DS8 -Informatique

Calculatrices interdites. Aucun document autorisé.

Si un élève est amené à repérer ce qui lui semble 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. Les réponses comportant du code doivent être accompagnées de commentaires et d'explications et le code doit être écrit avec le plus grand soin.

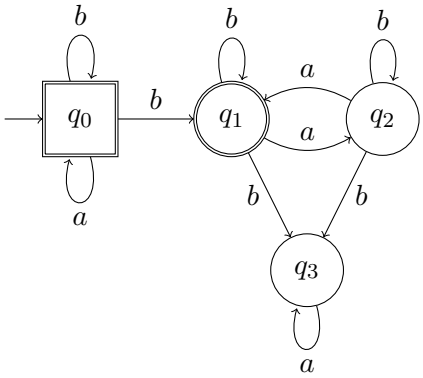
Automates alternants

Dans ce problème, on s'intéresse aux automates alternants, une extension des automates finis vus dans le programme.

Définition 1. Un **automate fini alternant** est représenté par un sextuplé $(\Sigma, Q_\exists, Q_\forall, q_0, F, \delta)$:

- $Q = Q_\exists \cup Q_\forall$ est un ensemble d'états partitionné en deux ensembles $Q_\exists$ d'états "existentiels" et $Q_\forall$ d'états "universels" disjoints ;
- Σ est l'alphabet utilisé ;
- $q_0 \in Q$ est l'état initial ;
- $F \subset Q$ est un ensemble d'états finaux ;
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ est une fonction de transition.

Voici un exemple d'automate alternant $(\{a, b\}, \{q_1, q_2, q_3\}, \{q_0\}, q_0, \{q_0, q_1\}, \delta)$ que l'on appellera $\mathcal{A}_1$:



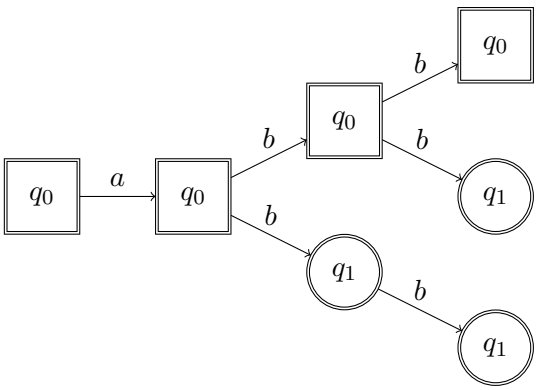
On représente les états existentiels par des rond et les états universels par des carrés.

Définition 2. Un **calcul** d'un mot $w = w_0....w_n \in \Sigma^*$ dans $A = (\Sigma, Q_\exists, Q_\forall, q_0, F, \delta)$ est un arbre T étiqueté par des états dans Q tel que :

- la racine de T est étiquetée par q_0 l'état initial ;
- si un noeud intérieur (racine comprise) de profondeur i est étiqueté par un noeud $q \in Q_\exists$, alors il a exactement un fils d'étiquette p tel que $p \in \delta(q, w_i)$;
- si un noeud intérieur (racine comprise) de profondeur i est étiqueté par un noeud $q \in Q_\forall$ tel que $\delta(q, w_i) = \{p_1, ..., p_k\}$, alors il a exactement k fils étiqueté par les p_j ;
- les feuilles sont toutes de profondeur n.

On rappelle que la profondeur d'un noeud est le nombre d'arête le séparant de la racine.
Un calcul est dit **acceptant** lorsque toutes les feuilles sont étiquetées par des états finaux.

Par exemple, voici un calcul acceptant pour *abb* dans l'automate exemple $\mathcal{A}_1$:



On pourra aussi parler de calculs partiels et les représenter par des arbres intuitivement lorsqu'il n'existe pas de transition depuis un état pour la lettre attendu dans l'arbre. En particulier, ces calculs ne sont pas acceptants.

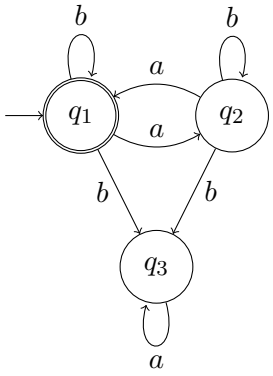
Le problème est traité en 5 parties qui ne sont pas indépendantes : la première partie comporte des questions de programmation en C. La sixième partie du sujet est un exercice sur les bases de données complètement indépendant. La septième partie est un exercice de logique court et indépendant.

1 Construction de calculs acceptants

Question 1. Donner pour l'automate exemple $\mathcal{A}_1$ un arbre pour un calcul acceptant de *abbaabaa*.

