Examen de Compilation

Licence Sciences Pour Ingénieur

14 Mai 2015

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

```
1  #!/bin/bash
2  # stupid prime generator
3  reg='^(xx+)\1+$' ; y=xx ; cpt=
       $1
4  while [[ cpt -gt 0 ]] ; do
5      if [[ ! $y =~ $reg ]]; then
6          echo -n ' '${#y}
7       let cpt--
8      fi
9      y=x$y
10 done
```

Le script utilise une commande `grep` basée sur une expression rationnelle étendue contenant une référence arrière. Une fonctionalité décrite dans `man 7 regex`.

```
REGEX(7)    Manuel du programmeur
    Linux
NOM         regex - Expressions
    rationnelles POSIX.2
DESCRIPTION
  la référence arrière: «\» suivi
d'un chiffre décimal non-nul n
correspond 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 ».
```

1. Que représentent les caractères spéciaux $\wedge$ et \$ dans une expression régulière ? Ils servent à mettre en correspondance avec le début et la fin de ligne.
2. Décrire le langage L des mots correspondants au motif $(a+)b\backslash 1c$.

$$\{a^n b a^n c \mid n > 0\}$$

3. Préciser la nature, régulier, algébrique, de L. Le langage est algébrique engendré par la grammaire

$$S \mapsto Tc; T \mapsto aTa; T \mapsto b$$

et non régulier car il possède une infinité de résiduel. En effet, le résiduel de $a^n b$ est $a^n c$.

4. Commenter le résultat de la commande ci-dessous. Le script utilise une référence arrière (vue en TDs) pour déterminer le langage des mots

$$\{x^p \mid p > 2 \quad \text{premier}\}.$$

En effet, si x^n correspond à l'expression $((xx+)\backslash 1)+$ cela veut dire qu'il existe un entier $k > 1$ et un entier $r > 1$ tel que :

$$x^n = \underbrace{x^k x^k \cdots x^k}_{r \text{ fois}} = x^{kr} \Longrightarrow \quad \text{i.e. } n \text{ est composé}$$

Nous avons vu en cours que ce langage n'est pas régulier. Le script illustre la différence entre les REs et les expressions rationnelles. Le temps d'exécution montre que le procédé est loin d'être pas performant !

```
exam>  time ./prime.sh 10
 2 3 5 7 11 13 17 19 23 29
real 0m0.209s
user 0m0.187s
sys 0m0.002s
```

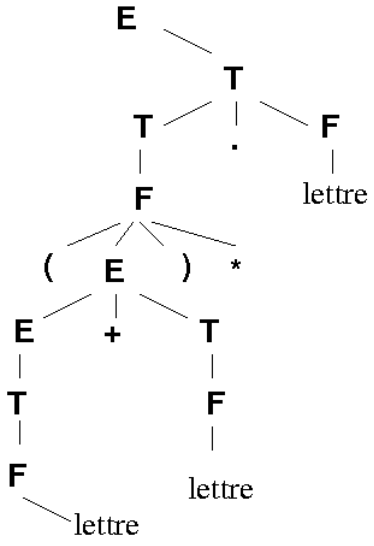
2 Théorie des langages

On note X le langage des expressions rationnelles basées sur les opérateurs union, produit, étoile et les lettres.

1. Le langage X est-il : régulier, rationnel, algébrique ? Le langage des expressions rationnelles est algébrique, non rationnel et non régulier.
2. Préciser la nature des opérateurs. Deux opérateurs binaires associatifs à gauche (usage), l'étoile est de priorité maximale, l'union de priorité minimale.
3. Ecrire une grammaire non ambigüe.

```
E -> E + T  | T
T -> T . F  | F
F -> ( E )* | ( E ) | lettre
```

4. Donner l'arbre syntaxique de $(a + b)^*.c$.



3 Automate

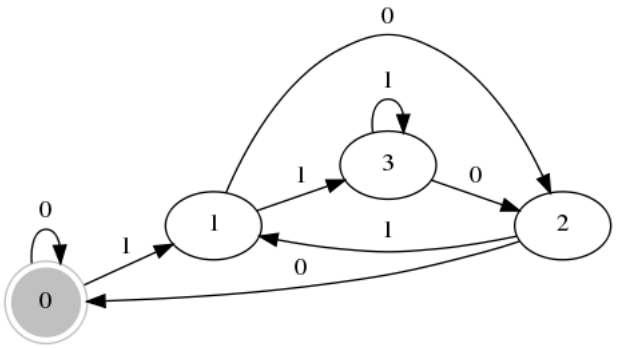
On considère l'alphabet $A = \{0, 1\}$. Pour un entier, n positif, on note X_n, l'automate ayant $\{0, 1, \ldots, n-1\}$ pour ensemble d'états, avec 0 pour unique état initial et final, et une fonction de transition δ définie par :

$$\forall q \in \{0, n-1\}, \quad \forall c \in \{0, 1\}, \qquad \delta(q, c) = (q * 2 + c) \mod n.$$

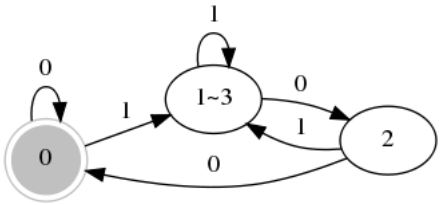
Pour tout langage X, on définit le langage des suffixes de X :

$$s(X) = \{y \in A^* \mid \exists x \in A^*, \quad xy \in X\}.$$

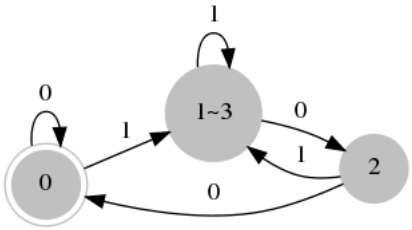
1. Démontrer que si X est régulier alors $s(X)$ est régulier. Il suffit de modifier un automate reconnaissant X, en considérerant tous les états utiles (accessibles et co-accessibles) comme des états initiaux.
2. Dessiner une représentation sagittale de X_4.



