Institut National Universitaire Champollion

TP 2 - Lecture d'un mot par un automate

Théorie des langages
L3 INFO - Semestre 6

Activité n° 1 : Lecture d'un mot par un Automate Fini Déterministe

Piste Verte

L'objectif de ce TP est de manipuler les automates finis déterministes. Les points importants de ce TP sont :

- Bien comprendre le type `afd` que l'on retrouvera dans les prochains TPs (§1)
- La lecture d'un mot par un automate fini déterministe quelconque (§2.2)

1. Rappel : Le type afd

1.1. Le type etat

Pour définir les automates finis déterministes, nous introduisons tout d'abord le type produit suivant :

```
type etat = {accept : bool ; t : char -> int} ;;
```

Ce type permet de décrire les états de l'automate. Le booléen `accept` est égal à `true` si l'état est acceptant, `false` sinon. La fonction `t` décrit les transitions à partir de cet état.

1.2. Le type afd

Nous représenterons les automates finis déterministes par le type produit suivant :

```
type afd = {sigma : char list ; nQ : int ; init : int ; e : int -> etat} ;;
```

Ici, `sigma` désigne l'alphabet, `nQ` est le nombre d'états numérotés de 1 à nQ, `init` est le numéro de l'état initial et enfin, e est une fonction qui à chaque entier désignant un état associe sa description de type `etat`.

Pour tester votre compréhension de ces nouveaux types Caml, quelques fonctions de base :

1. Écrire les fonctions suivantes :
   - `nombreEtats : afd -> int` qui renvoie le nombre d'état d'un automate passé en paramètre.
   - `etatInitial : afd -> int` qui renvoie le numéro de l'état initial d'un automate.
   - `descriptionE : afd -> int -> etat` qui renvoie la description de l'état numéro i dans un automate.
   - `nombreEtatsAcc : afd -> int` qui renvoie le nombre d'états acceptants d'un automate.

2. Lecture d'un mot par un automate

2.1. Cas d'un automate complet

On suppose dans un premier temps que l'automate fini déterministe est **complet**, comme c'est le cas de l'automate exemple a_1.

1. Écrire une fonction `lireComplet : afd -> int -> string -> int` (le type `afd -> int * string -> int` est aussi accepté) qui effectue la lecture d'un mot w (de type `string`), à partir d'un état i dans un AFD.
   Votre fonction renvoie l'état dans lequel on arrive à la fin de la lecture du mot. Vous utiliserez les fonctions sur les chaînes de caractères données dans le sujet du TP.

```
lireComplet a1 1 "ab" ;;
(*- : int = 3 *)
```

2.2. Cas d'un automate incomplet

Nous supposons à présent que les automates peuvent être incomplets. Nous donnons un automate a_2 en exemple.

1. Quel langage est reconnu par cet automate ? Que se passe-t-il lorsque vous testez votre fonction `lireComplet` sur cet exemple ?
2. Pour améliorer votre fonction `lireComplet` afin qu'elle prenne en compte les automates incomplets, procédez comme suit :
   (a) Définissez une exception PasTransition.
   (b) Écrire une fonction `transit : afd * int * char -> int` qui prend en paramètres un automate, un état q et une lettre $'a'$ et renvoie l'état q' si $(q,'a',q')$ est une transition de l'automate, et lève l'exception sinon.
   (c) Modifier alors votre fonction `lireComplet` en une fonction `lireIncomplet` : en cas d'absence de transition, votre fonction renvoie l'entier 0, sinon le numéro de l'état dans lequel on arrive à l'issue de la lecture.
3. En déduire une fonction `lire : afd * string -> int` qui lit un mot dans un automate à partir de l'état initial.

```
lire(a1,"abba") ;;
(*- : int = 3*)
```

```
lire(a1,"bbaaa") ;;
(*- : int = 2*)
```

```
lire(a2,"babb") ;;
(*- : int = 0*)
```

4. Écrire une fonction `accepte : afd -> string -> bool` qui teste si un mot est reconnu ou non par un automate.

```
let ac1 = accepte a1 ;;
val ac1 : string -> bool = <fun>

List.map ac1 ["abba";"bbaaa";"bbaaba";"ba";"ab";""] ;;
- : bool list = [true ; false ; true ; false ; true ; false]

accepte a2 "babb" ;;
- : bool = false
```

5. Définir un automate qui reconnaît les mots de la forme $a^n b$. Tester vos fonctions.