Question 2. On note L_i le langage reconnu depuis l'état i dans l'automate exemple $\mathcal{A}_1$ pour $i = 0, 1, 2, 3$.
Donner des équations liant les 4 langages. Justifier.

Question 3. Donner le langage reconnu par l'automate fini suivant.
On donnera une expression régulière et une interprétation informelle.



Question 4.

(*a*) Montrer que le langage reconnu par l'automate alternant $\mathcal{A}_1$ est $\{m = m_1...m_{|m|} | \forall i \in [\![1, |m|]\!], m_i = b \Rightarrow |m_{i+1}...m_{|m|}|_a \equiv 0[2]\}$.

(*b*) Donner une expression régulière (simple) dénotant ce langage. Justifier

Question 5. Proposer en pseudo-code un algorithme permettant de déterminer un calcul acceptant pour un mot donné et un automate alternant donné.

On se donne les types suivants en C pour les automates finis alternants et les arbres de calcul.

Soit `a` une variable de type `afa_t` représentant un automate fini alternant. Les états sont représentées par des entiers de `0` à `nmb-1` :

- l'état initial est toujours l'état `0` ;
- l'état `i` est final lorsque `final[i]` vaut `true` ;
- l'état `i` est universel lorsque `forall[i]` vaut `true`, sinon il est existentiel00.

D'autre part, pour un état `i`, `delta[i]` est :

- `NULL` si `i` est un état tel que $\delta(q_i, a) = \emptyset$ pour tout $a \in \Sigma$;
- une liste chaînée de transitions représentées par l'état d'arrivée et la lettre de l'étiquette.

```
struct liste_transitions {       // Liste de de transitions
    int etat ;
    char lettre ;
    struct liste_transitions * suite ;
};
typedef struct liste_transitions liste_transitions_t ;

struct afa {                     //Automate Fini Alternant
    int nmb ;                    // Nombre d'états
    bool* final ;
    bool* forall ;
    liste_transitions_t** delta ;  // Fonction de transition
};
typedef struct afa afa_t ;

struct arbre_calcul {            // Arbre de calcul
    char lettre ;                // Lettre de la première transition
    int etat ;                   // Etiquette
    int nmb_fils ;
    struct arbre_calcul * fils ;
};
typedef struct arbre_calcul arbre_calcul_t ;

struct mot {
    int longueur;
    char* lettres;
};
typedef struct mot mot_t ;
```

Question 6. Écrire une fonction `int nmb_trans(liste_transitions_t* l, char c)` qui donne le nombre de transitions d'une liste pointée faisant intervenir une lettre donnée.
Préciser la complexité.

Question 7. Écrire une fonction `mot_t mot_lu (arbre_calcul_t a)` qui renvoie le mot lu dans l'arbre de calcul donné en argument.
Préciser la complexité.
Dans cette question, on ne demande pas de vérifier que l'arbre est un arbre de calcul valide.

Question 8. Écrire une fonction `arbre_calcul_t arbre_init (mot_t m)` qui initialise un arbre de calcul partiel réduit à la racine sans fils pour un mot m non vide.

Question 9. Écrire une fonction `bool check(arbre_calcul_t a)` qui vérifie qu'un arbre est bien arbre de calcul complet (branches correspondant toutes au même mot, racine étiquetée par l'état initial).
Préciser la complexité.
On pourra décomposer la réponse en plusieurs fonctions.

Question 10. Écrire une fonction `bool valide(arbre_calcul_t arbre, afa_t a)` qui vérifie qu'un arbre de calcul complet est valide vis-à-vis d'un automate alternant donné.
Préciser la complexité.
On vérifie uniquement que l'arbre de calcul respecte les états universels et existentiels, ainsi que les transitions de l'automate.

Question 11. Écrire une fonction `bool accept(arbre_calcul_t arbre, afa_t a)` qui vérifie qu'un arbre de calcul complet valide est acceptant.
Préciser la complexité.

Question 12. Écrire une fonction `bool accept_mot(mot_t m, afa_t a)` qui vérifie qu'un mot admet un arbre de calcul acceptant pour l'automate alternant.
On pourra écrire une fonction auxiliaire :
`bool accept_mot_aux(mot_t m, afa_t a, int curseur, int etat)`
récursive qui vérifie si la fin d'un mot à partir d'une lettre $w_{\texttt{curseur}}$ est acceptée depuis un état donné.
Préciser la complexité.

Question 13. Écrire une fonction `arbre_calcul_t construit(mot_t m, afa_t a)` qui renvoie un arbre de calcul acceptant valide pour le mot et l'automate alternant donné.
On pourra décomposer la réponse en plusieurs fonctions.
Préciser la complexité.

