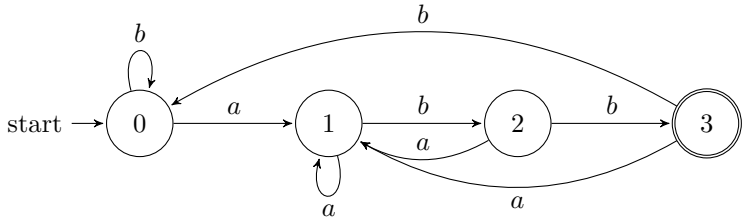
Automates finis déterministes

Dans ce TP, on se propose de manipuler des automates finis déterministes (AFD) en les représentant sous forme de graphes. Ils permettent notamment de reconnaître des langages représentés par une expression régulière. Un AFD est défini par :

1. un ensemble d'états $E = [0, |E| - 1]$;
2. l'alphabet des symboles d'entrée qui sera un sous-ensemble de $[a, z]$;
3. une fonction transiter qui associe à des couples états-symboles un état ; pour tous les couples états-symboles, il y a soit 0 règle de transition, soit 1 règle de transition ;
4. un état e_0 qui est distingué comme l'état de départ ou état initial ;
5. un ensemble d'états $F \subseteq E$ distingués comme état d'acceptation ou états finaux.

Un AFD peut être représenté graphiquement et informatiquement comme un graphe orienté étiqueté, appelé graphe de transition, dans lequel les sommets sont les états et les arcs étiquetés représentent la fonction de transition. Par exemple, le langage représenté par l'expression régulière $(a|b)^*abb$ peut être représenté par l'AFD suivant :



L'ensemble des états est $\{0, 1, 2, 3\}$, l'alphabet est $\{a, b\}$. L'état 0 est l'état de départ et l'état 3 est l'unique état final.

Un AFD accepte une chaîne d'entrée x si et seulement s'il existe un chemin entre l'état de départ et un état d'acceptation tel que les étiquettes d'arcs le long de ce chemin épellent le mot x. L'AFD de l'exemple accepte les chaînes *abb, aabb, babb, aaabb, ...* mais pas *bab*. Le langage défini par un AFD est l'ensemble des mots qu'il accepte.

1 Graphes

Dans un premier temps, on se propose de créer une structure permettant de représenter le graphe de transition.

1. Créer un fichier `graphe.h` contenant la structure `graphe` permettant de représenter un graphe sous forme de listes d'adjacence. On suppose les états (et donc les sommets) numérotés de 0 à $|E - 1|$. L'étiquette d'un arc est un caractère.
2. Créer un fichier `graphe.c`.
3. Créer une fonction `creer_graphe` qui permet d'initialiser une structure `graphe` contenant un nombre de sommets passé en paramètre et aucun arc. La structure sera allouée en mémoire dans cette fonction qui renverra l'adresse de la structure allouée.
4. Créer une fonction `ajouter_arc` qui permet d'ajouter un arc entre deux sommets passés en paramètre et qui lui assigne un symbole aussi passé en paramètre.
5. Créer une fonction `arc_existe` qui retourne 1 si l'arc entre deux sommets passés, étiquettés par un symbole aussi passé en paramètre existe, 0 sinon.
6. Créer une fonction `retirer_arc` qui prend en paramètre deux sommets et une étiquette et qui retire l'arc correspondant si ce dernier existe.
7. Créer une fonction `transiter` qui retourne l'état atteint à partir d'un état et d'une étiquette passés en paramètre. Si la transition n'existe pas, la fonction retourne -1.
8. Créer une fonction `afficher` qui affiche à l'écran le graphe en listant pour chaque sommet les sommets accessibles et les étiquettes associées.
9. Créer une fonction `liberer_graphe` qui prend en paramètre une référence sur un graphe alloué dynamiquement et qui libère l'espace mémoire occupé par la structure.

