Examen de Compilation
corrigé

Licence Sciences Pour Ingénieur

13 Mai 2014

Le sujet est composé d'exercices indépendants. Aucun document n'est autorisé. Durée de l'épreuve : 0x78 minutes. La présentation de la copie entre en compte dans la note finale.

1 Expression régulière

L'extrait de la page `regex` du manuel (`man 7 regex`) signale la possibilité de décrire des motifs avec des "références arrières".

1. Décrire le langage des mots correspondants au motif $(a+)b\backslash 1c$. Il s'agit du langage

$$\{a^n b a^n c \mid n \geq 1\}$$

2. Est-il légitime de parler d'expressions régulières ? Non car le langage n'est pas rationnel. En effet, les mots $a^n b$ sont deux à deux inéquivalents au sens de Nérode i.e. une infinité de résiduels.

```
REGEX(7)    Manuel du programmeur Linux
NOM         regex - Expressions rationnelles POSIX.2
DESCRIPTION
  la référence arrière: «\» suivi d'un chiffre décimal non-nul n corres-
-pond à la même séquence de caractères que ceux mis en correspondance avec
la n-ième sous-expression entre parenthèses. (les sous-expressions sont
numérotées par leurs parenthèses ouvrantes, de gauche à droite),
ainsi «([bc])\1» correspond à « bb » ou « cc » mais pas à « bc ».
```

2 Théorie des langages

Dans le cours, nous avons étudié la stabilité par intersection de deux classes de langages. Rappeler en quelques mots pourquoi :

1. l'intersection de deux langages rationels est rationel ; L'approche automatique montre que la classe rationnelle est stable par union est stable complément donc par intersection.
2. en général, l'intersection de deux langages algébriques n'est pas algébrique. Nous avons vu que $\{a^n b^n c^m \mid n, m \geq 0\}$ et $\{a^m b^n c^n \mid n, m \geq 0\}$ sont algébriques mais que l'intersection $\{a^n b^n c^n \mid n \geq 0\}$ ne l'est pas.

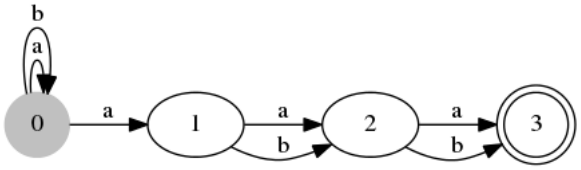

FIG. 1 – U_3 un automate non déterministe à 4 états.

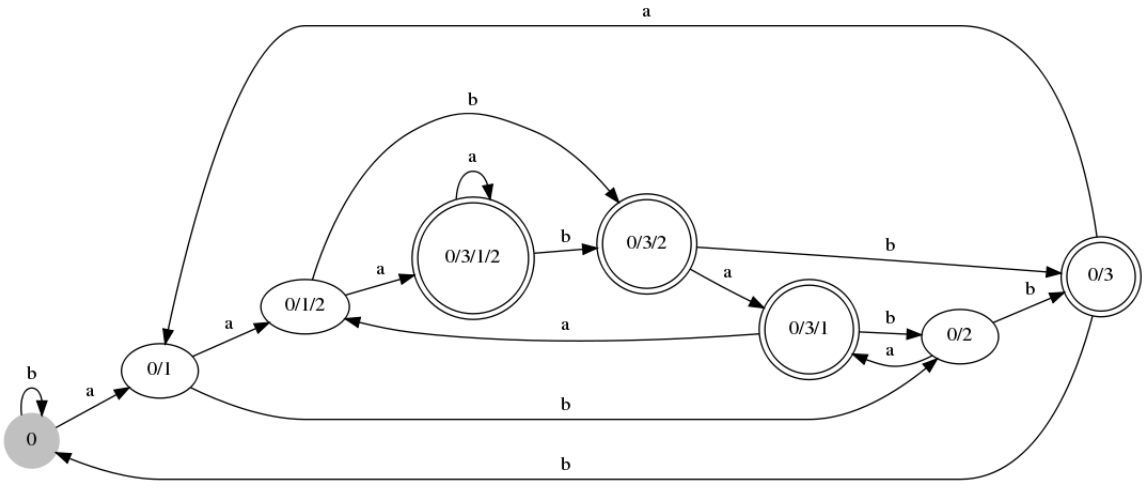

FIG. 2 – déterminisation de U_3.