2 Jeu associé à un automate alternant

A tout automate alternant $\mathcal{A} = (\Sigma, Q_\exists, Q_\forall, q_0, F, \delta)$ et tout mot $w = w_0...w_{n-1} \in \Sigma^*$, on peut associer un graphe orienté $G(\mathcal{A}, w) = (S_{\mathcal{A},w}, A_{\mathcal{A},w})$ avec $S_{\mathcal{A},w} = Q \times \{0, 1, 2, ..., n\}$ et $A_{\mathcal{A},w} = \{((q, i), (p, i + 1))|i \in \{0, .., n - 1\}, p \in \delta(q, w_i)\}$.

Question 14. Dessiner le graphe correspondant à l'automate $\mathcal{A}_1$ pour le mot $w = abbaa$.

Question 15. Justifier que le graphe obtenu est acyclique et biparti.

On définit le jeu suivant : deux joueurs appelés *Automaton* et *Pathfinder* s'affrontent sur le graphe $G(\mathcal{A}, w)$. *Automaton* contrôle les sommets (q, i) avec $q \in Q_\exists$ et $i \in \{0, ..., n\}$ et *Pathfinder* contrôle les autres sommets. On place un jeton sur le sommet $(q_0, 0)$.
A chaque tour, le joueur qui contrôle le sommet choisit un voisin directement accessible (en respectant l'orientation des arêtes) depuis ce sommet et déplace le jeton vers ce sommet.
Le jeu s'arrête lorsque le sommet ne dispose plus d'arêtes sortantes ;
Automaton gagne si le sommet où le jeton se trouve en fin de partie est dans $F \times \{n\}$. Sinon, *Pathfinder* gagne.

Question 16. Décrire une partie gagnante pour chaque joueur sur le graphe $G(\mathcal{A}_1, abbaa)$.

Question 17. Quelle est la durée maximum d'une partie ? Existe-t-il des parties plus courtes ?

Une **partie partielle** sur $G(\mathcal{A}, w)$ est représentée par une suite finie de sommets $(s_0, s_1, ..., s_k)$ de sommets du graphe tel que pour $0 \leq i \leq k, (s_i, s_{i+1}) \in A_{\mathcal{A},w}$ et $s_0 = (q_0, 0)$ et s_k admet au moins une arrête sortante.
On note $S^\omega_{\mathcal{A},w}$ l'ensemble de parties partielles sur $G(\mathcal{A}, w)$.
Une **stratégie** est une fonctions $f : S^\omega_{\mathcal{A},w} \to S_{\mathcal{A},w}$ qui à une partie partielle $(s_0, s_1, ..., s_k)$ associe $f((s_0, s_1, ..., s_k)) = s_{k+1}$ tel que $(s_k, s_{k+1}) \in A_{\mathcal{A},w}$.
On dit qu'une stratégie est **gagnante** pour un joueur si pour toute partie complète dans laquelle le joueur suit la stratégie, il gagne.

*Attention, on considère des stratégies depuis $(q_0, 0)$ et qui **ne** sont **pas** sans mémoire ici.*

Question 18. Montrer que *Automaton* dispose d'une stratégie gagnante s'il existe un calcul acceptant pour w sur l'automate alternant $\mathcal{A}$.

Question 19. Montrer que si *Automaton* dispose d'une stratégie gagnante, alors il existe un calcul acceptant pour w sur l'automate alternant $\mathcal{A}$.

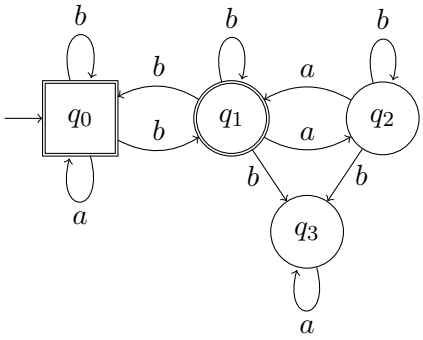
3 Stratégie positionelle

On dit qu'une stratégie dans un graphe $G_{\mathcal{A},w}$ est **positionelle** lorsque l'image ne dépend que de l'état sur l'étiquette du dernier sommet de la partie partielle.

Formellement $f : S^{\omega}_{\mathcal{A},w} \to S_{\mathcal{A},w}$ est positionelle s'il existe $\phi : Q \times \Sigma \to Q$ tel que pour toute partie partielle : $f((s_0, s_1, ..., (q_k, k))) = \phi(q_k, w_k)$.
Autrement dit, pour une lettre donnée, depuis un état, on choisit toujours d'utiliser la même transition.

Question 20. Décrire une stratégie positionelle gagnante sur $G(\mathcal{A}_1, abbaa)$ pour *Automaton*.

