Option Informatique

TP : Automates

1 Langage reconnu par un automate fini non déterministe

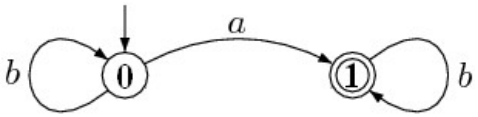
On définit le type afnd des automates finis (non déterministes) par :

```
type afnd = {initiaux : int list;
terminaux : int list ;
delta : (int*char*int) list};;
```

Les états sont numérotés par le type `int` (à renommage près, on considère que l'ensemble des états est inclus dans $\mathbb{N}$) et l'alphabêt considéré ici par le type `char`. L'ensemble des états initiaux et l'ensemble des états terminaux sont codés par des listes. Enfin, la relation de transition est codée par une liste de triplets (état avant transition, étiquette de la transition, état après transition).

Par exemple, l'automate représenté ci-dessous peut être écrit :

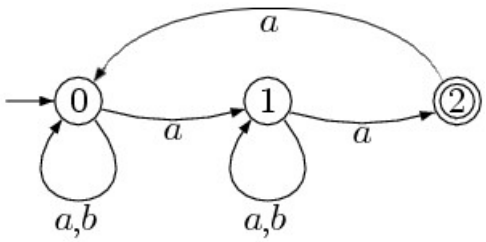
```
let automate1 = {init = [0]; finals = [1];
delta = [(0,'b',0);(0,'a',1);(1,'b',1)]};;
```



Le **langage reconnu par un automate** est l'ensemble des mots $u_1u_2\cdots u_n$ tels qu'il existe une suite de transitions étiquettées par u_1, ..., u_n permettant de passer d'un état initial à un état terminal.

1. Quel langage est reconnu par l'automate `automate1` ?

2. Définir l'automate `automate2` représenté ci-dessous. Quel langage reconnaît-il ?



On souhaite maintenant écrire une fonction `reconnu_par_afnd : string → afd → bool` qui détermine si un mot est ou pas reconnu par un afnd.

Première stratégie : on définit une fonction auxiliaire `accepte : afnd → string → int → int → bool` telle que `accepte automate chaine etat i` teste si, parmi les chemins qu'il est possible de parcourir dans `automate` en lisant le sous-mot de `chaine` commencant au caractère `i` à partir de l'état `etat`, il y en a au moins un qui conduit à un état terminal.

3. Écrire `images : (int * char * int) list → int * char → int list` telle que `images delta (i,c)` donne l'ensemble des états accessibles à partir de l'état `i` en lisant le caractère `c` pour la relation de transition `delta`.

4. Écrire la fonction `accepte`.

5. Écrire la fonction `reconnu_par_afnd`.

2 Un algorithme moins naif

Une stratégie un peu moins naïve est la suivante (et consiste en fait à déterminiser l'automate sans le dire).

Plutôt que de lister tous les chemins que l'on peut parcourir dans l'automate en lisant le mot pour tester si l'un de ces chemins s'achève sur un état terminal (ce qui correspond à un parcours en profondeur), on regarde après chaque lecture de lettre dans quel ensemble d'états on peut se trouver (en prenant garde d'éviter les répétitions) puis on teste à la fin si le dernier ensemble d'états comprend au moins un état terminal (ce qui correspond plutôt à un parcours en largeur).

6. Écrire une fonction `union : int list` $\to$ `int list` $\to$ `int list` permettant de former la réunion de deux ensembles d'états codés par des listes.

7. Réécrire la fonction `reconnu_par_afnd`.