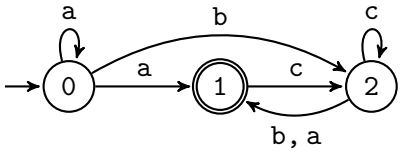
Université de Caen **Année 2017/2018**
UFR des Sciences **2ème session**
Jeudi 5 Juillet

L3 – INF6ACT
Théorie des langages et compilation
durée 2h
Les notes de cours et TD sont autorisées.

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 suivant :



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

Question 2. Parmis les mots suivants, lesquels sont reconnus par l'automate ? `aa, abc, cacb, abaab, babca, abcb`

Question 3. Déterminiser l'automate.

On regarde maintenant le langage : $(c(a^*b + \varepsilon))^*b$

Question 4. Parmis les mots suivants, lesquels appartiennent au langage ? `aaaab, caaab, cab, abaab, cbab, caaaabb`

Question 5. Construire un automate reconnaissant ce langage avec la méthode de votre choix.

Exercice II. Compilation

Cet exercice porte sur un cas simple de l'instruction de branchement `switch` (présente par exemple en java) et à sa compilation depuis le langage de la calculette vers les instructions de la MVàP.

La syntaxe est la suivante :

```
instruction returns [String code]
  <...>
  | 'switch' '(' e=expression ')' '{'
  ('case' ec=expression ':'  i=instruction NEWLINE )*
  'default' ':' idefault=instruction NEWLINE '}'
  ;
```

La sémantique associée est la suivante: si l'expression `e` est égale à l'un des cas, alors on exécute toutes les instructions en dessous de ce cas (incluant les cas suivants).

Pour vous aider à comprendre, voici un cas avec différentes traces d'exécutions.

(**Exemple 1:** Calculette)

```
int x
readln(x)
switch (x)  {
       case 1: println(1)
       case 3: println(3)
       case 2: println(2)
       default: println(42)
}
```

(Traces d'exécutions : r pour read, w pour write)

```
r 1
    w 1
    w 3
    w 2
    w 42
```

```
r 2
    w 2
    w 42
```

```
r 45
    w 42
```

Question 6. Que donne l'exécution si on lit 3?

Soit le code suivant **Exemple 2**

```
 JUMP 0
LABEL 1
  PUSHL -3
  DUP
  PUSHI 0
  NEQ
  JUMPF 2
  DUP
  PUSHI 1
  NEQ
  JUMPF 3
  POP
  JUMP 4
LABEL 2
LABEL 3
  PUSHI 1
  STOREL -4
  RETURN
LABEL 4
  PUSHI 0
  PUSHL -3
  PUSHI -1
  ADD
  CALL 1
  POP
  PUSHI 0
  PUSHL -3
  PUSHI -2
  ADD
  CALL 1
  POP
  ADD
  STOREL -4
  RETURN
LABEL 0
  PUSHI 0
  PUSHI 5
  CALL 1
  POP
  WRITE
  POP
  HALT
```

et le résultat de son assemblage

```
   Adr |  Instruction
-------+--------------
     0 | JUMP  48
     2 | PUSHL -3
     4 | DUP
     5 | PUSHI 0
     7 | NEQ
     8 | JUMPF 19
    10 | DUP
    11 | PUSHI 1
    13 | NEQ
    14 | JUMPF 19
    16 | POP
    17 | JUMP  24
    19 | PUSHI 1
    21 | STOREL -4
    23 | RETURN
    24 | PUSHI 0
    26 | PUSHL -3
    28 | PUSHI -1
    30 | ADD
    31 | CALL 2
    33 | POP
    34 | PUSHI 0
    36 | PUSHL -3
    38 | PUSHI -2
    40 | ADD
    41 | CALL 2
    43 | POP
    44 | ADD
    45 | STOREL -4
    47 | RETURN
    48 | PUSHI 0
    50 | PUSHI 5
    52 | CALL 2
    54 | POP
    55 | WRITE
    56 | POP
    57 | HALT
```

Question 7. Compléter la trace d'exécution du code de l'exemple 2 donné ci dessous:

```
  0 | JUMP       48 |     0 [ ] 0
 48 | PUSHI       0 |     0 [ ] 0
 50 | PUSHI       5 |     0 [ 0 ] 1
 52 | CALL        2 |     0 [ 0 5 ] 2
  2 | PUSHL      -3 |     4 [ 0 5 54 0 ] 4
  4 | DUP           |     4 [ 0 5 54 0 5 ] 5
  5 | PUSHI       0 |     4 [ 0 5 54 0 5 5 ] 6
  7 | NEQ           |     4 [ 0 5 54 0 5 5 0 ] 7
  8 | JUMPF      19 |     4 [ 0 5 54 0 5 1 ] 6
 10 | DUP           |     4 [ 0 5 54 0 5 ] 5
 11 | PUSHI       1 |     4 [ 0 5 54 0 5 5 ] 6
 13 | NEQ           |     4 [ 0 5 54 0 5 5 1 ] 7
 14 | JUMPF      19 |     4 [ 0 5 54 0 5 1 ] 6
```

```
  4 | DUP           |     8 [ 0 5 54 0 0 4 33 4 4 ] 9
  5 | PUSHI       0 |     8 [ 0 5 54 0 0 4 33 4 4 4 ] 10
  7 | NEQ           |     8 [ 0 5 54 0 0 4 33 4 4 4 0 ] 11
  8 | JUMPF      19 |     8 [ 0 5 54 0 0 4 33 4 4 1 ] 10
 10 | DUP           |     8 [ 0 5 54 0 0 4 33 4 4 ] 9
 11 | PUSHI       1 |     8 [ 0 5 54 0 0 4 33 4 4 4 ] 10
```

```
 30 | ADD           |     8 [ 0 5 54 0 0 4 33 4 0 4 -1 ] 11
 31 | CALL        2 |     8 [ 0 5 54 0 0 4 33 4 0 3 ] 10
  2 | PUSHL      -3 |    12 [ 0 5 54 0 0 4 33 4 0 3 33 8 ] 12
  4 | DUP           |    12 [ 0 5 54 0 0 4 33 4 0 3 33 8 3 ] 13
  5 | PUSHI       0 |    12 [ 0 5 54 0 0 4 33 4 0 3 33 8 3 3 ] 14
```

