1 Introduction

Si l'on considère l'alphabet $\Sigma = \{0, 1\}$, on souhaiterait construire un automate $\mathcal{A}_3$ tels que :

$$\mathcal{L}(\mathcal{A}_3) = \{w \in \Sigma^* \mid \overline{w}^2 = 0 \mod 3\}$$

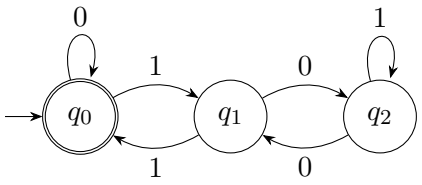
où $\overline{w}^2$ est l'entier dont w est l'écriture en binaire (avec bit de poids fort en tête).

Pour cela, il suffit de se donner un ensemble de trois états $Q = \{q_0, q_1, q_2\}$, et de trouver les transitions permettant de garantir la proposition suivante : si on lit un mot représentant un entier congrus à $i \mod 3$, alors on est en q_i.

Le seul état acceptant sera alors q_0, et les transitions sont obtenues par l'équation suivante :

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

Ainsi, de l'état q_i partent les transitions étiquetées par $b \in \Sigma$ et arrivant en q_k avec $k = 2i + b \mod 3$. On obtient alors l'automate suivant :



Le but de ce TP est de généraliser ce processus pour détecter à l'aide d'un automate si un entier codé en binaire est divisible par un entier d fixé.

Exercice 1 (Écriture binaire) :

1. Écrire une fonction `binaire_faible : int -> int list` qui à un entier associe la liste des bits de sa décomposition en base 2, le bit de poids le plus faible se trouvant en tête de liste.
2. En déduire une fonction `binaire_fort : int -> int list` qui décompose un entier en base 2 en plaçant cette fois le bit de poids le plus fort en tête de liste.

Exemple. L'entier $6 = 0.2^0 + 1.2^1 + 1.2^2$ est représenté par $\overline{011}^2$ avec la convention bit de poids faible en tête et par $\overline{110}^2$ avec la convention bit de poids fort en tête.

Exemple

```
# binaire_faible 6 ;;
- : int list = [0; 1; 1]
# binaire_fort 6 ;;
- : int list = [1; 1; 0]
```

2 Automates déterministes

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

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

Ainsi, si `auto1` est un automate, alors :

- `auto.init` permet d'accéder à l'état initial q_0 ;

- `auto.accept` permet d'accéder à la liste des états acceptants ;
- `auto.delta` permet d'accéder à la liste des transitions, écrites sous la forme du couple $((q_i, a), q_j)$ avec $\delta(q_i, a) = q_j$.

On fournit une représentation de l'automate $\mathcal{A}_3$ sur la page web du cours.

Exercice 2 (Automates déterministes) :

3. Écrire une fonction `mem : 'a -> 'a list -> bool` qui prend en arguments un élément x et une liste ℓ et renvoie le booléen $x \in \ell$.
4. Écrire une fonction `mem_fst : 'a -> ('a * 'b) list -> bool` qui prend en arguments un élément x et une liste ℓ de couples et renvoie le booléen : $\exists y \in$ `'b`, $(x, y) \in \ell$.

Exemple

```
mem_fst (1,1) auto3.delta ;;
- : bool = true
# mem_fst (1,1) auto3.delta ;;
- : bool = true
# mem_fst (2,2) auto3.delta ;;
```

5. Écrire une fonction `assoc : 'a -> ('a * 'b) list -> 'b` 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.
   On lèvera l'exception `Not_found` si un tel y n'existe pas.

Exemple

```
- : bool = false
# assoc (1,1) auto3.delta ;;
- : int = 0
# assoc (2,2) auto3.delta ;;
Exception: Not_found.
```

6. Écrire une fonction `reconnu : ('a, 'b) afd -> 'b list -> bool` 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).

Exemple

```
# reconnu auto3 (binaire_fort 6) ;;
- : bool = true
# reconnu auto3 (binaire_fort 7) ;;
- : bool = false
# reconnu auto3 (binaire_fort 8) ;;
- : bool = false
```

7. Écrire une fonction `genere_fort : int -> (int, int) afd` 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.

Exemple

```
# let a3 = genere_fort 3 ;;
...
# reconnu a3 (binaire_fort 365) ;;
- : bool = false
# reconnu a3 (binaire_fort 364) ;;
- : bool = false
# reconnu a3 (binaire_fort 363) ;;
- : bool = true
# reconnu a3 (binaire_fort 362) ;;
- : bool = false
```

Exemple

```
# let a5 = genere_fort 5 ;;
...
# reconnu a5 (binaire_fort 365) ;;
- : bool = true
# reconnu a5 (binaire_fort 364) ;;
- : bool = false
# reconnu a5 (binaire_fort 363) ;;
- : bool = false
# reconnu a5 (binaire_fort 362) ;;
- : bool = false
# reconnu a5 (binaire_fort 361) ;;
- : bool = false
# reconnu a5 (binaire_fort 360) ;;
- : bool = true
```

3 Automates non déterministes

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

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

Remarque. On autorise ici plusieurs états initiaux, dénotés par en ensemble $I \subseteq Q$.