TAB. 1 – opérations de la machine

STO r	stocke la valeur de ACC dans r.
VAL v	charge la valeur v dans l'accumulateur.
MEM m	charge le contenu de la mémoire m.
ADD r	ACC := ACC + registre[r].
MUL r	ACC := ACC + registre[r].

3 Automate

On considère $A = \{a, b\}$.

1. Quel est le langage reconnu par U_3 ? Il reconnait les mots qui terminent par axy, $x, y \in A$.
2. Déterminiser l'automate U_3.
3. Construire un automate non déterministe à $n + 2$ états qui reconnait A^*aA^n.

$$\to I \overset{a,b}{\to} 0 \overset{a}{\to} 1 \overset{a,b}{\to} 2 \overset{a,b}{\to} \cdots \overset{a,b}{\to} \cdots \overset{a,b}{\to} n \to$$

4. Montrer que les résiduels des mots de moins $n - 1$ lettres sont distincts. Soit deux mots distinct u et v de $n - 1$ lettres. Notons π le plus long prefixe commun.

$$u = \pi ax, \quad v = \pi by, \qquad |x| = |y| = r < n - 1.$$

et par exemple a^{n-r} est dans le résiduel de u mais pas de v.

5. Quelle conclusion en tirer sur l'algorithme de déterminisation ? Il existe un automate non déterministe à $\Theta(n)$ états dont la déterminisation passe par $\Theta(2^n)$ états. La complexité (temps/espace) d'un algorithme de déterminisation est exponentielle dans le pire des cas.

4 flex-bison

On considère une machine fictive possèdant 26 mémoires (A,B,C,...) et 16 registres (0,1,...), un accumulateur ACC et des opérations : STO, MEM, VAL, ADD, MUL.

Un fragment d'analyseur syntaxique `mini.y` est fourni. Il s'agit de le compléter pour obtenir l'ébauche d'une commande `mini.exe`.

1. Préciser la nature de la commande `mini.exe` : compilateur, interpréte ? La commande produit du code mnémonique sans faire de calcul : c'est un compilateur.
2. Ecrire un analyseur lexical adéquat en `flex`.

```
1  %{
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include "mini.h"
5  %}
6
7  %%
8  [0-9]+   yylval = atoi( yytext ); return NB;
9  [A-Z]    yylval = *yytext ; return MEM;
10 "+"      return PLUS;
11 ";"      return SEP;
12 %%
```

3. Quelle compilation permet d'obtenir `mini.exe` ?

```
mini.exe : mini.y mini.l
        bison -dv mini.y -o mini.c
        flex  mini.l
        gcc -Wall lex.yy.c  mini.c  -o mini.exe -lfl
```

4. Quel est le résultat de

```
echo -n 'A+B+C+7;' | ./mini.exe
```

```
014]$ make min
echo -n 'A+B+C+7;' | ./mini.exe
MEM A;
STO 0;
MEM B;
ADD 0;
STO 0;
MEM C;
ADD 0;
STO 0;
VAL 7;
ADD 0;
```

La commande produit du code pour réaliser les calculs dans l'accumulateur. Produire le code de $E + E'$ c'est produire le code de E, stocker le résultat dans un regitre temporaire r , produire le code de E' et ajouter le contenu de r.

5. Comment modifier l'analyseur syntaxique pour gérer la multiplication ? Il faut d'introduire un nouveau symbole pour la multiplication au niveau lexical

```
11 "*"      return FOIS;
```

Il faut définir le token au niveau syntaxique

```
18 %token  MEM NB SEP AFF
19 %left   PLUS
20 %left   FOIS
```

et l'action sémantique pour produire le code

```
27 EXP :  EXP FOIS {
28           $1 = reg( );
29           printf("\nSTO %d;", $1);
30        } EXP {
31        printf("\nMUL %d;", $1 );
32        use[ $1 ] = 0;
33        }
```

6. Compléter l'analyseur syntaxique pour gérer une affectation. On définit le symole terminal aux niveaux lexical et sémantique. Les action sémantiques sont :

```
25 INSTR : EXP | AFFECT
```

```
47 AFFECT : MEM AFF EXP {
48        printf("\nSTO %c;", $1 );
49     }
```