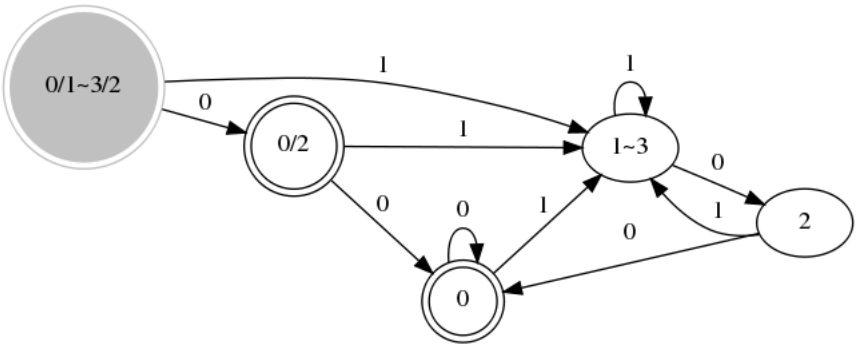
3. Minimiser X_4.



4. Déduire un automate non déterministe pour le langage $s(X_4)$



5. Déterminiser l'automate précédent.



6. Donner une expression rationnelle pour $s(X_4)$. On peut minimiser le dernier automate, ou bien faire un raisonnement direct, pour obtenir :

$$\sigma(X_4) = X_4 = \{0,1\}^*00 + 0 + \epsilon$$

4 flex-bison

On compile les sources `flex` et `bison` ci-dessous en un exécutable `rat.exe`. Représenter l'arbre syntaxique et l'image réalisés par la commande :

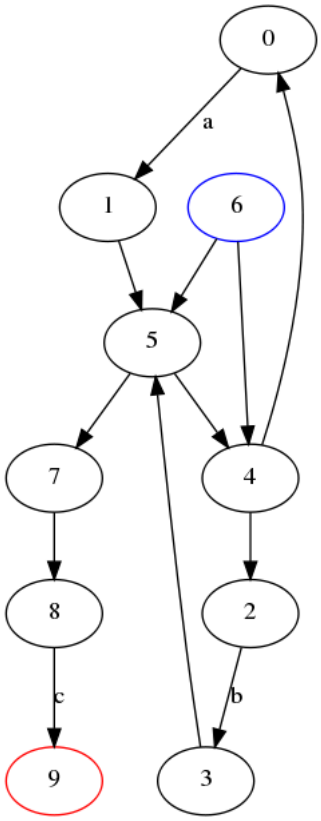
```
exam> echo '(a+b)*.c' | ./rat.exe | dot -Tpng > rat.png
```

L'analyseur construit l'arbre d'une expression rationnelle pour la grammaire :

```
REG  ->  EXP
EXP  ->  EXP plus EXP | EXP dot EXP | EXP star | lettre
```

```
REG  ->  EXP  ->  EXP dot EXP
                     /       \
                ( EXP ) star    c
                   |
              EXP plus EXP
              |        |
              a        b
```

Les actions sémantiques forment un automate fini non déterministe par la méthode de Thompson. Le graphe est représenté au format `dot`. L'état initial est en bleu, l'état final en rouge.




```
%{
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#define  ORG(x) ( ( x >> 8  ) )
#define  END(x) ( ( x & 255 ) )
int yylex( void );
int yyerror( char *x );
int mknodes( void );
extern int lino;
%}
%token  ALPHA LEFT RIGHT
%left   PLUS
%left   DOT
%left   STAR
%%
REG :   EXP  {
 printf("%d [color=blue]\n", ORG($1) );
 printf("%d [color=red]\n", END($1) );
    }
EXP :   ALPHA
    |   LEFT EXP RIGHT {
        $$ = $2;
        }
    |   EXP DOT EXP {
```

```
26          $$ = ( ORG($1) << 8 ) + END($3);
27          printf("%d->%d\n", END($1) , ORG($3) );
28          }
29      |   EXP PLUS EXP {
30          $$ = mknodes();
31          printf("%d->%d\n", ORG($$) , ORG($1) );
32          printf("%d->%d\n", ORG($$) , ORG($3) );
33          printf("%d->%d\n", END($1) , END($$) );
34          printf("%d->%d\n", END($3) , END($$) );
35          }
36      |   EXP STAR {
37          $$ = mknodes();
38          printf("%d->%d\n", END($1) , ORG($1) );
39          printf("%d->%d\n", ORG($$) , ORG($1) );
40          printf("%d->%d\n", ORG($$) , END($1) );
41          printf("%d->%d\n", END($1),  END($$) );
42          }
43      ;
44  %%
45  int main(void) {
46    printf("digraph {\n");
47    yyparse();
48    printf("}\n");
49    return 0;
50  }
51  int yyerror( char*x) {
52     printf("%s:%d\n", x , lino); return 0;
53  }
```