Question 21. Considérons l'automate alternant $\mathcal{A}_2$ suivant :



(*a*) Représenter $G(\mathcal{A}_2, abbbaa)$.
(*b*) Représenter un arbre de calcul acceptant pour *abbbaa* dans $\mathcal{A}_2$. Est-il unique ?
(*c*) Décrire une stratégie gagnante non positionelle pour *Automaton*.
(*d*) Décrire une stratégie gagnante positionelle pour *Automaton*.
(*e*) *Pathfinder* dispose-t-il d'une stratégie gagnante ?

On définit un calcul positionel sur un automate alternant de même. Pour chaque noeud de même état étiquette de l'arbre et correspondant à la même lettre, on exploite la même transition.

Question 22. Montrer que *Automaton* dispose d'une stratégie positionelle gagnante dans $G(\mathcal{A}, w)$ si et seulement s'il existe un calcul positionel pour w dans $\mathcal{A}$.

4 Existence d'une stratégie positionnelle gagnante

On considère $\mathcal{A}$ un automate alternant et $w = w_0...w_{n-1}$ un mot.
Le but de cette section est de montrer que sur le graphe $G(\mathcal{A}, w)$, soit *Automaton* dispose d'une stratégie **positionelle** gagnante, soit *Pathfinder* dispose d'une stratégie gagnante.

On introduit pour cela la notion de stratégie depuis un sommet s du graphe, on considère les parties partielle qui commence en s.

Question 23. Justifier que depuis tout sommet $s = (q, n)$ avec $q \in Q$, soit *Automaton* dispose d'une stratégie **positionelle** gagnante, soit *Pathfinder* dispose d'une stratégie gagnante.

Question 24. Soit $0 \leq i < n$. On suppose que pour tout $s = (q, j)$ avec $j > i$ et $q \in Q$, soit *Automaton* dispose d'une stratégie **positionelle** gagnante, soit *Pathfinder* dispose d'une stratégie gagnante.

(*a*) Montrer que l'on peut construire une stratégie positionelle gagnante depuis tout (q, i) avec $q \in Q_{\exists}$ pour l'un des deux joueurs.
(*b*) Montrer que l'on peut construire une stratégie positionelle gagnante depuis tout (q, i) avec $q \in Q_{\forall}$ pour l'un des deux joueurs.
(*c*) Conclure

5 Langage reconnu par un automate alternant

Question 25. Soit $\mathcal{A} = (\Sigma, Q_\exists, Q_\forall, q_0, F, \delta)$ un automate alternant et $L(\mathcal{A})$ le langage des mots acceptés. Montrer que $\mathcal{A} = (\Sigma, Q_\forall, Q_\exists, q_0, Q \setminus F, \delta)$ reconnaît $\Sigma^* \setminus L(\mathcal{A})$ le complémentaire de $L(\mathcal{A})$.
On s'appuiera sur les parties précédentes.

Question 26. Soit $\mathcal{A} = (\Sigma, Q_\exists, Q_\forall, q_0, F, \delta)$ un automate alternant.
Montrer que l'on peut construire un automate fini non déterministe qui reconnaît le même langage que $\mathcal{A}$.

6 Un exercice sur les bases de données

On s'intéresse à la base de données concernant un festival de musique. On donne le schéma suivant :

```
Artiste(nom : texte,
        id_artiste : entier)
Scene(numero : entier,
      capacite : entier)
Joue(id_artiste : entier,
     numero_scene : entier,
     date : entier,
     heure : entier,
     duree : entier)
Spectateur(nom : texte,
           prenom : texte,
           date : entier)
```

Les dates sont représentées sous la forme de l'entier $AAAAMMJJ$.
Par exemple, le 1er février 1970 est représenté par 19700201 et le 10 mars 1970 par 19700310 qui est plus grand. On remarquera qu'on peut utiliser les inégalités classiques pour comparer les dates ainsi représentées.
Les heures sont représentées sous la forme de l'entier $HHMM$.
Par exemple, 11h50 est représenté par 1150.
Les durées sont exprimées en minutes.

Question 27. Proposer une clé primaire raisonnable pour la table Spectateur. Justifier.

Question 28. Écrire des requêtes SQL répondant aux questions suivantes :

(*a*) Quelle est la capacité totale d'accueil du festival pour les scènes ?

(*b*) Quel est le nombre de spectateur par jour ?

(*c*) Quels sont les jours où la capacité d'accueil est plus petite que le nombre de spectateurs potentiellement présents ?

(*d*) Quels sont les couples d'artistes qui jouent simultanément ?

7 Un exercice de logique

Question 29. En logique minimale, donner une dérivation du séquent suivant :
$(p \wedge q) \to r, \neg r \vdash p \to \neg q$