

On trouvera sur le dépôt un fichier `TP5.c` à modifier pendant le TP.

### Exercice 1

On représente en C un automate fini déterministe par le type suivant :

```
struct AFD {
    int Q;
    int Sigma;
    int q0;
    bool* finaux;
    int** delta;
};

typedef struct AFD afd;
```

Si `A` est un automate représentant  $A = (Q, \Sigma, \delta, q_0, F)$ , alors :

- `A.Q` représente  $|Q|$  et `A.Sigma` représente  $|\Sigma|$  ;
- `Q` est représenté par  $\llbracket 0, |Q| - 1 \rrbracket$  et  $\Sigma$  est représenté par  $\llbracket 0, |\Sigma| - 1 \rrbracket$  ;
- `A.q0` représente l'état initial  $q_0$  ;
- `A.finaux` est un tableau de booléens de taille  $|Q|$  tel que  $q \in F \Leftrightarrow A.finaux[q]$  prend la valeur `true` ;
- `A.delta` représente un tableau de tableaux d'entiers tel que si  $q \in Q$  et  $a \in \Sigma$ , alors `A.delta[q][a]` vaut  $\delta(q, a)$  s'il existe, ou  $-1$  sinon.

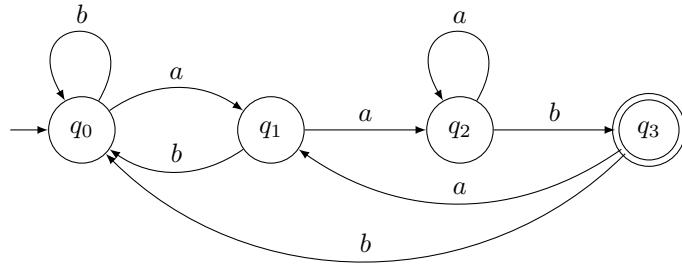
On supposera pour l'ensemble du sujet que les mots sont des chaînes de caractères contenant les 26 lettres de l'alphabet en minuscule. On rappelle que si `a` est un caractère, alors `(int) a - 97` est un entier compris entre 0 et 25.

1. Écrire une fonction `void liberer_afd(afd A)` qui libère la mémoire occupée par un AFD `A`. On pensera à libérer la mémoire occupée par **tous** les tableaux.
2. Écrire une fonction `afd init_afd(int Q, int Sigma, int q0)` qui crée un AFD sans transition et aucun état final dont la taille de  $Q$ , la taille de  $\Sigma$  et l'état initial sont donnés en arguments.
3. Écrire une fonction `void ajout_transition_afd(afd A, int q, char a, int p)` qui modifie l'automate `A` en ajoutant une transition  $\delta(q, a) = p$ . On pensera à transformer le caractère `a` (dont la valeur entière est comprise entre 97 et 122) en un entier de  $\llbracket 0, |\Sigma| - 1 \rrbracket$ .
4. Décommenter les lignes commentées du `main` pour créer un automate `A1` correspondant à l'automate de la figure 1.

En utilisant les fonctions précédentes, créer dans le `main` un AFD `A2` reconnaissant les mots commençant par `a` dont la taille est un multiple de 3, sur l'alphabet  $\{a, b\}$ .

5. Écrire une fonction `int delta_etoile_afd(afd A, int q, char* u)` qui prend en arguments un automate  $A$ , un état  $q$  et un mot  $u$  et renvoie  $\delta^*(q, u)$ . La fonction renverra  $-1$  si cet état n'est pas défini.
6. En déduire une fonction `bool reconnu_afd(afd A, char* u)` qui détermine si un mot  $u$  est reconnu par un automate  $A$ .
7. Tester la fonction précédente sur les deux automates précédents :
  - vérifier que  $u = abbabbabaab \in L(A_1)$  et  $v = baababbbba \notin L(A_1)$  ;
  - vérifier que  $u \notin L(A_2)$ ,  $v \notin L(A_2)$  et  $w = aaabababb \in L(A_2)$ .

