Université de Caen **Année 2018/2019**
UFR des Sciences **1ère session**
Mercredi 20 mars

L3 – INF6ACT
Théorie des langages et compilation
durée 2h

Documents autorisés : notes personnelles, diapos du cours.

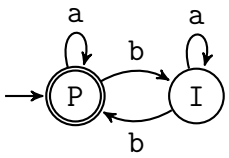
Chaque candidat doit, en début d'épreuve, porter son nom dans le coin de la copie réservé à cet usage; il le cachettera par collage après la signature de la feuille d'émargement. Sur chacune des copies intercalaires, il portera son numéro de place.

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

On se donne l'automate fini $\mathcal{A}_1$ suivant :



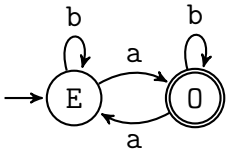
Question 1. Donner la table de transitions de cet automate.

Question 2. Donner l'exécution de l'automate $\mathcal{A}_1$ sur les quatre mots suivants : ε, `aa`, `ab`, `abbbaaab`. Lesquels sont acceptés par $\mathcal{A}_1$?

Question 3. Quel est le langage reconnu par l'automate $\mathcal{A}_1$?

Question 4. Donner une expression régulière de ce langage.

On considère également l'automate fini $\mathcal{A}_2$:



Question 5. Quel est le langage reconnu par $\mathcal{A}_2$? En donner une expression régulière.

On désigne maintenant par $\mathtt{L}_1$ et $\mathtt{L}_2$ les langages réguliers reconnus par $\mathcal{A}_1$ et $\mathcal{A}_2$, et par $\mathtt{L}_1 \cap \mathtt{L}_2$ leur intersection.

Question 6. Construire l'automate produit de $\mathcal{A}_1$ et $\mathcal{A}_2$ reconnaissant $\mathtt{L}_1 \cap \mathtt{L}_2$.

Exercice II. Compilation — À rendre sur une copie séparée

On souhaite ajouter à notre calculette le support de la boucle `for in`. Le principe de cette boucle (présente en python) est que `for el in tab` effectue une boucle en prenant pour `el` tous les éléments du tableau `tab`.

Pour la suite, on supposera qu'un tableau de taille `n` 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 longueur.

On utilise l'opcode **PUSHR x** qui effectue l'opération suivante:

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

Soit le code suivant

```
  PUSHI 0
  PUSHI 0
  PUSHI 2
  PUSHI 12
  PUSHI 3
  JUMP 0
LABEL 0

  PUSHI 0
  STOREG 0
LABEL 1
  PUSHG 0
  PUSHG 2
  INF
  JUMPF 2

  PUSHG 0
  PUSHR 3
  STOREG 1

  PUSHG 1
  WRITE
  POP

  PUSHG 0
  PUSHI 1
  ADD
  STOREG 0

  JUMP 1
LABEL 2
  HALT
```

et le résultat de son assemblage

```
Adr |  Instruction
----+--------------
   0 | PUSHI       0
   2 | PUSHI       0
   4 | PUSHI       2
   6 | PUSHI      12
   8 | PUSHI       3
  10 | JUMP       12
  12 | PUSHI       0
  14 | STOREG      0
  16 | PUSHG       0
  18 | PUSHG       2
  20 | INF
  21 | JUMPF      42
  23 | PUSHG       0
  25 | PUSHR       3
  27 | STOREG      1
  29 | PUSHG       1
  31 | WRITE
  32 | POP
  33 | PUSHG       0
  35 | PUSHI       1
  37 | ADD
  38 | STOREG      0
  40 | JUMP       16
  42 | HALT
```

