Université de Caen **Année 2020/2021**
UFR des Sciences **1ère session**
Lundi 29 mars

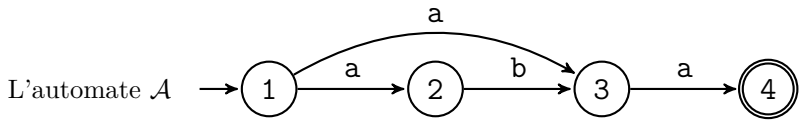
L3 – INF6ACT
Théorie des langages et compilation
durée 2h
Documents autorisés : notes personnelles, diapos du cours.

Chaque candidat doit, en début d'épreuve, porter son nom dans le coin de la copie réservé à cet usage; il le cachettera par collage après la signature de la feuille d'émargement. Sur chacune des copies intercalaires, il portera son numéro de place.

Rendre 2 copies séparées en notant bien le numéro de place :
- l'une qui traite l'exercice I (Automate fini) et l'exercice III (analyse LL)
- l'autre qui traite l'exercice II (Compilation)

Exercice I. Automate Fini — À rendre avec l'exercice III

Question 1. On considère l'automate fini $\mathcal{A}$ décrit par le graphe de transition ci-dessous.

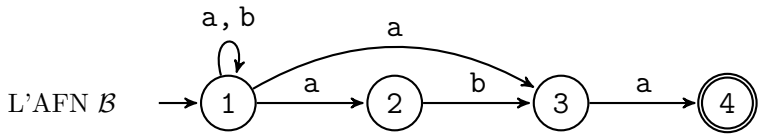


1 . a) Quels sont les deux mots acceptés par l'automate fini $\mathcal{A}$?

1 . b) Donner une expression régulière qui décrit le langage reconnu par l'automate $\mathcal{A}$.

1 . c) L'automate $\mathcal{A}$ est-il déterministe ou non ? Justifier.

Question 2. On modifie l'automate $\mathcal{A}$ en un automate $\mathcal{B}$ avec l'ajout de boucles sur l'état initial.



2 . a) Donner 3 mots de longueur 3 acceptés par $\mathcal{B}$ et en donner la trace d'un calcul acceptant.

2 . b) Donner 2 mots de longueur 4 rejetés par $\mathcal{B}$.

2 . c) Donner une expression régulière du langage reconnu par l'automate $\mathcal{B}$.

Question 3. On veut construire un automate $\mathcal{C}$ version déterministe de l'automate $\mathcal{B}$.

3 . a) Construire la table de transition de l'automate $\mathcal{B}$.

3 . b) Construire un automate déterministe $\mathcal{C}$ qui reconnaît le même langage que $\mathcal{B}$.

Exercice II. Compilation À rendre sur une copie séparée

On souhaite ajouter à notre calculette le support du modulo (entier) à l'aide du symbole `%`. L'opérateur `a % b` renvoie le reste `r` de la division de `a` par `b`. C'est à dire le nombre $0 \leq r < b$ tel que : $a = b * k + r$ où `k` est un entier.

Pour cela, on ajoute à la MVàP l'opcode suivant:

Code	Pile	*sp*	*pc*
MOD	P[sp-1] := P[sp-2] % P[sp-1]	sp - 1	pc+1

Soit le code suivant

```
  PUSHI 0
  JUMP Main
LABEL Main
  PUSHI 45
  STOREG 0
  PUSHI 12
  PUSHG 0
  MUL
  PUSHI 11
  MOD
  PUSHI 5
  PUSHG 0
  SUB
  ADD
  STOREG 0
  PUSHG 0
  WRITE
  POP
  HALT
```

et le résultat de son assemblage

```
 Adr |  Instruction
-----+--------------
   0 | PUSHI 0
   2 | JUMP  4
   4 | PUSHI 45
   6 | STOREG 0
   8 | PUSHI 12
  10 | PUSHG 0
  12 | MUL
  13 | PUSHI 11
  15 | MOD
  16 | PUSHI 5
  18 | PUSHG 0
  20 | SUB
  21 | ADD
  22 | STOREG 0
  24 | PUSHG 0
  26 | WRITE
  27 | POP
  28 | HALT
```

Question 4. Compléter la trace d'exécution suivante.

```
  pc |               |    fp   pile
=====================================================
   0 | PUSHI       0 |     0 [ ] 0
   2 | JUMP        4 |     0 [ 0 ] 1
   4 | PUSHI      45 |     0 [ 0 ] 1
   6 | STOREG      0 |     0 [ 0 45 ] 2
   8 | PUSHI      12 |     0 [ 45 ] 1
  10 | PUSHG       0 |     0 [ 45 12 ] 2
  12 | MUL           |     0 [ 45 12 45 ] 3
  13 | PUSHI      11 |     0 [ 45 540 ] 2
...
```

Question 5. Quelle est la formule calculée par ce code ?