## PALIER 1

FIGURE 1 – L’automate  $A_1$ **Exercice 2**

Pour cet exercice et le suivant, on utilisera les fichiers `dicts.c` et `dicts.h` (à placer dans le même répertoire que `TP5.c`).

On s’intéresse dans cet exercice à des automates non déterministes. On les définit en utilisant les types suivant :

```

struct Liste {
    int val;
    struct Liste* suivant;
};

typedef struct Liste liste;

struct AFND {
    int Q;
    int Sigma;
    bool* initiaux;
    bool* finaux;
    liste*** Delta;
};

typedef struct AFND afnd;
  
```

Les différences avec le type d’AFD utilisé précédemment sont les suivantes : si  $B$  est un AFND  $B = (Q, \Sigma, \Delta, I, F)$ , alors :

- $B.\text{initiaux}$  est un tableau de booléens de taille  $|Q|$  tel que  $q \in I \Leftrightarrow B.\text{initiaux}[q]$  prend la valeur `true`;
- $B.\Delta$  représente un tableau de tableaux de listes tel que si  $q \in Q$  et  $a \in \Sigma$ , alors  $B.\Delta[q][a]$  est une liste chaînée contenant les états de  $\Delta(q, a)$ .

Attention, le type `liste` est déjà défini dans le fichier `dicts.h`, et on trouve dans `dicts.c` les fonctions `liste* cons(int x, liste* lst)` qui crée une nouvelle liste étant donnée une tête et une queue et `void liberer_liste(liste* lst)` qui libère l’espace mémoire occupé par une liste.

On se donne, comme pour les AFD, des fonctions de libération de la mémoire `liberer_afnd` et d’initialisation d’automate sans transition, sans état initial et sans état final `init_afnd`.

Avant de tester le code, on décommentera la ligne 7 (`#include "dicts.h"`) et les lignes à partir de la définition du type `AFND` et avant la fonction `main`. On rappelle que la compilation peut se faire avec la commande `gcc dicts.c TP5.c -lm` (avec des options de compilation éventuelles). Le `-lm` est nécessaire pour utiliser la bibliothèque `math.h`.

1. Écrire une fonction `void ajout_transition_afnd(afnd B, int q, char a, int p)` qui modifie l'automate  $B$  et rajoute une transition  $q \xrightarrow{a} p$ . On supposera que cette transition n'existe pas déjà.
2. Décommenter les lignes commentées du `main` pour créer un automate  $B_1$  correspondant à l'automate de la figure 2.

En utilisant les fonctions précédentes, créer dans le `main` un AFND  $B_2$  reconnaissant les mots contenant un  $a$  en avant-dernière position, sur l'alphabet  $\{a, b\}$ .

3. Écrire une fonction `bool* Delta_etats(afnd B, bool* X, char a)` qui prend en argument un AFND  $B$ , un tableau de booléens  $X$  représentant un ensemble d'états  $X \subseteq Q$  et une lettre  $a$  et renvoie un tableau de booléens représentant  $\Delta(X, a)$ . On fera attention à ne pas modifier le tableau  $X$  donné en entrée.
4. En déduire une fonction `bool* Delta_etoile_afnd(afnd B, bool* X, char* u)` qui prend en argument un AFND  $B$ , un tableau de booléens  $X$  représentant un ensemble d'états  $X \subseteq Q$  et un mot  $u$  et renvoie un tableau de booléens représentant  $\Delta^*(X, u)$ . On fera attention à ne pas modifier le tableau  $X$  donné en entrée.
5. En déduire une fonction `bool reconnu_afnd(afnd B, char* u)` qui détermine si un mot  $u$  est reconnu par un AFND  $B$ .
6. Tester la fonction précédente sur les automates  $B_1$  et  $B_2$  :
  - vérifier que  $u \in L(B_1), v \in L(B_1)$  et  $w \notin L(B_1)$ ;
  - vérifier que  $u \in L(B_2), v \notin L(B_2)$  et  $w \in L(B_2)$ .

