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
```

```
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 ».
```

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

1. Que représentent les caractères spéciaux $\wedge$ et \$ dans une expression régulière ?
2. Décrire le langage L des mots correspondants au motif $(a+)b\backslash 1c$.
3. Préciser la nature, régulier, algébrique, de L.
4. Commenter le résultat de la commande ci-dessous.

```
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 ?
2. Préciser la nature des opérateurs.
3. Ecrire une grammaire non ambigüe
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.
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 $s(X_4)$.

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
```

```
%{
#include <stdio.h>
#include <stdlib.h>
#include "rat.h"
int x, node = 0, lino = 1;
int mknodes( void ) {
int x, y;
x = node++; y = node++;
return (x << 8) + y;
}
%}
%%
[a-z]       {
            x = yylval = mknodes();
            printf("%d->%d [label=\"%c
    \"]\n", x >> 8, x & 255, *yytext)
    ;
            return ALPHA;
}
"("         return LEFT;
")"         return RIGHT;
"+"         return PLUS;
"."         return DOT;
"*"         return STAR;
\n          lino++;
.           ;
%%
```

```
%{
#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 {
        $$ = $2;
        }
    |   EXP DOT EXP {
        $$ = ( ORG($1) << 8 ) + END($3);
        printf("%d->%d\n", END($1) , ORG($3) );
        }
    |   EXP PLUS EXP {
        $$ = mknodes();
        printf("%d->%d\n", ORG($$) , ORG($1) );
        printf("%d->%d\n", ORG($$) , ORG($3) );
        printf("%d->%d\n", END($1) , END($$) );
        printf("%d->%d\n", END($3) , END($$) );
        }
    |   EXP STAR {
        $$ = mknodes();
        printf("%d->%d\n", END($1) , ORG($1) );
        printf("%d->%d\n", ORG($$) , ORG($1) );
        printf("%d->%d\n", ORG($$) , END($1) );
        printf("%d->%d\n", END($1),  END($$) );
        }
    ;
%%
int main(void) {
  printf("digraph {\n");
  yyparse();
  printf("}\n");
  return 0;
}
int yyerror( char*x) {
   printf("%s:%d\n", x , lino); return 0;
}
```