Université de Caen **Année 2021/2022**
UFR des Sciences **2ème session**
Vendredi 1 juillet

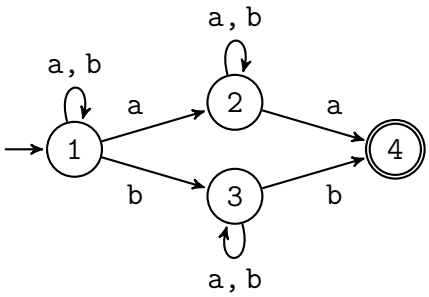
L3 – INF6ACT
Théorie des langages et compilation
durée 2h
Documents autorisés : notes personnelles, diapos du cours.

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.

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 SLR)
- l'autre qui traite l'exercice II (Compilation)

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

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



Question 1. Indiquer si les mots suivants `abaa` , `aaab` , `baaba` sont reconnus par l'automate $\mathcal{A}$ et, si oui, donner un calcul acceptant de $\mathcal{A}$ sur le mot.

Question 2. Pourquoi cet automate $\mathcal{A}$ est non déterministe ?

Question 3. Donner une expression régulière du langage reconnu par cet automate $\mathcal{A}$.

Question 4. Donner une grammaire qui engendre le langage reconnu par $\mathcal{A}$.

Question 5. Donner un automate qui reconnaît le complémentaire du langage reconnu par $\mathcal{A}$.

Exercice II. Compilation À rendre sur une copie séparée

On souhaite ajouter à notre calculette le support de l'opération de composition des fonctions. Pour cela, nous introduisons l'opérateur `o` défini par:

`f o g(x) = f(g(x))`

On utilise l'opcode **DUP** qui duplique la valeur en haut de la pile.

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

Question 6. Quelle valeur affiche le programme `A` ci-dessous lorsqu'on l'exécute ?

```
var y : int  = 1
fun f ( x : int ) : int {
  if (x > 0)
    return x+1
  else
    return -x
}
fun g ( x : int ) : int {
  return x*x
}
write( f o g (y))
write( g o f (y+2))
```

Programme `A`.

On se donne le code suivant :

```
var x : int = 10
fun h : int ( y : int) {
return x + y
}
write(h o h (x))
```

qui se traduit en MVàP par

```
  PUSHI 10
  JUMP Start
LABEL h
  PUSHG 0
  PUSHL -3
  ADD
  STOREL -4
  RETURN
RETURN
LABEL Start
  PUSHI 0
  PUSHG 0
  CALL h
  POP
  DUP
  CALL h
  POP
  WRITE
  POP
HALT
```

et le résultat de son assemblage

```
 Adr |  Instruction
-----+--------------
   0 | PUSHI      10
   2 | JUMP       13
   4 | PUSHG       0
   6 | PUSHL      -3
   8 | ADD
   9 | STOREL     -4
  11 | RETURN
  12 | RETURN
  13 | PUSHI       0
  15 | PUSHG       0
  17 | CALL        4
  19 | POP
  20 | DUP
  21 | CALL        4
  23 | POP
  24 | WRITE
  25 | POP
  26 | HALT
```

Question 7. Compléter la trace d'exécution suivante.

```
  pc |               |    fp   pile
=================================================
   0 | PUSHI      10 |     0 [ ] 0
   2 | JUMP       13 |     0 [ 10 ] 1
  13 | PUSHI       0 |     0 [ 10 ] 1
  15 | PUSHG       0 |     0 [ 10 0 ] 2
...
```

Question 8. Compléter le code MVàP ci-dessous correspondant au programme `A` donné dans la question 6.

```
  PUSHI 1
  JUMP Start
LABEL f
  PUSHL -3
  PUSHI 0
  SUP
  JUMPF Label1
  PUSHL -3
  PUSHI 1
  ADD
  STOREL -4
  RETURN
LABEL Label1
  PUSHL -3
  PUSHI -1
  MUL
  STOREL -4
  RETURN
RETURN
```

```
LABEL g
  PUSHL -3
  PUSHL -3
  MUL
  STOREL -4
  RETURN
RETURN
Label Start
< À compléter >
```

On donne la grammaire suivante des expressions:

```
expression
: '-' s=expression
| g=expression op=('*'|'/') d=expression
| g=expression op=('+'|'-') d=expression
| '(' s=expression ')'
| ENTIER
| IDENTIFIANT
| IDENTIFIANT '(' args ')'
```

Question 9. Proposer une modification de la grammaire qui permette l'utilisation de la composition de deux fonctions uniquement.