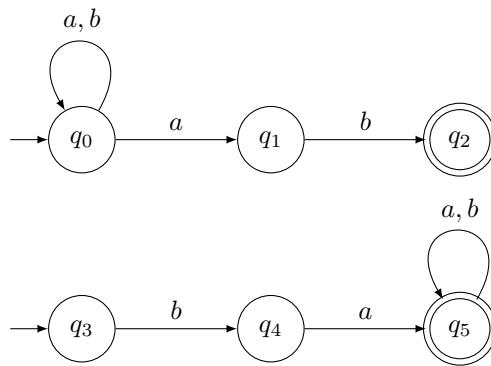


FIGURE 2 – L'automate  $B_1$

## PALIER 2

### Exercice 3

On cherche à écrire une fonction de déterminisation d'automate non déterministe. Pour ce faire, on choisit d'assimiler une partie  $X \subseteq Q$  à un entier compris entre 0 et  $2^{|Q|} - 1$ . On utilisera cette numérotation des états de l'automate des parties.

1. Écrire une fonction `int etats_vers_entier(bool* X, int Q)` qui prend en argument un tableau de booléens représentant une partie  $X \subseteq Q$  et un entier correspondant à  $|Q|$  et renvoie un entier  $x$  de  $[0, 2^{|Q|} - 1]$  associé à  $X$ .
2. Écrire sa fonction réciproque `bool* entier_vers_etats(int x, int Q)`.
3. En déduire une fonction `afnd determiniser(afnd B)` qui détermine un automate non déterministe selon la construction de l'automate des parties.
4. Déterminiser les automates  $B_1$  et  $B_2$  en vérifiant que les mots reconnus par les résultats sont bien cohérents.

La construction naïve de l'automate des parties crée systématiquement un automate déterministe de

taille  $2^{|Q|}$ . On a cependant pu constater lors d'un calcul à la main qu'on peut se contenter de ne garder que les états accessibles depuis l'état initial. On cherche à écrire une nouvelle version de la fonction précédente en suivant ce principe. À cet effet, on se donne une structure de dictionnaire `dict` implémentée par une table de hachage. On dispose des primitives suivantes :

- `void dict_free(dict D)` libère la mémoire occupée par un dictionnaire ;
- `dict create(void)` crée un dictionnaire vide ;
- `int size(dict D)` renvoie la taille d'un dictionnaire ;
- `bool member(dict D, int k)` teste l'appartenance d'une clé à un dictionnaire ;
- `int get(dict D, int k)` renvoie la valeur associée à une clé dans un dictionnaire ;
- `void add(dict* D, int k, int v)` ajoute une association (clé, valeur) à un dictionnaire ;
- `void del(dict* D, int k)` supprime une association d'un dictionnaire ;
- `liste* key_list(dict D)` renvoie une liste chaînée contenant toutes les clés d'un dictionnaire.

On supposera que toute les opérations sauf la première et la dernière sont en complexité  $\mathcal{O}(1)$  en moyenne. Attention, certaines fonctions (celles qui modifient le dictionnaire) prennent en argument un pointeur de dictionnaire `dict*` et pas un dictionnaire `dict`.

5. Écrire une fonction `dict accessibles(afnd B)` renvoyant un dictionnaire dont les clés sont les numéros des parties accessibles de l'automate des parties d'un AFND  $B$  et les valeurs sont des numéros consécutifs partant de 0 permettant de renuméroter ces états.
6. En déduire une fonction `afd determiniser2(afnd B)` qui déterminise un automate non déterministe en ne représentant que les états accessibles de l'automate des parties.
7. Vérifier la correction de cette fonction sur les automates  $B_1$  et  $B_2$ . Combien d'états contiennent les versions déterminisées ?
8. Quelle est la complexité de cette nouvelle fonction en fonction du nombre d'états accessibles ?

## PALIER 3

### Exercice 4

1. Créer un type d'automate fini non déterministe avec  $\varepsilon$ -transitions, ainsi que les fonctions d'initialisation, de libération et d'ajout de transitions adéquates.
2. Écrire des fonctions qui calculent les clôtures avant et arrière d'un  $\varepsilon$ -AFND.