Université de Versailles – Saint-Quentin-en-Yvelines 2019-2020
Licence 2 Informatique

Projet – Évaluation d'expressions régulière

Version du 24 mars 2020

Xavier Badin de Montjoye– xavier.badin-de-montjoye@ens-lyon.fr
Loric Duhazé – loric.duhaze@ens.uvsq.fr
Franck Quessette – Franck.Quessette@uvsq.fr
Guillaume Scerri – Guillaume.Scerri@uvsq.fr
Yann Strozecki – Yann.Strozecki@uvsq.fr

Le but de ce projet est d'évaluer des expressions régulières. Il est découpé en trois parties dont voici la première. Une correction de la première partie sera donnée en même temps que le sujet de la deuxième partie.

1 Opération sur les automates

Une archive nommée `IN406_Projet_Partie_1.zip` vous est fournie sur MOODLE. Elle contient le dossier `IN406_Projet_Partie_1` contenant un canevas du code et en particulier la structure de donnée représentant un automate ainsi que les fonctions de manipulation, que vous trouverez décrite dans le fichier `automate.h`.

Le premier type important est le type `TRANSITION`, qui représente une transition. Les transitions sont organisées en liste. L'alphabet est ici toujours de taille 26 et contient les lettres de `a` à `z` en minuscules. Les lettres de l'alphabet sont codés par le caractère correspondant dans les transitions.

```
struct transition {
    char car;//caractère étiquetant la transition. -1 si c'est une epsilon transition
    int arr;//état d'arrivée de la transition
    struct transition *suiv;
};

typedef struct transition *TRANSITION ;
```

Le deuxième type est `AUTOMATE`. Les transitions de l'automate sont représentées par un tableau indexé par les états de listes de transitions.

```
typedef struct{
    int sigma; // Le nombre de lettre de l'alphabet, les lettres de l'alphabet étant les minuscules.
    int Q; // Le nombre d'états qui sont numérotés de 0 à Q-1. 0 étant l'état initial
    int *F; // Un tableau de taille Q qui indique pour chaque état s'il est final ou non
    TRANSITION *T; //Un tableau de taille Q donnant pour chaque état
    //la liste des transitions depuis cet état
}AUTOMATE;
```

Les fonctions de manipulation d'automates fournies sont les suivantes. Vous pouvez aussi travailler directement sur les structures de données automate et transition.

```
TRANSITION copie_liste(TRANSITION T, int decalage, int conserve_epsilon);
//copie une liste de transitions, en ajoutant décalage au numero de l'état d'arrivée
//si conserve_epsilon == 0, ne conserve pas les epsilon transitions

AUTOMATE automate_creer (int Q);
```

```
// Création d'un automate de taille Q, sans états finaux et sans transitions

AUTOMATE automate_copier(AUTOMATE A, int conserve_epsilon);//créé une copie fraiche de l'automate A
//si conserve_epsilon == 0, ne conserve pas les epsilon transitions

void automate_ajouter_transition (AUTOMATE A, int dep, char car, int arr);
// Ajoute la transition (dep,car,arr) à l'automate (ne fait rien si elle existe)

void automate_ajouter_final (AUTOMATE A, int q); //ajoute l'état q aux états finaux
void automate_supprimer_final (AUTOMATE A, int q); //supprime l'état q des états finaux
void automate_liberer_memoire(AUTOMATE A);// libère la mémoire allouée dynamiquement pour l'automate
void automate_ecrire (AUTOMATE A, char* nomfic); // Ecrit un automate dans un fichier sous la forme :
Q F T
F_1 F_2 F_3 ... F_F
dep_1 car_1 arr_1
dep_2 car_2 arr_2
...
dep_T car_T arr_T
Q : nombre d'états
F : nombre d'états finaux
T : nombres de transitions
F_i : les numéros des états finaux
dep_j car_j arr_J : une transition

AUTOMATE automate_lire (char* nomfic);
// Lit un automate depuis un fichier sous la forme précédente
AUTOMATE automate_supprimer_epsilon(AUTOMATE A);
//renvoie un automate sans epsilon transition qui reconnait le même langage que A
```

On va implémenter l'ensemble des automates et opérations sur les automates qui permettent de reconnaître une expression régulière comme nous l'avons vu en TD. Les prototypes des fonctions à implémenter se trouvent dans automate.h.

Question 1 : Donner une fonction qui renvoie un automate reconnaissant le langage $\{\epsilon\}$.

Question 2 : Donner une fonction qui renvoie un automate reconnaissant le langage $\{\alpha\}$ pour $\alpha \in \Sigma$.

Question 3 : Étant donné deux automates A et B, donner une fonction qui renvoie un automate reconnaissant le langage $L(A) + L(B)$ (disjonction).

Question 4 : Étant donné deux automates A et B, donner une fonction qui renvoie un automate reconnaissant le langage $L(A).L(B)$ (concaténation).

Question 5 : Étant donné un automate A, donner une fonction qui renvoie un automate reconnaissant le langage $L(A)^*$ (étoile de Kleene).

Question 6 : Étant donné un automate A, donner une fonction qui renvoie un automate **déterministe** reconnaissant le même langage que A (déterminisation).

Question 7 : [Optionnel] Étant donné un automate A, donner une fonction qui renvoie un automate **minimal** reconnaissant le même langage que A (minimisation).

2 Modalités pratiques

Merci de respecter les consignes suivantes :

- le projet est à faire en seul ou en binôme (en restant chacun chez soi). Si vous travaillez en binôme ne remettez qu'un seul projet. ;
- le projet est à rendre dans MOODLE, la première partie doit être rendue au plus tard le

  dimanche 12 avril 2020, 23h59

- vous devez déposer un fichier `XY_NOM_Prenom.zip` ou `XY_NOM_Prenom-NOM2_Prenom2.zip` qui est le zip du dossier `XY_NOM_Prenom.zip` ou `XY_NOM_Prenom-NOM2_Prenom2` contenant les fichiers donnés avec l'énoncé et complétés par vos soins.
  Le plus simple est de dès le départ renommer le dossier **IN406_Projet_Partie_1**.
  `XY` sont les initiales (`XB`, `LD`, `FQ`, `GS`, `YS`) du chargé de TD de `NOM`. Pour être évalué, votre travail doit compiler et s'exécuter quand on tape make dans le terminal. Si le nom de fichier remis sur moodle ne respecte pas ces règles, le projet ne sera pas évalué.

Le retard de la remise du projet entraine 1 point de moins par heure de retard.