Si vous l'estimez utile, vous pouvez bien sur créer d'autres fonctions.

```
graphe.h
-------
struct larc {
  ... voisin;
  ... etiquette;
  ... arc_suiv;
};

struct graphe {
  ... nbsommets;
  ... adjs ; // ensemble de
             // listes d'adjacence
  ... ;
};
...
```

```
graphe.c
-------
... creer_graphe(...) {
}
... ajouter_arc(...) {
}
... arc_existe(...) {
}
... transition(...) {
}
...
```

2 Automate fini déterministe

Un AFD est caractérisé par un graphe de transition, un alphabet (qui sera représenté par une chaîne de caractères formée de lettres dans $[a, z]$, une lettre ne peut pas apparaître plus d'une fois), un entier représentant l'état initial et un tableau (par exemple) représentant les états finaux. De la même façon que pour un graphe, on créera les fichiers `automate.h` et `automate.c`

1. Créer une structure permettant de représenter un automate.
2. Créer une fonction permettant d'initialiser la structure à partir d'un ensemble d'états, d'un alphabet, d'un état initial et d'un tableau contenant les états finaux. On suppose au départ que l'automate ne contient aucune transition. La structure sera allouée en mémoire dans cette fonction qui renverra l'adresse allouée.
3. Ecrire une fonction qui prend en paramètre la référence sur un automate et libère toute la mémoire associée.
4. Créer une fonction qui prend en paramètre deux états e_1 et e_2 et une étiquette v et qui ajoute la transition $(e_1, v) = e_2$. La fonction doit vérifier que les deux états existent, que l'étiquette fait bien partie de l'alphabet qu'il n'y pas déjà de transition de e_1 vers e_2.

3 Langage

Le but est maintenant d'écrire des algorithmes liés à la reconnaissance du langage par l'automate.

1. Ecrire une fonction `accepte` qui prend en paramètres un automate et un mot (une chaîne de caractères) et qui renvoie 1 si le mot est reconnu par l'automate, 0 sinon.

2. Ecrire une fonction `affiche_langage_taille_croissante` qui affiche les mots reconnus par un automate en commençant par les mots de longueur 1, puis les mots de longueur deux jusqu'aux mots de longueur `maxlongueur` qui sera passé en paramètre de la fonction. Pour cela, vous adapterez l'algorithme de parcours en largeur vu en cours de graphes. Ici, on peut passer plusieurs fois par un sommet du graphe. Vous aurez besoin d'une structure de données vous permettant de manipuler une file.
3. Ecrire une fonction `affiche_langage_alphabetique` qui affiche les mots de longueur maximale `maxlongueur` (qui sera passé en paramètre de la fonction) reconnus par un automate par ordre alphabétique. Il s'agit ici d'adapter l'algorithme de parcours en profondeur vu en cours de graphes. Ici, on peut passer plusieurs fois par un sommet du graphe. Vous aurez besoin d'une structure de données vous permettant de manipuler une pile. Trier les listes d'adjacence par ordre alphabétique peut simplifier grandement la vie. Vous pourrez ainsi créer les deux fonctions suivantes :
   (a) Créer une fonction `trier_liste` qui prend en paramètre une liste d'adjacence et la trie alphabétiquement sur les étiquettes.
   (b) Créer une fonction `trier_listes` qui trie toutes les listes d'adjacence du graphe comme ci-dessus.

4 Ecriture et lecture dans un fichier

Dans un premier temps, créez les fonctions ci-dessous.

1. Créer une fonction `lire_graphe` qui prend en paramètre un descripteur de fichier et qui retourne un graphe qui sera alloué en mémoire dans la fonction.
2. Créer une fonction `ecrire_graphe` qui prend en paramètre un graphe et un descripteur de fichier.
3. Créer une fonction qui prend en paramètre un nom de fichier et qui retourne un AFD qui sera alloué en mémoire dans la fonction à partir des données du fichier
4. Créer une fonction qui prend en paramètre un automate et un nom de fichier et sauve l'automate dans le fichier.

Ecrire un programme qui prend un nom de fichier en paramètre, qui charge l'automate sauvegardé dans le fichier, qui demande à l'utilisateur s'il veut modifier l'automate (et dans ce cas, quelle transition il veut ajouter/retirer), s'il veut énumérer les langages, ou reconnaître un mot. A la fin du programme, l'automate est sauvegardé dans le fichier.