Question 6. Écrire le code MVàP correspondant au code calculette suivant :

```
int b

read(b)

while (b >= 2) {
  b = b - 4
}

write(b)
```

Question 7. Écrire le code MVàP correspondant au code calculette suivant :

```
int b = 15

int f ( x ) {
  return (b * x) % 45
}

write(f(1))
```

On donne la grammaire suivante des expressions :

```
expression
  : '-' expression
  | expression ('*'|'/') expression
  | expression ('+'|'-') expression
  | '(' expression ')'
  | ENTIER
  ;
```

On veux faire en sorte que la priorité de `%` soit la même que celle de $*$ et `/`.

Question 8. Donner l'arbre d'analyse des expressions suivantes, en respectant les priorités:

- `4 + 1 % 4`
- `- 5 % (4 + 45)`
- `4 * 6 + 4 % 8 / 8`

Question 9. Compléter la grammaire pour prendre en compte les modulo (dans cette question, on s'intéresse seulement aux règles syntaxiques sans décrire les actions sémantiques associées). On précisera bien où se situent les lignes ajoutées ou modifiées par rapport au code donné.

Question 10. Indiquer le code d'action à écrire dans la ligne de grammaire concernant le modulo pour générer le code MVàP correspondant. On suposera qu'il n'y a que des entiers.

On supposera que l'expression renvoie une chaîne de caractères :

```
expression returns [ String code]
```

et que l'on dispose de la fonction :

```
private String opCode(String op) {
  if ( op.equals("*") ){return "MUL";}
  else if ( op.equals("/") ){ return "DIV";}
  else if ( op.equals("+") ){ return "ADD";}
  else if ( op.equals("-") ){ return "SUB";}
  else if ( op.equals("\%") ){ return "MOD";}
  }
```

Exercice III. Analyse LL — À rendre avec l'exercice I

Soit la grammaire $\mathcal{G}$ d'axiome S avec quatre symboles terminaux { [,] , x , ; }, et définie par :

$$\begin{cases} \texttt{S} \rightarrow \texttt{[Q} \mid \texttt{x} \\ \texttt{Q} \rightarrow \texttt{S R]} \mid \texttt{]} \\ \texttt{R} \rightarrow \texttt{;\ S R} \mid \varepsilon \end{cases}$$

Question 11. Donner un arbre d'analyse pour le mot suivant :

```
[ x ; x ; x ]
```

On dispose des tables Effacable, Premier et Suivant de la grammaire $\mathcal{G}$:

Symbole	Effacable	Premier	Suivant
S	Non	[, x	$,] , ;
Q	Non	[, x ,]	$,] , ;
R	Oui	;	]

Question 12.

12.a) Indiquer de façon précise comment l'ensemble Premier(Q) est déterminé.

12.b) Expliquer de même comment l'ensemble Suivant(S) est obtenue.

Question 13. Construire la table d'analyse LL(1) de la grammaire $\mathcal{G}$.

Question 14.

14.a) À l'aide de cette table d'analyse LL(1), dérouler l'analyse sur l'entrée

```
[ x ; x ; x ]
```

14.b) De même, dérouler l'analyse LL(1) sur l'entrée

```
[ x ; ]
```

Question 15. On considère la grammaire $\mathcal{H}$ suivante :

$$\begin{cases} \texttt{S} \rightarrow \texttt{[S R]} \mid \texttt{[ ]} \mid \texttt{x} \\ \texttt{R} \rightarrow \texttt{;\ S R} \mid \varepsilon \end{cases}$$

15.a) Pourquoi cette grammaire n'est pas LL(1) ?

15.b) Transformer cette grammaire en une grammaire équivalente pour la rendre LL(1).