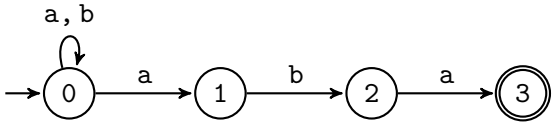
Université de Caen **Année 2018/2019**
UFR des Sciences **2ème session**
Mercredi 3 Juillet

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.

Exercice I. Automate Fini

On se donne l'automate fini suivant :



Question 1. Donner la table de transitions de cet automate.

Question 2. Parmi les mots suivants, lesquels sont reconnus par l'automate ? `aaa, aba, aabbaba, abaaba, aababaaaa, abcb`

Question 3. Quel est le langage reconnu par cet automate ?

Question 4. Déterminiser l'automate.

On considère maintenant le langage : $\texttt{aba(a + b)}^*$

Question 5. Parmi les mots suivants, lesquels appartiennent au langage ? `aaa, aba, aabbaba, abaaba, aababaaaa, abcb`

Question 6. Construire un automate fini reconnaissant ce langage.

Exercice II. Compilation

On souhaite ajouter à notre calculette le support des tableaux.

Pour la suite, on supposera qu'un tableau de taille `lg` connue et fixée est stocké dans la pile en empilant d'abord sa longueur puis ses éléments. Par exemple, le tableau `tab = [12, 3]` est transcrit par la suite de mots `2 12 3`. On supposera également que l'adresse du tableau correspond à la position du mot indiquant sa taille.

On utilise les opcodes **PUSHR x** et **STORER x** qui effectuent les opérations suivantes :

Code	Pile	*sp*	*pc*
PUSHR n	P[sp-1] := P[P[sp-1] + n]	sp	pc+2
STORER n	P[P[sp-2] + n] := P[sp-1]	sp-2	pc+2

Soit le code suivant

```
PUSHI 3
PUSHI 3
PUSHI 0
PUSHI 1

PUSHI 2
PUSHI 5
STORER 1

PUSHI 0
PUSHI 0
PUSHR 1
PUSHI 1
SUB
PUSHR 1
STORER 1

PUSHI 0
PUSHR 1
WRITE
POP
HALT
```

et le résultat de son assemblage

```
 Adr |  Instruction
-----+--------------
   0 | PUSHI       3
   2 | PUSHI       3
   4 | PUSHI       0
   6 | PUSHI       1
   8 | PUSHI       2
  10 | PUSHI       5
  12 | STORER      1
  14 | PUSHI       0
  16 | PUSHI       0
  18 | PUSHR       1
  20 | PUSHI       1
  22 | SUB
  23 | PUSHR       1
  25 | STORER      1
  27 | PUSHI       0
  29 | PUSHR       1
  31 | WRITE
  32 | POP
  33 | HALT
```

Question 7. Compléter la trace d'exécution suivante.

```
 pc |               |    fp   pile
====================================================
   0 | PUSHI       3 |     0 [ ] 0
   2 | PUSHI       3 |     0 [ 3 ] 1
   4 | PUSHI       0 |     0 [ 3 3 ] 2
   6 | PUSHI       1 |     0 [ 3 3 0 ] 3
   8 | PUSHI       2 |     0 [ 3 3 0 1 ] 4
  10 | PUSHI       5 |     0 [ 3 3 0 1 2 ] 5
  12 | STORER      1 |     0 [ 3 3 0 1 2 5 ] 6
  14 | PUSHI       0 |     0 [ 3 3 0 5 ] 4
  16 | PUSHI       0 |     0 [ 3 3 0 5 0 ] 5
  18 | PUSHR       1 |     0 [ 3 3 0 5 0 0 ] 6
  20 | PUSHI       1 |     0 [ 3 3 0 5 0 3 ] 6
   .
   .
   .
   .
   .
  31 | WRITE         |     0 [ 3 5 0 5 5 ] 5
     5
  32 | POP           |     0 [ 3 5 0 5 5 ] 5
  33 | HALT          |     0 [ 3 5 0 5 ] 4
```

Question 8. Écrire le code MVàP correspondant à la fonction suivante :

```
int b = [2, 5, 6]
int f ( x ) {
  b[x]=1
  return b[x+1]
}
println(f(1))
```

On donne la grammaire suivante permettant de prendre en compte la déclaration (initialisée) de tableaux :

```
declaration returns [ String code ]
    <...>
    | 'int'  IDENTIFIANT '=' '[' elems ']'
      {  // À compléter }
    <...>
    ;

elems returns [ String code, int size]
@init{ $code = new String(); $size = 0; }
    : ( ENTIER
        {  // À compléter  }
        ( ',' ENTIER
        { // À compléter  }
        )*
      )?
    ;
```

Question 9. Compléter les actions de la grammaire afin de générer le code MVàP correct.