Question 8. Donner le code MVaP généré par le programme suivant:

```
int x
readln(x)
switch (x)
       case 1: println(1)
       case 3: println(3)
       default: println(42)
```

Question 9. Compléter le code antlr pour obtenir la génération de code.

Question 10. Donnez le code antlr qui permet de générer le code MVàP approprié pour le switch. Si on le souhaite, on peut s'inspirer du la structure donnée ci-dessous.

```
instruction returns [String code]
  <...>
  | 'switch' '(' e=expression ')' '{'
    \{
      // calculer e

      String inst_code =  new String();
    \}
  ('case' ec=expression ':'  i=instruction NEWLINE
   \{
     // Faire le test et le saut associé

     // Préparer le code des instruction et l'ajouter à instr_code
   \}
  )*
  'default' ':' idefault=instruction NEWLINE '}'
   \{
      // Faire le saut par défaut

      // Mettre les codes stockées précédemment

      // Ajouter le code de l'instruction du cas par défaut
   \}
  ;
```

Question 11. Expliquer ce que fait le code MVaP de l'exemple 2. Donner un programme calculette produisant ce code.

Exercice III. Analyse SLR

Soit la grammaire `G` d'axiome `S` et de terminaux {`(`, `)`, `instruction`} définie par :

$$\begin{cases} \texttt{S} & \rightarrow & \texttt{Bloc} \mid \texttt{( )} \\ \texttt{Bloc} & \rightarrow & \texttt{(instruction)} \end{cases}$$

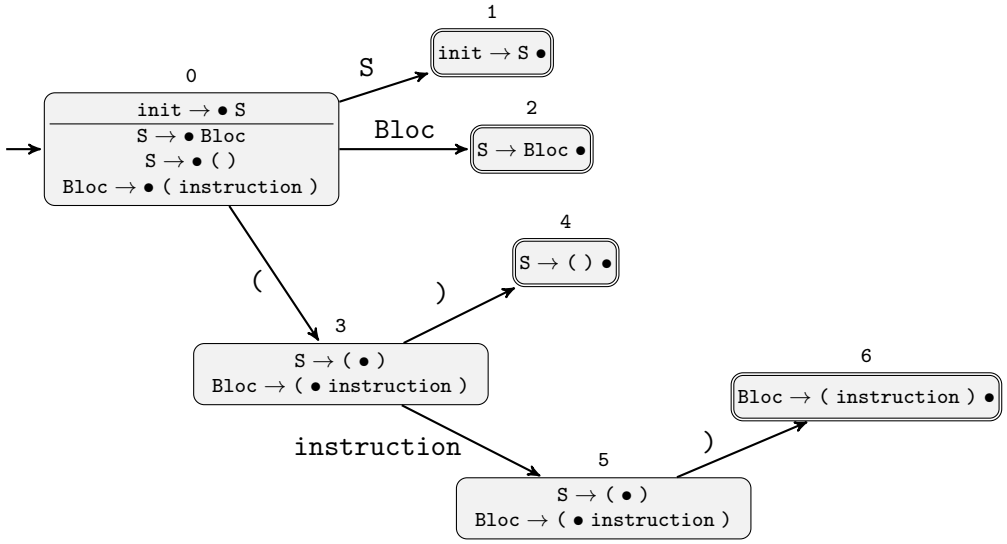
Question 12. Donner un arbre d'analyse pour le mot suivant : `( instruction )` .

Question 13. Peut-on appppliquer une analyse `LL(1)` à la grammaire `G` ? Justifier.

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

$$\begin{cases} \texttt{init} & \rightarrow & \texttt{S} \\ \texttt{S} & \rightarrow & \texttt{Bloc} \mid \texttt{( )} \\ \texttt{Bloc} & \rightarrow & \texttt{(instruction)} \end{cases}$$

On donne l'automate fini caractéristique des items `LR(0)` de la grammaire augmentée et sa table d'analyse `SLR`.



	$	(	)	instruction	S	Bloc
0		decaler 3			1	2
1	accepter					
2	reduire S → Bloc					
3			decaler 4	decaler 5		
4	reduire S → ( )					
5			decaler 6			
6	reduire Bloc → (instruction)					

Question 14.

14.a) Dérouler l'analyse SLR sur l'entrée (instruction) . Comment retrouver à partir de cette analyse la dérivation droite associée à (instruction) ?

14.b) Dérouler l'analyse SLR sur l'entrée (instruction instruction) .

Question 15. Expliquer de façon claire et détaillée comment la ligne relative à l'état 0 dans la table SLR est obtenue. Même question pour la ligne de l'état 2.