TP 2 : Automates

23 novembre

1 Automates déterministes

Les automates que l'on considère dans cette partie sont déterministes (pour chaque couple (état,lettre), il y a *au plus* une transition).

L'enregistrement `taille` donne le nombre d'états. L'état initial est donné par l'enregistrement `initial`, et les états finaux sont donnés par le tableau de booléens `final`. Le tableau `transitions` contient l'ensemble des transitions.

La taille des tableaux `transitions` et `final` doit être `taille`, mais ceci n'est pas spécifié dans le type.

```
type automate =
  { taille :  int ;
    initial :  int ;
    transitions :  (char * int) list array ;
    final :  bool array } ;;
```

▷ **Question 1.** Il existe dans la bibliothèque Caml une fonction `assoc` définie de la manière suivante: `assoc : 'a -> ('a * 'b) list -> 'b` qui gère les listes de couples (appelées aussi listes associatives) : par exemple `assoc 2 [(1,'a'); (2,'b') ; (2,'c') ; (3,'d')]` vaut `'b'`. La fonction `assoc` déclenche l'exception `Not found` en cas d'échec.

Ecrire une telle fonction. ◁

▷ **Question 2.** Écrire une fonction `calcul_det` qui étant donné un mot et un automate supposé déterministe, détermine si l'automate accepte ce mot (on pourra utiliser la fonction `assoc` ou bien la fonction que vous venez de coder). Quelle est sa complexité ?

Définir l'automate 1 représenté à la fin du TP, et vérifier sur les exemples aa, aba et bab que la fonction `calcul_det` est correcte. ◁

2 Automates non-déterministes

Considérons à présent les automates non-déterministes, toujours avec le type `automate` où on ne suppose plus que chaque liste du tableau `transitions` ne contient qu'une seule occurence pour chaque lettre. On modifiera aussi le type du champs `initial` en une liste.

▷ **Question 3.** Ecrire une fonction `delta : automate -> int -> char -> int list` telle que `delta a q x` renvoie la liste correspondant à l'ensemble $\delta(q, x)$. ◁

2.1 Recherche en profondeur d'abord

La recherche en profondeur d'abord consiste à tester les chemins étiquetés par le mot `u` les uns après les autres, jusqu' à en trouver un, s'il en existe, qui soit un chemin réussi (s'il n'en existe pas alors `u` n'est pas reconnu).

▷ **Question 4.** Écrire une fonction récursive `rech_prof : automate -> char list -> int list -> bool` telle que `rech_prof a m l` renvoie `true` si et seulement s'il existe, dans l'automate associé à a un chemin étiqueté par `m` dont l'origine est un état de l et la fin est un état final. ◁

▷ **Question 5.** Écrire une fonction `reconnait_prof : automate -> string -> bool` telle que `reconnait_prof` renvoie `true` si et seulement si le mot `m` est reconnu par l'automate associé à `a`. ◁

Définir l'automate 2 représenté à la fin du TP , et vérifier sur les exemples aa, aba et bab.

2.2 Recherche en largeur

Soit $u = a_1 a_2 \dots a_n$ un mot et $A = \{Q, I, F, T\}$ un automate. On construit la suite (E_p) définie par : $E_0 = I$, pour tout entier $p \leq n$, $E_p = \{q \in Q | \exists q' \in E_{p-1}, (q', a_p, q) \in T\}$ où T représente l'ensemble des transitions.

▷ **Question 6.** Montrer que $u \in \mathcal{L}(A)$ si et seulement si $E_n \cap F \neq \emptyset$. ◁

▷ **Question 7.** Écrire une fonction `etape : automate -> char -> int list -> int list` telle que `etape a x le` renvoie la liste `le_res` des états `i` pour lesquels il existe un état `j` de `le` tel que `(j, x, i)` soit une transition de `a`. ◁

▷ **Question 8.** Écrire une fonction `reconnait_larg : automate -> string -> bool` telle que `reconnait_larg` renvoie `true` si et seulement si `u` est reconnu par l'automate `a` : on utilisera pour cela l'algorithme suggéré par la question 6. ◁

