TP 1 : Expressions rationnelles et automates

24 septembre

Ce TP aborde certaines questions sur les langages reconnaissables : trouver un mot de longueur minimale dans deux situations : si le langage est représenté par une expression régulière mais aussi dans le cas où on connait un automate le reconnaissant. On construira aussi une fonction permettant de déterminer si un mot appartient au langage dénoté par une expression régulière.

1 Expressions régulières

▷ **Question 1.** Définissez par induction structurelle une fonction des expressions rationnelles qui détermine si un langage est vide. ◁

On définit le type des expressions rationnelles :

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

▷ **Question 2.** Écrire la fonction `vide : expr->bool`. qui calcule la fonction précédente. ◁

▷ **Question 3.** 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 4.** Écrire une fonction `est_eps : expr -> bool` telle que l'appel `est_eps e` s'évalue à vrai si et seulement si le langage dénoté par l'expression régulière représentée par `e` est exactement $\{\epsilon\}$, le langage formé de l'unique mot vide. ◁

▷ **Question 5.** Définissez par induction structurelle une fonction des expressions rationnelles vers les entiers qui calcule la longueur du plus court mot de L si le langage n'est pas vide, et ∞ sinon. ◁

▷ **Question 6.** Écrire la fonction `longueur_mot_min` qui calcule la fonction précédente. Elle retournera un objet de type `int option`. ◁

▷ **Question 7.** 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 8.** 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 9.** 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 10.**

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

▷ **Question 11.** Déterminer $(ab)^{-1}L = b^{-1}L(e_2)$ ainsi qu'une expression regulière e_3 qui dénote ce langage. ◁

▷ **Question 12.** 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 13.** 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 14.** 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. ◁

2 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.

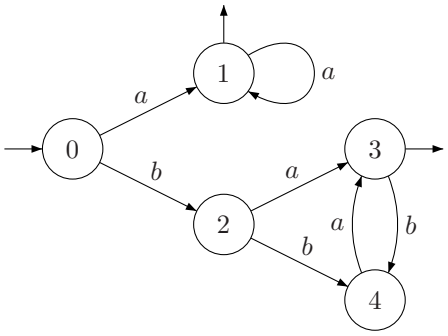
```
type automate =
  { taille :  int ;
    initial :  int ;
    transitions :  (char * int) list array ;
    final :  bool array } ;;
```

▷ **Question 15.** 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 16.** 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 ci-dessous, et vérifier sur les exemples aa, aba et bab que la fonction `calcul_det` est correcte. ◁



▷ **Question 17.** Ecrire une fonction qui fait un parcours en profondeur de l'automate depuis l'état initial et se contente d'afficher les noms des états parcourus. ◁

▷ **Question 18.** Écrire une fonction `accessible` qui supprime les états inaccessibles d'un automate. Pour cela, on doit renuméroter les états, on pourra maintenir deux tableaux `tab_conversion` et `tab_inversion` qui

gèrent la correspondance entre nouveaux et anciens états. On adaptera le code de la question précédente pour parcourir les sommets depuis l'état initial en les renumérotant. ◁

▷ **Question 19.** Écrire une fonction `est_vide_auto` qui détermine si le langage de l'automate est vide. Écrire une fonction `longueur_mot_min_auto` qui calcule la longueur du mot minimal accepté par l'automate. Elle retournera un objet de type `int option`. ◁

▷ **Question 20.** Écrire une fonction `langage_auto` qui retourne la liste de tous les mots de taille minimale reconnus par l'automate. Quelle est sa complexité? ◁