Question 7. Compléter la trace d'exécution suivante.

```
 pc |              |    fp   pile
====================================================
  0 | PUSHI       0 |     0 [ ] 0
  2 | PUSHI       0 |     0 [ 0 ] 1
  4 | PUSHI       2 |     0 [ 0 0 ] 2
  6 | PUSHI      12 |     0 [ 0 0 2 ] 3
  8 | PUSHI       3 |     0 [ 0 0 2 12 ] 4
 10 | JUMP       12 |     0 [ 0 0 2 12 3 ] 5
 12 | PUSHI       0 |     0 [ 0 0 2 12 3 ] 5
 14 | STOREG      0 |     0 [ 0 0 2 12 3 0 ] 6
 16 | PUSHG       0 |     0 [ 0 0 2 12 3 ] 5
 18 | PUSHG       2 |     0 [ 0 0 2 12 3 0 ] 6
 20 | INF           |     0 [ 0 0 2 12 3 0 2 ] 7
 .
 .
 .
 .
 .
 31 | WRITE         |     0 [ 0 12 2 12 3 12 ] 6
 32 | POP           |     0 [ 0 12 2 12 3 12 ] 6
 33 | PUSHG       0 |     0 [ 0 12 2 12 3 ] 5
 35 | PUSHI       1 |     0 [ 0 12 2 12 3 0 ] 6
 37 | ADD           |     0 [ 0 12 2 12 3 0 1 ] 7
 38 | STOREG      0 |     0 [ 0 12 2 12 3 1 ] 6
 40 | JUMP       16 |     0 [ 1 12 2 12 3 ] 5
 16 | PUSHG       0 |     0 [ 1 12 2 12 3 ] 5
 18 | PUSHG       2 |     0 [ 1 12 2 12 3 1 ] 6
 20 | INF           |     0 [ 1 12 2 12 3 1 2 ] 7
 21 | JUMPF      42 |     0 [ 1 12 2 12 3 1 ] 6
 23 | PUSHG       0 |     0 [ 1 12 2 12 3 ] 5
 25 | PUSHR       3 |     0 [ 1 12 2 12 3 1 ] 6
 27 | STOREG      1 |     0 [ 1 12 2 12 3 3 ] 6
 29 | PUSHG       1 |     0 [ 1 3 2 12 3 ] 5
 31 | WRITE         |     0 [ 1 3 2 12 3 3 ] 6
 32 | POP           |     0 [ 1 3 2 12 3 3 ] 6
 33 | PUSHG       0 |     0 [ 1 3 2 12 3 ] 5
 35 | PUSHI       1 |     0 [ 1 3 2 12 3 1 ] 6
 37 | ADD           |     0 [ 1 3 2 12 3 1 1 ] 7
 38 | STOREG      0 |     0 [ 1 3 2 12 3 2 ] 6
 40 | JUMP       16 |     0 [ 2 3 2 12 3 ] 5
 16 | PUSHG       0 |     0 [ 2 3 2 12 3 ] 5
 .
 .
 .
 .
```

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

```
int b = [ 4 , 5]
int f ( x ) {
  return x+3
}
println( f(6) + 1)
```

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

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

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

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

On se donne également la grammaire pour la boucle `for in`:

```
instruction returns [ String code ]
    <...>
    | 'for' IDENTIFIANT 'in' IDENTIFIANT bloc
      { // À completer }
    <...>
    ;
```

Question 10. Compléter les actions de la grammaire afin de générer le code MVàP pour cette boucle. On pourra supposer l'existence d'une variable globale spéciale `i_for` que l'on pourra utiliser pour stocker l'indice courant du tableau.

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

On considère la grammaire G définie par l'axiome S, les variables {S, GN, GV, GP, PR}, les terminaux {art, de, nom, prepo, verbe} et les règles de production :

$$\begin{cases} \texttt{S} & \to & \texttt{GN GV} \\ \texttt{GN} & \to & \texttt{art nom GP} \\ \texttt{GP} & \to & \texttt{de GN} \mid \varepsilon \\ \texttt{GV} & \to & \texttt{verbe PR GN} \\ \texttt{PR} & \to & \texttt{prepo} \mid \varepsilon \end{cases}$$

Question 11. Donner un arbre d'analyse pour le mot suivant :
`art nom de art nom verbe prepo art nom`

Question 12.

12.a) Quelles sont les variables effaçables ?

12.b) Donner la table des ensembles Premier.

12.c) On note $ le symbole terminal qui marque la fin des mots à analyser. Donner la table des ensembles Suivant.

On dispose de la table d'analyse LL(1) de G :

	$	art	de	nom	prepo	verbe
S		S → GN GV				
GN		GN → art nom GP				
GP	GP → ε		GP → de GN			GP → ε
GV						GV → verbe PR GN
PR		PR → ε			PR → prepo	

Question 13. Expliquer de façon claire et détaillée comment la ligne associée à la variable S dans la table est obtenue. Même question pour la ligne correspondant à GP.

Question 14. Dérouler l'analyse LL(1) sur l'entrée `art nom verbe art nom` . À partir de cette analyse, comment retrouver la dérivation gauche associée à `art nom verbe art nom` ?