3 Mot reconnu par une expression régulière

▷ **Question 9.** Définissez par induction structurelle une fonction des expressions régulières qui détermine si un langage est vide. ◁

On définit le type des expressions régulières :

▷ **Question 10.** Écrire la fonction `vide : expr->bool.` qui calcule la fonction précédente. ◁

```
type expr = Vide
 | Epsilon
 | Lettre of char
 | Union of expr * expr
 | Concat of expr * expr
 | Etoile of expr ;;
```

▷ **Question 11.** Écrire une fonction `a_eps : expr -> bool` telle que `a_eps e` s'évalue à vrai si et seulement si le langage dénoté par l'expression régulière représentée par `e` contient le mot vide ϵ. ◁

▷ **Question 12.** On considère l'expression régulière $e_1 = ab^*a$. Définir cette expression régulière en CAML. Determiner le langage $L(e_1)$ dénoté par cette expression régulière. ◁

Si $L \subset \Sigma^*$ est un langage, son résiduel à gauche (on dira simplement résiduel) pour un mot $u \in \Sigma$ est le langage $u^{-1}L = \{v \in \Sigma, uv \in L\}$. Autrement dit c'est le langage des mots v qui peuvent compléter le mot u pour obtenir un mot de L.

▷ **Question 13.** Montrer que $u \in L$ si et seulement si $\epsilon \in u^{-1}L$. ◁

On va maintenant chercher à écrire une fonction qui détermine si un mot u est dans un langage L. Pour un mot $u \in \Sigma^*$ l'objectif est donc de calculer $u^{-1}L$ et de savoir si ϵ est dans ce langage pour savoir si $u \in L$.

▷ **Question 14.** Pour $u = av$, avec $u, v \in \Sigma^*$ et $a \in \Sigma$, montrer que $u^{-1}L = v^{-1}(a^{-1}L)$. ◁

Il suffit donc de lire les lettres de u une par une et de déterminer successivement les langages résiduels pour aboutir à $u^{-1}L$.
Par exemple, déterminons si aba appartient à $ab^*a = L(e_1)$.

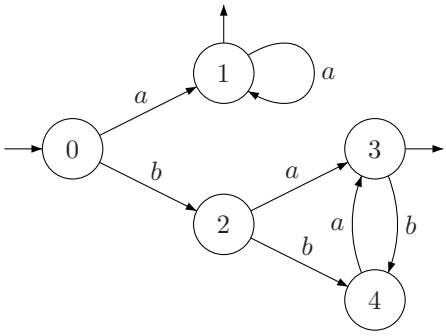
▷ **Question 15.**
Déterminer $a^{-1}L(e_1)$ ainsi qu'une expression régulière e_2 qui dénote ce langage. ◁

▷ **Question 16.** Déterminer $(ab)^{-1}L = b^{-1}L(e_2)$ ainsi qu'une expression regulière e_3 qui dénote ce langage. ◁

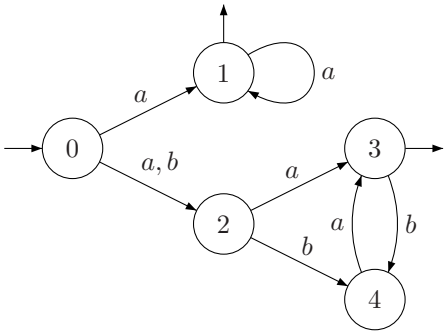
▷ **Question 17.** Écrire une fonction `residuel : char -> expr -> expr` qui étant donné un caractère `a` et une expression régulière `e` s'évalue en une expression régulière `ê` qui dénote le langage résiduel $L(\hat{e}) = a^{-1}L(e)$. ◁

▷ **Question 18.** Écrire une fonction `appartient : char list -> expr -> bool` qui vérifie si un mot représenté par une liste de caractères appartient au langage dénoté par une expression régulière. ◁

▷ **Question 19.** Écrire de même une fonction `appartient_bis : string -> expr -> bool` qui a le même comportement que la fonction précédente mais avec une représentation des mots par le type string de CAML. ◁



Automate 1



Automate 2