I53- Compilation et théorie des langages
- TP 4 -

Licence 3 - 2022/2023

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).

1 Simulation d'automates finis

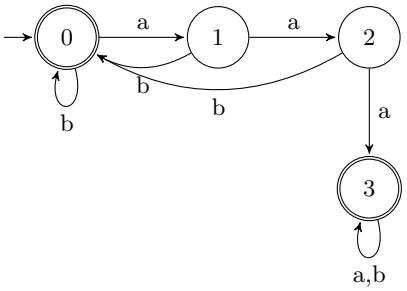
On se propose d'implanter un simulateur d'automates finis (AFD et AFN). Le premier travail consiste à compléter les fichiers `afd.c` et `afn.c` permettant la manipulation des AFD et des AFN. Les structures de données sont celles choisies en cours. Par convention les automates ne gèrent que les alphabets contenant des symboles dont le comme ascii est compris en 38 et 127. De plus le symbole & est réserver pour représenter le caractère spécial ε.

```c
#define MAX_SYMBOLES 90

struct AFD{
  int Q,q0,lenF,lenSigma;
  int *F;
  char * Sigma;
  char dico[MAX_SYMBOLES];
  int **delta;
};

typedef struct AFD * AFD;
```

```c
struct AFN{
  int Q,lenI,lenF,lenSigma;
  int *I,*F;
  char * Sigma;
  char dico[MAX_SYMBOLES];
  int ***delta;
};

typedef struct AFN * AFN;
```

1. Récupérer les fichiers `afd.[ch], afn.[ch], af.c, makefile`, compiler et tester le programme.
2. Compléter les fichier `afd.c` et `afn.c` pour pouvoir initialiser des AF à partir d'un fichier et simuler le fonctionnement d'un automate sur une chaine de caractères.

   Par exemple l'AFN suivant pourra être représenté par l'automate ci-dessous.




```
3
0
2
0 3
ab
0 a 1
0 b 0
1 a 2
1 b 0
2 a 3
2 b 0
3 a 3
3 b 3
```

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 $\rightarrow$ *Exreg* + *Exreg* (union)
| *Exreg* . *Exreg* (concaténation)
| *Exreg* $*$ (fermeture de Kleene)
| `Char`
| *(Exreg)*

Ici le terminal `Char` représente n'importe quel caractère ascii alphanumérique. En particulier on impose ici à tous les automates de travailler sur le même alphabet à savoir `&abc...xyzABC...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 `void afn_char(afn *C, char c)` qui contruit un AFN acceptant le langage constitué du seul symbole `c`.
2. Écrire les trois fonctions suivantes

   ```
   void afn_union(afn *C, afn A, afn B);
   void afn_concat(afn *C, afn A, afn B);
   void afn_kleene(afn *C, 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. (facultatif) On pourra améliorer le programme comme suit:
   - supprimer le symbole point (.) de la grammaire pour représenter la concaténation (ainsi on pourra écrire `abc` au lieu de `a.b.c`;
   - $[c_1 c_2 ... c_n]$ qui réprésente exactement un des caractères c_i.
   - $Exreg\{n\}$ qui dénote la répétition exactement n fois d'une expression régulière.
   - implanter l'algorithme de déterminisation des AFN afin d'utiliser la fonction de simulation des AFD à la place de celles des AFN.