

Ce TP est à effectuer en OCaml, sous Visual Studio Code, dans Codespaces ou sur un ordinateur personnel. Compléter le fichier `tp2.ml` fourni. Pour exécuter les définitions dans `utop`, sélectionner les lignes puis appuyer sur Maj + Entrée. Ajouter `;;` pour terminer une saisie dans `utop`. Interrompre une boucle infinie avec Ctrl + C.

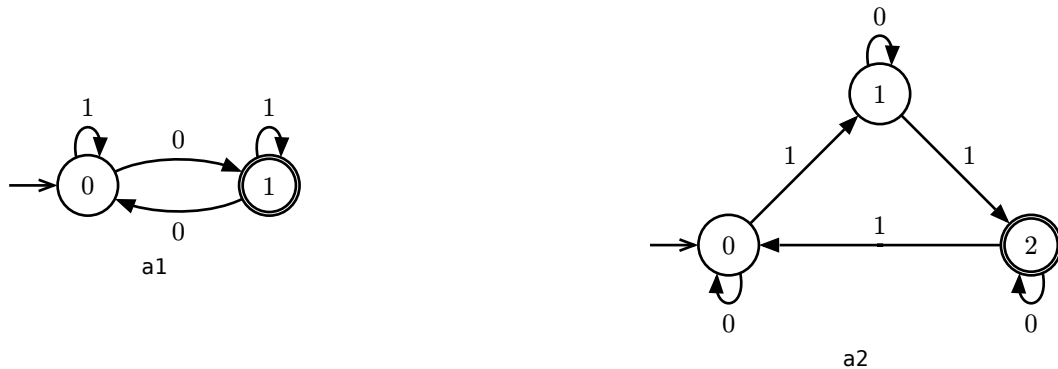
Automate déterministe

On représente un ADC par le type suivant :

```
type adc = {
  initial : int;
  finaux : int list;
  delta : int array array
}
```

Les états sont numérotés de 0 à $n - 1$ et les lettres sont des entiers : a est représenté par 0, b par 1, etc. Pour un automate a , $a.\text{delta}.(i).(j)$ est l'état atteint depuis l'état i en lisant la lettre j . Le nombre d'états est `Array.length a.delta`.

1. Définir en OCaml les automates **a1** et **a2** suivants :



- Définir `delta_etoile : adc -> int -> int list -> int` telle que `delta_etoile a q m` renvoie l'état atteint depuis q après lecture de m , c'est-à-dire $\delta^*(q, m)$. Vérifier avec **a1**.
- Définir `accepte : adc -> int list -> bool` telle que `accepte a m` détermine si m est reconnu par **a**. Vérifier avec **a1**. Quelle est sa complexité ?
- Définir `complementaire : adc -> adc` qui construit un automate reconnaissant le complémentaire du langage reconnu par **a**. Vérifier avec **a1**.
- Définir `accessibles : adc -> int list` qui détermine les états accessibles de **a** par un parcours en profondeur. Vérifier sur des exemples.

Indication. Utiliser un tableau `vus`, puis visiter récursivement chaque successeur non encore visité depuis l'état initial.

- En déduire `vide : adc -> bool` qui détermine si le langage reconnu par **a** est vide.

Automate produit

- Écrire `inter : adc -> adc -> adc` pour construire un automate reconnaissant l'intersection des langages reconnus par **a** et **b**, sur le même alphabet. Construire l'automate produit à np états, où n et p sont les nombres d'états de **a** et **b**. Numéroté l'état (i, j) par $i \times p + j$. Vérifier avec **a1** et **a2**.
- En déduire `inclus : adc -> adc -> bool` qui teste l'inclusion des langages reconnus par **a** et **b**.
- En déduire `equivalent : adc -> adc -> bool` qui teste l'égalité des langages reconnus par **a** et **b**.