Question 10. Compléter les actions de la grammaire concernant la composition afin de générer le code MVàP correct. On essaiera de limiter au maximum la taille utilisée dans la pile.

Question 11. Proposer une modification de la grammaire qui permette l'utilisation de la composition d'un nombre arbitraire de fonctions (ex: `g o f o f o g (x)`).

Exercice III. Analyse `SLR` À rendre avec l'exercice I

Soit la grammaire $\mathcal{G}$ d'axiome `S` avec trois symboles terminaux { `(` , `)` , `a`}, et définie par :

$$\begin{cases} \texttt{S} \rightarrow \texttt{(}\ \texttt{L}\ \texttt{)} \mid \texttt{a} \\ \texttt{L} \rightarrow \texttt{S}\ \texttt{L} \mid \varepsilon \end{cases}$$

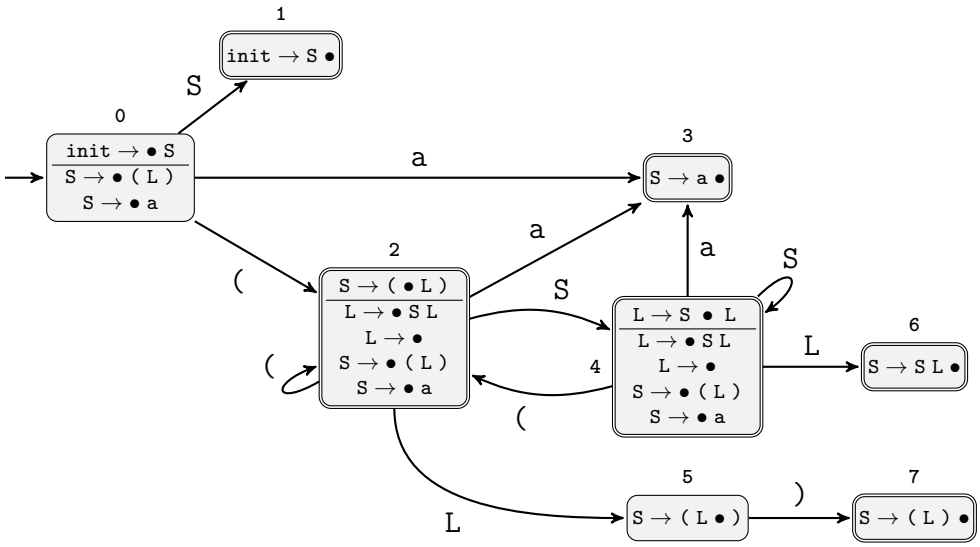
Question 12. Donner un arbre d'analyse pour le mot suivant :

`( ( ) a )`

On considère la version augmentée de la grammaire $\mathcal{G}$:

$$\begin{cases} \texttt{init} \rightarrow \texttt{S} \\ \texttt{S} \rightarrow \texttt{(}\ \texttt{L}\ \texttt{)} \mid \texttt{a} \\ \texttt{L} \rightarrow \texttt{S}\ \texttt{L} \mid \varepsilon \end{cases}$$

On dispose de l'automate fini caractéristique des items `LR(0)` de $\mathcal{G}$.



Question 13.

13 . a) Quels états de l'automate contiennent un conflit *décaler/réduire* ?

13 . b) Calculer l'ensemble `Suivant(L)` .

13 . c) Expliquer comment les conflits *décaler/réduire* précédent se résolvent.

On donne la table d'analyse `SLR` de $\mathcal{G}$.

	$	(	)	a	S	L
0		d 2		d 3	1	
1	accepter					
2		d 2	r L → ε	d 3	4	5
3	r S → a	r S → a	r S → a	r S → a		
4		d 2	r L → ε	d 3	4	6
5			d 7			
6			r L → SL			
7	r S → (L)	r S → (L)	r S → (L)	r S → (L)		

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

14 . a) la ligne associée à l'état `5` ;

14 . b) la ligne associée à l'état `6` ;

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

À noter que `Suivant(S)` = {`a`, `(`, `)`, `$`}

Question 15.

15 . a) Dérouler l'analyse `SLR` sur l'entrée `( ( ) a )` . À chaque étape, préciser le symbole examiné, l'état de la pile et l'action réalisée.

15 . b) Grâce à cette analyse, comment obtient on la dérivation droite pour `( ( ) a )` ?

15 . c) Dérouler l'analyse `SLR` sur l'entrée `()()` .