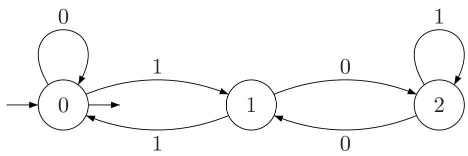
Simulation d'automates

Rappel : Comment construire un automate $\mathcal{A}_3$ sur l'alphabet $\Sigma = \{0, 1\}$ reconnaissant les mots $w \in \Sigma^*$ tels que $\overline{w}^2$ soit un entier divisible par 3 dont w est l'écriture binaire avec bit de poids fort en tête ?
Pour cela, il suffit de trois états représentants les restes modulo 3, et les transitions sont données par l'expression suivante :

$$\forall w \in \Sigma^*,\ \forall b \in \Sigma,\ \ \overline{wb}^2 = 2\overline{w}^2 + b$$

Ainsi, de l'état k partent les transitions étiquetées par $b \in \Sigma$ et arrivant dans $2k + b \bmod 3$. Cela donne l'automate suivant :



1. Introduction

Question 1.

(a) Définir une fonction `binaire_faible` qui à un entier associe la liste des bits de sa décompostion en base 2, le bit de poids le plus faible se trouvant en tête de liste.

(b) En déduire une fonction `binaire_fort` qui décompose un entier en base 2 en plaçant cette fois le bit de poids le plus fort en tête de liste.

L'entier 4 est représenté par $\overline{001}^2$ avec la convention bit de poids faible en tête et la liste obtenue par `(binaire_faible 4)` est `[0;0;1]` : $4 = 0.2^0 + 0.2^1 + 1.2^2$.
L'entier 4 est représenté par $\overline{100}^2$ avec la convention bit de poids fort en tête et la liste obtenue par `(binaire_fort 4)` est `[1;0;0]` : $4 = 1.2^2 + 0.2^1 + 0.2^0$.

```
binaire_faible : int -> int list
binaire_fort : int -> int list
```

2. Automates déterministes

On choisit de représenter un automate fini déterministe $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ à l'aide d'un type `'a` pour les états Q et d'un type `'b` pour l'alphabet Σ. Ceci permet de définir le type suivant :

```
type ('a, 'b) afd = { init: 'a ;
                      accept: 'a list ;
                      delta: (('a * 'b) * 'a) list} ;;
```

Ainsi si `auto1` est un automate, alors `auto1.init` permet d'accéder à l'état initial q_0, `auto1.accept` permet d'accéder à la liste des états acceptants, et `auto1.delta` permet d'accéder à la liste des transitions écrites sous la forme du couple $((q_i, a), q_j)$, avec $\delta(q_i, a) = q_j$.

Question 2.

(a) Redéfinir la fonction `mem` qui prend en arguments un élément x et une liste ℓ et renvoie le booléen $x \in \ell$.

(b) Définir la fonction `mem_fst` qui prend en arguments un élément x et une liste ℓ de couples et renvoie le booléen : $\exists y,\ (x, y) \in \ell$.

(c) Redéfinir la fonction `assoc` qui prend en arguments un élément x et une liste de couples ℓ et renvoie y tel que (x, y) est le premier élément dans la liste ℓ ayant x comme première coordonnée.

```
mem : 'a -> 'a list -> bool
mem_fst : 'a -> ('a * 'b) list -> bool
assoc : 'a -> ('a * 'b) list -> 'b
```

Question 3.
Définir une fonction `reconnu` qui détermine si un automate **déterministe** reconnait un mot de Σ^* (un mot sera représenté par le type `'b list` où la tête de liste contient la première lettre du mot).

```
reconnu : ('a, 'b) afd -> 'b list -> bool
```

On pourra utiliser une fonction auxiliaire qui contient en arguments l'état dans lequel on se trouve et le mot restant à lire.

Question 4.
Définir une fonction `genere_fort` qui à un entier d associe un automate déterministe $\mathcal{A}_d$ qui reconnait les entier divisibles par d lorsque ceux-ci sont lus en base 2 à partir du bit de poids le plus fort.
L'automate construit $\mathcal{A}_d$ a pour langage $\mathcal{L}(\mathcal{A}) = \left\{w \in \{0,1\}^* \mid \overline{w}^2 \equiv 0[d]\right\}$.
On numérotera les états de $\mathcal{A}_d$ par les entiers $[\![0, d-1]\!]$.

```
genere_fort : int -> (int, int) afd
```

Remarque : On testera la correction de cette fonction en utilisant les fonctions `binaire_fort` et `reconnu`.

3. Automates non déterministes

On choisit de représenter un automate fini non déterministe $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ par le type :

```
type ('a, 'b) afnd = { nd_init: 'a list ;
                       nd_accept: 'a list ;
                       nd_delta: (('a * 'b) * 'a) list} ;;
```

Question 5.
Comment peut-on, à partir d'un automate déterministe obtenu par la fonction `genere_fort`, obtenir un automate **non déterministe** qui reconnaît les entiers divisibles par d lorsque ceux-ci sont lus à partir du bit de poids faible ?
Écrire une fonction `genere_faible` qui génère un tel automate.

```
genere_faible : int -> (int, int) afnd
```

On pourra utiliser l'automate déterministe obtenu à partir de `genere_fort` *et modifier astucieusement les transitions.*

Question 6.
Redéfinir la fonction `exists` qui prend en entrée un prédicat p et une liste ℓ et renvoie vrai si $\exists x \in \ell, p(x)$.

```
exists : ('a -> bool) -> 'a list -> bool
```

Question 7.
Définir une fonction `reconnu2` qui détermine si un automate non déterministe reconnaît un mot de Σ^*.

```
reconnu2 : ('a, 'b) afnd -> 'b list -> bool
```

Attention, l'automate est non déterministe. Il faut prendre en compte le fait qu'il existe un chemin partant d'un état initial et arrivant à un état final. On pourra construire une fonction auxiliaire prenant en arguments l'état courant, le mot restant à lire et les transitions possibles. En effet, lorsque que l'on est dans un état q et qu'on lit une lettre a, il y a éventuellement plusieurs transitions à visiter...

Remarque : On testera la correction de cette fonction en utilisant les fonctions `binaire_faible` et `reconnu2`.