TP n° 6 – Automates finis

esimon@esimon.fr thomas.h@ens.fr
thomash.fr/teaching/

On représente un automate fini par :

```
type automate = { transition : (char * int) list vect ; final : bool vect} ;;
```

L'alphabet sur lequel est défini l'automate est l'ensemble des valeurs d'un `char`. Son état initial est l'état 0. Un état i est final si `final.(i)=true`.

Les transitions sont représentées par un tableau de listes semblable à une représentation par listes d'adjacence d'un graphe. Par exemple, une transition vers l'état 42 étiqueté par « m » sera représentée par un couple `(`m`, 42)`.

1 Oups !

Question 1. Yorel Reivax a oublié le nom de son tahc[1], fort heureusement, il a un robot qui a été entraîné à reconnaître le nom de Yorel et de son animal de compagnie :

```
let robot = { transition=[| ['Y', 1; 'M', 2]; ['O', 3]; ['A', 3]; ['R', 4]; ['E',
    5]; ['L', 6]; [] |]; final=[|false ; false ; false ; false ; false ; false ; true
   |] } ;;
```

Quel est le nom du tahc de Yorel ?

2 Automates finis déterministes

Dans cette partie on s'intéressera uniquement à des automates déterministes (à partir de chaque état il n'existe qu'une seule transition portant une lettre donnée). La fonction standard `assoc : 'a -> ('a * 'b) list -> 'b` risque de s'avérer utile.

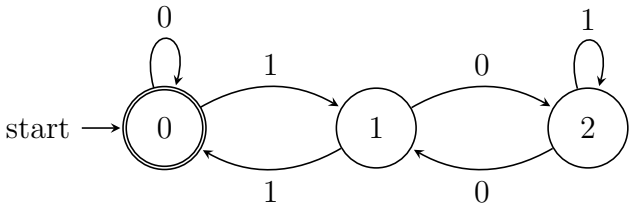

FIGURE 1 – Exemple d'un automate déterministe reconnaissant les mots binaires multiples de 3.

Question 2. Écrire l'automate `double: automate` qui reconnaît le langage $(aa + bb)^*$.

Question 3. Écrire une fonction `accepte: automate -> string -> bool` qui décide si un mot est accepté par un automate.

Question 4. Écrire une fonction `langage_vide: automate -> bool` qui renvoie `true` si et seulement si l'automate reconnaît le langage vide (et n'accepte donc aucun mot.)

Question 5. Écrire une fonction `minimum_acceptable: automate -> string -> string option` qui étant donné un automate et un préfixe, renvoie un plus petit mot accepté par l'automate commençant par ce préfixe ou `None` si un tel mot n'existe pas.

1. Un animal qui fait uoaim.

3 Automates finis non déterministes

On regarde maintenant les automates finis non déterministes, c'est à dire les automates finis dans lesquels il peut y avoir plusieurs transitions provenant d'un même état et étiquetées d'une même lettre [2].

Question 6. Écrire une fonction `poly_assoc: 'a -> ('a * 'b) list -> 'b list` qui étant donnée une liste associative et une clef renvoie la liste de toutes les valeurs associées à cette clef. Elle vous sera utile.

Question 7. Écrire une fonction `naccepte: automate -> string -> bool` qui détermine si un mot est accepté par un automate fini non déterministe. Votre fonction devra avoir une complexité polynomiale en temps dans le pire cas, vous devrez certainement maintenir la liste des états accessibles après la lecture d'un préfixe du mot.

Question 8. * Écrire une fonction `intersection: automate -> automate -> automate` qui étant donné deux automates reconnaissant les langages $\mathcal{L}_1$ et $\mathcal{L}_2$, renvoie un automate reconnaissant le langage $\mathcal{L}_1 \cap \mathcal{L}_2$.

Question 9. * Il est possible de déterminiser un automate fini, pour cela, il faut considérer l'automate dont les états sont des parties de l'ensemble des états de l'automate d'origine. Écrire une fonction `determinise: automate -> automate` qui déterminise un automate.

2. Remarquez que tous les automates déterministes sont des automates non déterministes.