à un automate A. On n'impose pas de complexité pour cette fonction.

Question 27 Écrire une fonction `void liberer_graphe(graphe G)` qui libère la mémoire allouée pour un graphe G.

Soit $G = (Q, T)$ un graphe orienté, $X \subseteq Q$ et $j \in \{1, 2\}$. Pour $q \in Q$, on note $V(q)$ l'ensemble des voisins de q. On définit par récurrence les ensembles $\mathrm{Attr}_i(X, j)$ par :

- $\mathrm{Attr}_0(X, j) = \{(q, j) \mid q \in X\} \cup \{(q, 3-j) \mid q \in X\}$;

– pour $i \in \mathbb{N}$,

$$\begin{aligned}\mathrm{Attr}_{i+1}(X,j) = \quad & \mathrm{Attr}_i(X,j) \\ \cup \quad & \{(q,j) \mid q \in Q \text{ et } \exists q' \in V(q), (q', 3-j) \in \mathrm{Attr}_i(X,j)\} \\ \cup \quad & \{(q,3-j) \mid q \in Q, V(q) \neq \emptyset \text{ et } \forall q' \in V(q), (q', j) \in \mathrm{Attr}_i(X,j)\}\end{aligned}$$

Question 28 Décrire en français à quoi correspond l'ensemble $\mathrm{Attr}_i(X,j)$, pour $i \in \mathbb{N}$, $X \subseteq Q$ et $j \in \{1,2\}$. On justifiera succinctement la description.

Question 29 Montrer que pour $X \subseteq Q$ et $j \in \{1,2\}$, la suite $(\mathrm{Attr}_i(X,j))_{i\in\mathbb{N}}$ converge vers un ensemble qu'on notera $\mathrm{Attr}(X,j)$.

Question 30 Montrer que pour un automate fini déterministe A, le joueur 1 a une stratégie gagnante si et seulement si $(q_0, 1) \in \mathrm{Attr}(F, 1)$. Donner une condition nécessaire et suffisante pour que le joueur 2 ait une stratégie gagnante.

5 Jeu dans un langage continuable

Si le langage d'un automate fini déterministe complet A est continuable, alors A ne possède pas d'état qui soit accessible sans être co-accessible. Ainsi, le joueur 2 ne pourrait pas gagner de partie. On propose pour cette partie une légère modification de la règle du jeu, à savoir que pour gagner, le joueur 2 doit empêcher la victoire du joueur 1. Ainsi, les parties infinies qui étaient considérées comme des matchs nuls précédemment sont maintenant des victoires pour le joueur 2.

Question 31 Comment déterminer qui possède une stratégie gagnante pour cette variante ? Justifier.

Question 32 Écrire une fonction `graphe transpose(graphe G)` qui renvoie le graphe transposé d'un graphe G donné.

Question 33 Écrire une fonction `int gagnant_continuable(afd A)` qui renvoie un entier valant 1 ou 2 selon qu'il existe une stratégie gagnante pour le joueur 1 ou le joueur 2, pour le jeu dans un automate A donné reconnaissant un langage continuable.

⋆ ⋆ ⋆