Déterminisation des automates

Introduction

Un automate non-déterministe n'est pas toujours utilisable en pratique car il demande pour la reconnaissance un calcul assez complexe et plus lent que pour un automate déterministe. Ainsi, dans certains cas, il est utile de transformer un automate non-déterministe en un automate déterministe équivalent.

On a vu pour la reconnaissance que l'on manipulait en fait des ensembles d'états et non des états. La théorie montre que l'on peut créer un nouvel automate déterministe, cette fois-ci, dont les états sont les ensembles obtenus lors de la reconnaissance et qui reconnait le même language que l'automate initial.

C'est ce que nous nous proposons de faire dans ce TD. Dans un premier temps, nous n'examinerons que les automates sans ϵ-transitions, puis nous écrirons un programme qui supprime ces transitions lorsqu'il y en a.

1 Déterminisons les automates

Reprenons la représentation introduite au premier TD :

```
type ('a,'b) automate = { init : 'a list ;
    term : 'a list ; trans : ('a * 'b * 'a) list};;
```

L'algorithme que nous allons employer est le suivant : en partant de l'ensemble des états initiaux, on calcule d'abord tous les ensembles d'états que l'on peut atteindre à partir de celui-ci et par toutes les lettres de l'alphabet, puis on recommence sur les nouveaux états obtenus jusqu'à ne plus trouver de nouveaux états.

Il faut donc à chaque étape traiter une liste d'états (qui est un état du nouvel automate), ajouter les nouveaux états obtenus à une liste qui contient les états déjà vus ainsi qu'à une liste qui contient les états restants à traiter, jusqu'à ce que cette dernière liste soit vide. Pour traiter une liste d'état, plutôt que d'utiliser la fonction `image`, nous allons le faire en un seul parcours de la liste des transitions : à chaque nouvelle transition, on ajoute l'état dans une liste de type `('b * ('a list)) list`, (`'a` étant le type des états de l'automate et `'b` étant le type de ces lettres), qui contient les transitions trouvées au départ de l'état en cours de traitement. Ensuite, on parcourt cette liste et on ajoute les nouvelles transitions à la liste des transitions du nouvel automate et les nouveaux états créés à la liste des états déjà vus et à la liste des états restants.

Question 1 *Créer une fonction qui calcule la liste de type* `('b * ('a list)) list` *contenant les transitions trouvés au départ une liste d'états données :*

```
nouv_trans : 'a list -> ('a * 'b * 'c) list -> ('b * 'c list) list
```

où `'a list` *est la liste d'états et* `('a * 'b * 'c) list` *la liste des transitions de l'automate.*

Question 2 *Créer la fonction qui déterminise un automate :*

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

Je rappelle que les états finaux de l'automate obtenu sont les états qui contiennent au moins un état final de l'automate de départ.

2 Les ϵ-transitions

Dans certains cas, il est utile d'ajouter un type particulier de transition qui "reconnaissent" le mot vide ϵ.

Question 3 *A partir du type* `automate`*, créer un type* `automate_e` *permettant de coder les ϵ-transitions.*

Question 4 *En s'insprirant de la fonction* `image` *pour les automates non-déterministes, créer une fonction* `cloture` *qui donne la cloture d'une liste d'états :*

```
cloture : ('a, 'b) automate_e -> 'a list -> 'a list
```

Question 5 *A partir des fonctions* `image` *et* `cloture`*, créer une fonction* `image_e` *prennant en compte les ϵ-transitions :*

```
image_e: ('a,'b) automate_e -> 'a list -> 'b -> 'a list
```

Question 6 *Montrer que l'on peut enlever les ϵ-transitions dans un automate non-déterministe (c'est à dire obtenir un automate non-déterministe sans ϵ-transitions qui reconnait le même language) et écrire la fonction réalisant cette opération.*

```
ote_e: ('a,'b) automate_e -> ('a,'b) automate
```