Automates Déterministes

Introduction

Ce TD traite de la reconnaissance de certains langages par des automates déterministes. Nous y construisons un interpréteur et un émondeur.

1 Représentation des automates

On choisit de représenter les transitions des automates par des listes de triplets (état, lettre, état). Cette représentation a l'avantage de servir aussi bien pour les automates déterministes et non-déterministes et elle conserve la symétrie gauche-droite de la lecture.

Un automate sera donc un enregistrement à quatre champ : l'alphabet, l'ensemble des états initiaux, l'ensemble des états finaux, la liste des transitions. Par contre, les ϵ-transitions ne sont pas possibles avec cette définition de type.

Question 1 *Créer le type* `('a,'b) automate`*,* `'a` *étant le type des états de l'automate et* `'b` *étant le type de ces lettres.*

Question 2 *Ecrire une fonction qui dresse la liste des états d'un automate (on pourra utiliser une fonction* `sans_doublon: 'a list -> 'a list` *qui élimine les doublons d'une liste):*

```
etats: ('a,'b) automate -> 'a list
```

Question 3 *Ecrire une fonction qui affiche agréablement un automate (pas un graphe, mais la liste des transitions de façon lisible), elle prend en arguments les fonctions d'impression des états et des caractères :*

```
affiche: ('a,'b) automate -> ('a -> unit) -> ('b -> unit) -> unit
```

Question 4 *Ecrire une fonction qui test si un automate est bien déterministe:*

```
isdeterm : ('a,'b) automate -> bool
```

Essayer les fonctions précédentes sur $\mathcal{A}_1$, $\mathcal{A}_2$ et $\mathcal{A}_3$.

2 La reconnaissance par un automate déterministe

Question 5 *Ecrire une fonction qui renvoie l'image d'un état par une lettre, elle lève l'exception* `Not_found` *s'il n'y a pas d'image:*

```
image : ('a,'b) automate -> 'a -> 'b -> 'a
```

Question 6 *Ecrire un interprète. Il prendra en argument l'automate et la liste de symboles restant à lire. Il renverra un booléen indiquant si le mot est accepté ou non.*

```
reconnait : ('a,'b) automate -> 'b list -> bool
```

Pour utilser cette fonction, on transforme les chaînes de caractères en liste de charactères par la fonction :

```
let string_to_list s =
  let n = string_length s in
  let rec aux = function
    | i when i = n -> []
    | i -> s.[i] :: (aux (i+1)) in
  aux 0;;
```

Essayer la fonction sur $\mathcal{A}_1$.

3 Emondons les automates (et ils nous le rendrons)

Définition 1 *Soit $\mathcal{A}$ un automate. Un état q de $\mathcal{A}$ est accessible s'il existe dans $\mathcal{A}$ un chemin partant d'un état initial et arrivant en q. L'état q est co-accessible s'il existe un chemin de $\mathcal{A}$ partant de q et arrivant à un état final.*

Définition 2 *Un automate $\mathcal{A}$ est accessible si tous ses états sont accessibles. Il est co-accessible si tous ses états sont co-accessibles. Il est émondé s'il est accessible et co-accessible.*

Dans un automate, seuls les états accessibles *et* co-accessibles sont utiles.

Soient $\mathcal{A}$ l'automate *(Q, T, I, F)* et Q' la suite :

$$\begin{cases} Q'_0 & = I \\ Q'_{n+1} & = Q'_n \cup \bigcup_{a,q \in A \times Q'_n} \{q' \in Q \mid (q, a, q') \in T\} \end{cases}$$

Question 7 *Soit Q_{acc} l'ensemble des états accessibles. Montrer qu'il existe un entier N, tel que $n \geq N$ implique $Q'_n = Q_{acc}$. En déduire un algorithme pour calculer les états accessibles de $\mathcal{A}$. Ecrire une fonction qui prend en argument une liste de transitions et une liste d'états initiaux, et renvoie la liste des états accessibles. On fera en sorte que les* `'a list` *soient ordonnées de manière à s'assurer que l'égalité des ensembles corresponde à l'égalité des listes.*

`accessibles : ('a * 'b * 'a) list -> 'a list -> 'a list`

Question 8 *S'inspirer de la question précédente pour proposer un algorithme qui calcule l'ensemble des états co-accessibles d'un automate. Ecrire une fonction qui prend en argument une liste de transitions et une liste d'états finaux, et renvoie la liste des états co-accessibles.*

`coaccessibles : ('a * 'b * 'a) list -> 'a list -> 'a list`

Question 9 *Ecrire une fonction qui prend en argument un automate et renvoie son émondé. Essayer sur $\mathcal{A}_2$.*

Question 10 *Quel est le langage $L(\mathcal{A}_3)$ reconnu par l'automate $\mathcal{A}_3$?*

Annexe : La liste des fonctions utiles

On donne quelques fonctions prédéfinies qui pourront servir:

```
value mem : 'a -> 'a list -> bool
      mem a l is true if and only if a is structurally equal
      (see module eq) to an element of l.

value map : ('a -> 'b) -> 'a list -> 'b list
      map f [a1; ...; an] applies function f to
      a1, ..., an, and builds the list [f a1; ...; f an] with
      the results returned by f.

value it_list : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
      it_list f a [b1; ...; bn] is f (... (f (f a b1) b2) ...) bn.

value do_list : ('a -> 'b) -> 'a list -> unit
      do_list f [a1; ...; an] applies function f in turn to
      a1; ...; an, discarding all the results.  It is equivalent to
      begin f a1; f a2; ...; f an; () end.

value intersect : 'a list -> 'a list -> 'a list
      intersect l1 l2 returns the list of the elements of l1
      that are structurally equal to an element of l2.

value union : 'a list -> 'a list -> 'a list
      union l1 l2 appends before list l2 all the elements of
      list l1 that are not structurally equal to an element of l2.
```