Examen de Compilation

Licence Sciences Pour Ingénieur

25 juin 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 ? Voir correction sujet 1, 2015.
2. Décrire le langage L des mots correspondants au motif $(a+)\backslash 1$. Il s'agit du langage des mots a^{2n}, $n > 0$.
3. Préciser la nature, régulier, algébrique, de L. Le langage est régulier $(aa)+$ et donc algébrique.
4. Commenter le résultat de la commande ci-dessous. Voir correction sujet 1, 2015.

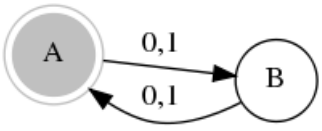
```
exam>  time ./prime.sh 10
 2 3 5 7 11 13 17 19 23 29
real 0m0.209s
user 0m0.187s
sys 0m0.002s
```

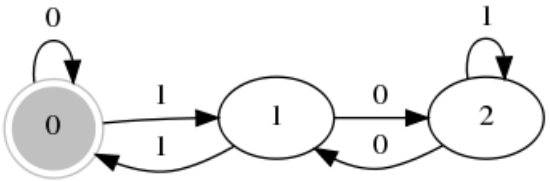
2 Théorie des Langages

On considère l'alphabet $A = \{0, 1\}$. Pour tout langage X, on note $p(X)$ les mots de X qui ont une longueur paire :

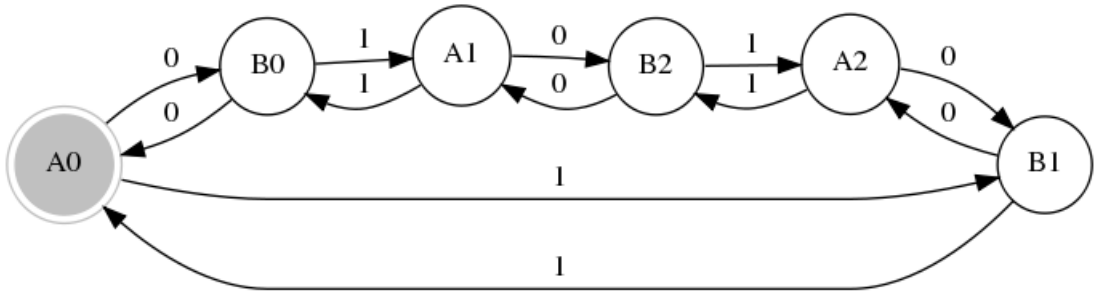
$$p(X) = \{x \in X \mid |x| \equiv 0 \mod 2\}.$$

1. Montrer que si X est rationnel alors $p(X)$ est rationnel. Le langage des mots de longueur paire P est reconnu par un automate à deux états. La classe des langages réguliers est stable par intersection. Il suit que $p(X) = P \cup X$ est régulier.
2. La réciproque est fausse, donner un contre-exemple. Par exemple, le langage $\{x^p \mid p \text{ premier}\}$.
3. Construire l'automate des mots binaires de longueur paire qui représentent un multiple de trois. Il faut construire l'intersection des automates :





soit :



3 gcc-flex

Le langage BF est un langage minimaliste qui s'appuie sur un modèle de machine composé d'un tableau d'octets MEM (initialisés à 0), d'un pointeur PTR sur le tableau et de deux files d'octets pour les entrées et sorties. La machine gère les huit opérations décrites par la table [1]. Un programme en langage BF est une suite d'octets : >, <, +, −, ?, !, [et] qui sont interprétés, les espaces et les caractères situés à droite d'un ';' sont ignorés.

```
%{
#include <stdio.h>
void prologue( void ) {
puts("#include <stdio.h>\ntypedef unsigned char uchar;\n\
uchar mem[1024]={0};\n int ptr=0;\nint main( void ) {\n\
printf(\"#brainf*** v1.0\\n\");");
}
```

TAB. 1 – Les 8 instructions de BFM

symb	mnémo.	ignification	codage
>	INC	incrémente le pointeur	`ptr++`
<	DEC	décrémente le pointeur	`ptr--`
+	ADD	incrémente l'octet pointé	`mem[ptr]++`
–	SUB	décrémente l'octet pointé	`mem[ptr]--`
!	OUT	sortie de l'octet pointé	`putchar(mem[ptr])`
?	IN	entrée d'un octet	`mem[ptr]=getchar()`
[	LOOP	saut si 0 aprés]	`while ( mem[ptr] ) {`
]	RET	retour vers [	`}`

```
void epilogue( void )
{
puts("puts(\"#eoj\\n\");return 0;}");
}
%}

%%
[ ]+ ;
;.*  ;

\n   ;
.    ;
%%

int main( void )
{
prologue();
yylex();
epilogue();
return 0;
}
```

```
++++++++++
[  ;Boucle initiale qui affecte des valeurs utiles au tableau MEM
  >+++++++>++++++++++>+++>+<<<<-
]
  ;A la sortie de la boucle le tableau de la machine contient:
>++!                        ;'H'    = 72  (70  plus 2)
>+!                         ;'e'    = 101 (100 plus 1)
+++++++!                    ;'l'    = 108 (101 plus 7)
!                           ;'l'    = 108
+++!                        ;'o'    = 111 (108 plus 3)
>++!                        ;espace = 32  (30  plus 2)
<<+++++++++++++++!          ;'W'    = 87  (72  plus 15)
>!                          ;'o'    = 111
+++!                        ;'r'    = 114 (111 plus  3)
------!                     ;'l'    = 108 (114 moins 6)
--------!                   ;'d'    = 100 (108 moins 8)
>+!                         ;'!'    = 33  (32  plus  1)
```

```
>!                              ;nouvelle ligne = 10
```

hw.bf

1. Comment compiler `bftoc.l` pour obtenir un exécutable `bftoc.exe` ?

```
$> flex -obf.c bftoc.l
$> gcc -Wall bf.c -o bf.exe -lfl
```

2. Quel est le résultat de `bftoc.exe` appliqué à un fichier (entrée standard) vide ?

```
#include <stdio.h>
typedef unsigned char uchar;
uchar mem[1024]={0};
 int ptr=0;
int main( void ) {
printf("#brainf*** v1.0\n");
puts("#eoj\n");return 0;}
```

3. Compléter le fichier `bftoc.l` pour obtenir un traducteur en `flex` du langage BF vers le langage C.

   Il suffit d'ajouter les règles :

```
">"   puts("ptr++; \n");
"<"   puts("ptr--; \n");
"+"   puts("mem[ptr]++;\n");
"-"   puts("mem[ptr]--;\n");
"!"   puts("putchar(mem[ptr]);\n");
"?"   puts("mem[ptr]=getchar();\n");
"["   puts("while ( mem[ptr] ) {\n");
"]"   puts("};\n");
```

4. Préciser comment (compilations comprises) exécuter le progamme **hw.bf**.

```
$> ./bftoc.exe <hw.bf >hw.c
$> gcc hwx.c
$> ./a.out
```