I53 - Compilation et théorie des langages
- TP 3 -

Licence 3 - 2023/2024

Abstract

L'objectif de ce TP est de concevoir un programme similaire à `grep` capable de lire une expression régulière dans l'entrée standard et une chaîne de caractères quelconque et d'afficher si oui ou non la chaîne appartient au langage dénoté par l'expression régulière (en construisant un automate acceptant le langage correspondant).

Introduction

On se propose d'implanter un simulateur d'automates finis (AFD et AFN). Le premier travail consiste à compléter les fichiers `afd.c` et `afn.c` permettant la manipulation des AFD et des AFN. Les structures de données sont celles choisies en cours. Par convention les automates ne gèrent que les alphabets contenant des lettres minuscules. De plus le symbole & est réserver pour représenter le caractère spécial ε. Les structures des automates sont données par les structures C suivantes.

```c
#define AFD_ASCII_SHIFT 97
#define AFD_NB_SYMB 26

struct AFD{
  int Q, q0;
  int *F;
  char * sigma;
  int **delta;
};

typedef struct AFD * AFD;
```

```c
#define AFN_ASCII_SHIFT 96
#define AFN_NB_SYMB 27

struct AFN{
  int Q;
  int *I,*F;
  char *sigma;
  int ***delta;
};

typedef struct AFN * AFN;
```

- Le champ `Q` représente le plus grand état de l'automate (les états sont donc numérotés de 1 à `Q`).
- Le champ `q0` représente l'état initial d'un AFD. Les champs `I` et `F` représente les ensembles d'états initiaux et finaux.
- Dans ce TP les ensembles sont représentés par des tableaux d'entiers se terminant par -1 (le tableau `{-1}` représente donc l'ensemble vide).
- Le champ `sigma` représente l'alphabet de l'automate sous forme de chaîne de caractères.
- Le champ `delta` représente la fonction de transition de l'automate:
  - pour les AFD il s'agit d'un tableau d'entiers a deux dimensions. Étant donné un état q et caractère σ on accède à la transition $\delta(q, \sigma)$ par l'expression `delta[q][sigma - AFD_ASCII_SHIFT]` qui est soit un numéro d'état soit -1 si la case est vide.
  - pour les AFN il s'agit d'un tableau a deux dimensions dont les éléments sont des ensembles d'états. Étant donné un état q et caractère σ on accède à l'ensemble des transitions possibles $\delta(q, \sigma)$ par l'expression `delta[q][sigma - AFN_ASCII_SHIFT]` qui est un tableau d'entiers se terminant par -1.

1 Simulation d'automates finis

1. Récupérer les fichiers `afd.[ch], afn.[ch], af.c, makefile`, compiler et tester le programme.
2. Compléter la fonction `simuler` du fichier `afd.c` permettant de simuler l'action d'un AFD sur une chaine de caractères. La fonction renvoie 1 si la chaîne est acceptée par l'automate et 0 sinon.
3. Compléter les fonctions permettant de gérer les ensembles sous formes de tableaux: `ajoute_etat, cherche_etat, set_union, set_intersection`.
4. Compléter les fonctions `afn_epsilon_fermeture` et `afn_determinisation`.

2 Compilation d'expressions régulières en automate

Le but de cette partie est de pouvoir transformer une expression régulière en AFN. On utilisera la grammaire des expressions régulières suivantes:

Exreg	→ *Exreg* | *Exreg*	(union)
	| *Exreg* . *Exreg*	(concaténation)
	| *Exreg* $*$	(fermeture de Kleene)
	| `Char`	
	| `(` *Exreg* `)`	

Ici le terminal `Char` représente n'importe quel caractère minuscule. En particulier on impose ici à tous les automates de travailler sur le même alphabet à savoir `&abc...xyz`

Exemple d'utilisation:

```
$ ./mygrep
usage: ./mygrep <exreg> <chaine>
$ ./mygrep '(a|b).(a|c).b*' 'acbbbbb'
acceptee
$ ./mygrep '(a|b).(a|c).b*' 'bbbbbb'
rejetee
```

1. Écrire une fonction `AFN afn_char(char c, char *Sigma)` qui contruit un AFN acceptant le langage constitué du seul symbole `c`.
2. Écrire les trois fonctions suivantes

   ```
   AFN afn_union(AFN A, AFN B);
   AFN afn_concat(AFN A, AFN B);
   AFN afn_kleene(AFN A);
   ```

   qui construisent respectivement les automates reconnaissant l'union, la concaténation et la fermeture de Kleene des langages acceptés par les automates A et B.
3. Construire une grammaire décrivant la structure des expressions régulières sur l'alphabet complet (ex: `a.(f.x|d.b)*.z.x*.y`).
4. Dans trois nouveaux fichiers (`compregex.h`, `compregex.c` et `mygrep.c` qui produiront un programme `mygrep`), écrire un traducteur dirigé par la syntaxe (sur le modèle du TP 2) qui construit un AFN reconnaissant le langage décrit par une expression régulière lue dans l'entrée standard.
5. supprimer le symbole point (.) de la grammaire pour représenter la concaténation (ainsi on pourra écrire `abc` au lieu de `a.b.c`);
6. ajouter les constructions suivantes pour simplifier l'écriture des expressions:
   (a) $\alpha+$ qui représente la répetition au moins une fois de l'expression régulière α.
   (b) ? qui représente n'importe quel caractère.
   (c) $[c_1 c_2 \dots c_n]$ qui représente exactement un des caractères c_i.
   (d) $[\hat{}c_1 c_2 \dots c_n]$ qui représente tous les caractères exceptés les c_i.
   (e) $\alpha\{n\}$ qui dénote la répétition exactement n fois de l'expression régulière α.