Exercice 3 (Automates non déterministes) :

8. Comment peut-on, à partir de l'automate renvoyé par `genere_fort d`, obtenir un automate fini non déterministe qui reconnait les entiers divisibles par d lorsque ceux-ci sont lus à partir du bit de poids faible ? Écrire une fonction `genere_faible : int -> (int, int) afnd` qui génère un tel automate.
9. Écrire une fonction `exists : ('a -> bool) -> 'a list -> bool` qui prend en entrée un prédicat p et une liste ℓ et renvoie vrai si $\exists x \in \ell,\ p(x)$.
10. Écrire une fonction `reconnu2 : ('a, 'b) afnd -> 'b list -> bool` qui détermine si un automate non déterministe reconnaît un mot de Σ^*.

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 (on pourra faire du backtracking).

Exemple

```
# let af5 = genere_faible 5 ;;
...
# reconnu2 af5 (binaire_faible 3664064) ;;
- : bool = false
# reconnu2 af5 (binaire_faible 3664061) ;;
- : bool = false
# reconnu2 af5 (binaire_faible 3664063) ;;
- : bool = false
# reconnu2 af5 (binaire_faible 3664065) ;;
- : bool = true
```

4 Automates émondés

On se propose dans cette partie de représenter un automate fini déterministe par le type suivant :

```
type afd2 = {
  nb : int;
  init : int;
  final : int list;
  trans: (int*char*int) list
}
;;
```

On prend la convention que si n est le nombre d'états, alors les états sont $Q = [\![0, n-1]\!]$.

Le but de cette partie est d'émonder un automate, c'est-à-dire de supprimer ses états inutiles. Pour cela, il va nous falloir raisonner sur le graphe orienté sous-jacent (en oubliant les étiquettes des transitions).

```
type graph = int list array ;;
```

Si `g` est de type `graph` de taille n, alors `g` représente le graphe $G = (S, A)$ avec $S = [\![0, n-1]\!]$, et pour tout sommet i, `g.(i)` est la liste d'adjacence de i.

Exercice 4 (Automates émondés) :

11. Écrire une fonction `auto_to_graph : afd2 -> graph` qui prend en entrée un automate et renvoie le graphe associé.

Exemple

```
# let g1 = auto_to_graph ex1;;
val g1 : int list array =
  [|[0; 1]; [0; 2]; [3; 6]; []; [1; 0]; [5; 4]; [3]; [7; 3]|]
```

12. Écrire une fonction `transpose : graph -> graph` qui prend en entrée un graphe orienté, et renvoie son graphe transposé (où tous les arcs sont inversés).

Exemple

```
# let g1' = transpose g1 ;;
val g1' : int list array =
  [|[4; 1; 0]; [4; 0]; [1]; [7; 6; 2]; [5]; [5]; [2]; [7]|]
```

13. Écrire une fonction `intersecte : 'a list -> 'a list -> 'a list` qui prend en entrée deux listes et renvoie une liste contenant uniquement les éléments communs aux deux listes.
14. Écrire une fonction `nettoie : 'a list -> 'a list` qui prend en entrée une liste et renvoie une liste où chaque élément n'apparait qu'une seule fois.
15. Écrire une fonction `numeros : int -> int list -> int -> int` telle que, si ℓ est une liste de valeurs inférieures à `nb`, alors `numeros nb l i` la valeur k telle que i est la k-ème plus petite valeur de ℓ (on renverra -1 si $i \notin \ell$).
16. Écrire une fonction `parcours : graph -> int -> int list` qui prend en entrée un graphe et un sommet source, et renvoie une liste des sommets atteignables depuis la source.
17. Écrire une fonction `emonde : afd2 -> afd2` qui prend en entrée un automate et renvoie l'automate émondé. On prendra garde à renuméroter les états utiles. Si l'automate émondé est vide (`nb = 0`), on renverra par convention : `{nb = 0; init = -1; final = []; trans = []}`.

Exemple

```
# emonde ex1 ;;
- : afd2 =
{nb = 5; init = 0; final = [1; 3];
 trans =
  [(0, 'a', 1); (0, 'b', 0); (1, 'a', 2); (1, 'b', 0); (2, 'a', 4);
   (2, 'b', 3); (4, 'b', 3)]}
```