On se donne également la grammaire de l'utilisation des tableaux :

```
instruction returns [ String code ]
    <...>
    | IDENTIFIANT '[' expr ']' '=' expr finInstruction
      { // À compléter }
    <...>
    ;

expr returns [ String code ]
    <...>
    | IDENTIFIANT '[' expr ']'
      { // À compléter }
    <...>
    ;
```

Question 10. Compléter les actions de la grammaire afin de générer le code MVàP.

Exercice III. Analyse LL et SLR

Soit la grammaire G d'axiome S et de terminaux {id, nb, [,]} définie par :

$$\begin{cases} \texttt{S} \rightarrow \texttt{id A} \\ \texttt{A} \rightarrow \texttt{[B] A} \mid \texttt{[B]} \\ \texttt{B} \rightarrow \texttt{nb} \end{cases}$$

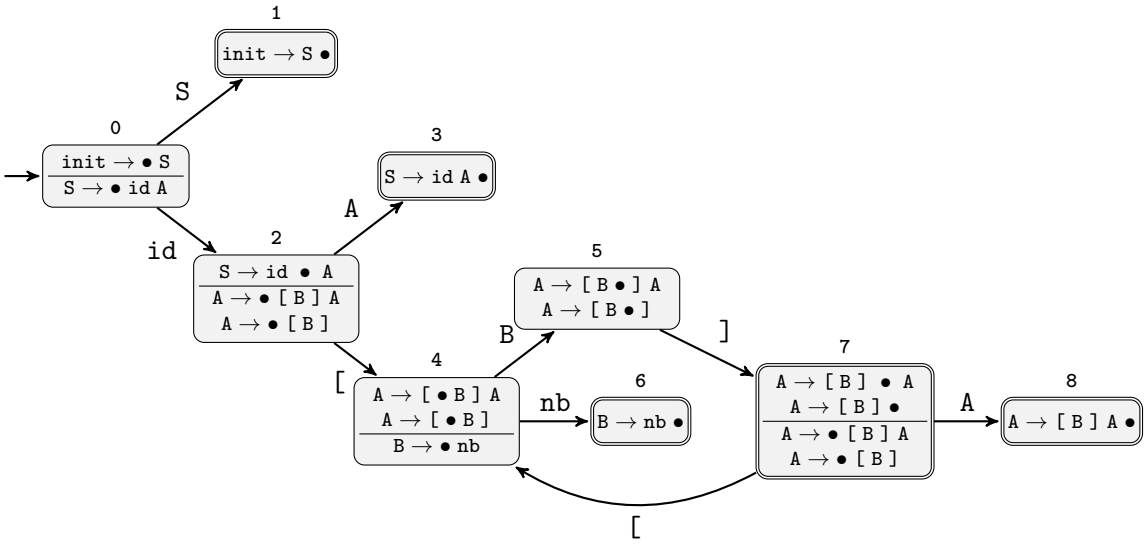
Question 11. Donner un arbre d'analyse pour le mot suivant : `id [nb] [nb]` .

Question 12. Cette grammaire est elle `LL(1)` ? Justifier.

On considère la version augmentée de la grammaire `G` :

$$\begin{cases} \texttt{init} \to \texttt{S} \\ \texttt{S} \to \texttt{id A} \\ \texttt{A} \to \texttt{[B] A} \mid \texttt{[B]} \\ \texttt{B} \to \texttt{nb} \end{cases}$$

On dispose de l'automate fini caractéristique des items `LR(0)` de `G`.



Question 13.

13 . a) Quel état de l'automate contient un conflit décaler/réduire ?

13 . b) Donner la table des ensembles `Suivant` .

13 . c) Expliquer comment le conflit décaler/réduire précédent se résout.

On donne la table d'analyse `SLR` de `G`.

	$	id	nb	[	]	S	A	B
0		d 2				1		
1	accepter							
2				d 4			3	
3	r S → id A							
4			d 6					5
5					d 7			
6					r B → nb			
7	r A → [B]			d 4			8	
8	r A → [B] A							

Question 14. Pour la table `SLR`, expliquer de façon claire et détaillée comment est déterminée :

14 . a) la ligne associée à l'état `0` ;

14 . b) la ligne associée à l'état `1` ;

14 . c) la ligne associée à l'état `7` .

Question 15.

15 . a) Dérouler l'analyse `SLR` sur l'entrée `id [nb] [nb]` .

15 . b) Grâce à cette analyse, comment obtient on la dérivation droite pour `id [nb] [nb]` ?

15 . c) Dérouler l'analyse `SLR` sur l'entrée `id [nb [nb]` .