Institut National Universitaire Champollion

Théorie des langages

L3 INFO - Semestre 6

TP n° 5 : Émonder un automate

Piste Noire

L'objectif de ce TP est d'implémenter l'algorithme pour émonder un automate, vu en cours. Rappelons que la stratégie est la suivante :

- (§1) Déterminer les états accessibles.
- (§2) Construction de l'automate inverse, recherche des sommets co-accessibles
- (§3) Construction de l'automate émondé *

Introduction

Dans le fichier *.ml*, vous retrouverez les types `etatN` et `afn` pour représenter en Caml les automates non déterministes.
Vous retrouverez, également, la fonction `transitN` qui teste s'il existe des transitions étiquetées par le caractère c dans un automate aut à partir d'un état i. La fonction lève l'exception `PasTransition` s'il n'y a pas de telles transitions.

Dans ce fichier, on se donne un automate noté `an1`, qui servira pour les exemples. Il s'agit du même automate que l'animation *Émonder un AFD* sur Moodle.

Par ailleurs, nous notons `sigmaAug = ['a';'b';'e']`, l'alphabet Σ auquel nous rajoutons le mot vide ε.

1. Déterminer les états accessibles

L'objectif de cette section est de déterminer les états accessibles, c'est à dire, la liste des états que l'on peut atteindre à partir des états initiaux. Pour cela, nous effectuerons un parcours en largeur de l'automate à partir des états initiaux.

Remarquez que cela ressemble beaucoup au calcul des clôtures du TP précédent. Dans ce TP, nous avions effectué un parcours en largeur de l'automate en empruntant seulement les ε-transitions.

Écrire une fonction `etatsAccessibles : afn -> int list` qui détermine les états accessibles d'un automate à partir de l'ensemble des états initiaux.

```
etatsAccessibles an1 ;;
(*- : int list = [1; 4; 3; 6; 5]*)
```

2. Construction de l'automate inverse, recherche des sommets co-accessibles

À présent, nous devons déterminer les états co-accessibles, c'est à dire les états à partir desquels il est possible de rejoindre un état acceptant. La stratégie est la suivante :

- Construire l'automate inverse (nous allons inverser les transitions)
- Dans l'automate inverse, effectuer un parcours à partir des états acceptants à l'aide de la fonction du paragraphe précédent (§1).

Construction de l'automate inverse

Écrire une fonction `autoInverse : afn -> afn` qui inverse les transitions d'un automate.

```
let an2 = autoInverse an1 ;;

(an2.eN(1)).tN('b') ;;    (*- : int list = [2] *)
(an2.eN(1)).tN('a') ;;    (*Exception : PasTransition.*)
(an2.eN(2)).tN('a') ;;    (* - : int list = [2] *)
(an2.eN(2)).tN('b') ;;     (*Exception : PasTransition.*)
(an2.eN(3)).tN('a') ;;     (*- : int list = [4 ; 1]*)
```

Déterminer les états co-accessibles

Écrire une fonction `etatsCoAccessibles : afn -> int list` qui détermine les états co-accessibles d'un automate.

```
etatsCoAccessibles an1 ;;
(*- : int list = [2 ; 1 ; 4 ; 3]*)
```

3. Construction de l'automate émondé

Écrire une fonction `emonde : afn -> afn` qui émonde un automate. Vous pouvez écrire des fonctions auxiliaires.