Université de Caen **Année 2013/2014**
UFR de Sciences **2ème session**
L3 **Vendredi 27 juin**

ULI6B 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

Soit L le langage décrit par l'expression rationnelle $a(a + ba)^*b$.

Question 1. Donner trois mots de longueur 6 appartenant au langage L. Donner le mot de longueur minimal appartenant à L.

Question 2. En utilisant l'algorithme de Glushkov vu en cours, donner un automate fini non déterministe A qui reconnaît le langage L.

Question 3. Construire un automate fini déterministe équivalent à A.

Question 4. Donner une grammaire G qui engendre le langage L.

Exercice II. Génération de code

On souhaite enrichir le langage de calculette vu en TD/TP avec les assignations parallèles. On veut ainsi pouvoir affecter simultanément une suite d'expressions à une suite de variables. Par exemple, avec l'assignation `x,y,z = 1,2,3` , la variable `x` (respectivement `y`, `z`) prend la valeur `1` (respectivement `2`, `3`).
On suppose données les règles de la grammaire pour les autres instructions de la calculette.

Question 5. On suppose connues `adr(x)`, `adr(y)`, `adr(z)`, les adresses des variables `x,y,z`. Quel code MVàP doit être produit par le compilateur pour effectuer le calcul suivant ?

```
x,y,z = 1,7-4,3*6
```

Question 6. Voici les règles de la grammaire pour la prise en compte des assignations parallèles :

```
assignation : listeparams '=' listeargs ;
listeparams : (IDENTIFIANT ( ',' IDENTIFIANT)* ) ;
listeargs : (expression ( ',' expression)* ) ;
```

Écrire les règles ANTLR pour la génération de code. Attention à l'ordre : c'est la première expression qui doit être affectée à la première variable et ainsi de suite.
(NB. `tablesSymboles.adresseVar($IDENTIFIANT.text)` récupère l'adresse de l'identifiant.)

Question 7. Soit le code suivant produit par le compilateur.

```
    PUSHI 0
    PUSHI 0
    PUSHI 0
    PUSHI 1
    PUSHI 1
    PUSHI 0
    STOREG 2
    STOREG 1
    STOREG 0
LABEL 0
    PUSHG 2
    PUSHI 8
```

```
    INF
    JUMPF 1
    PUSHG 1
    PUSHG 0
    PUSHG 1
    ADD
    PUSHG 2
    PUSHI 1
    ADD
    STOREG 2
    STOREG 1
    STOREG 0
    PUSHG 1
    WRITE
    POP
    JUMP 0
LABEL 1
    HALT
```

Que réalise ce code ? Identifier les affectations parallèles.

Exercice III. Analyse *LL*

Soit la grammaire G d'axiome `S` et de terminaux { `x` , `[` , `]` , `;` } définie par :

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

Question 8. Donner un arbre d'analyse et les dérivations droite et gauche correspondantes, pour le mot suivant : `[ x ; ; x ]` .

Question 9. Déterminer les variables effaçables. Donner la table des ensembles `Premier`.

Les ensembles `Suivant` obtenus sont:

- `Suivant`$(S) =$ { `]`, `;`, `$` }
- `Suivant`$(El) =$ { `]`, `;` }
- `Suivant`$(R) =$ { `]` }

Question 10. Expliquer en justifiant pourquoi `Suivant`$(El) =$ { `]`, `;` } .

Question 11. Construire la table d'analyse et vérifier que la grammaire est bien $LL(1)$.

Question 12. Appliquer l'analyse $LL(1)$ à l'aide de la table construite sur la formule `[ [ x ] ; ]`

Exercice IV. Analyse *SLR*

On considère la version augmentée de la grammaire G précédente :

$$\begin{cases} \texttt{Init} & \rightarrow & \texttt{S} \\ \texttt{S} & \rightarrow & \texttt{[El R]} \\ \texttt{El} & \rightarrow & \texttt{x} \mid \texttt{S} \mid \epsilon \\ \texttt{R} & \rightarrow & \texttt{;\ El R} \mid \epsilon \end{cases}$$

On donne l'automate fini caractéristique des items $LR(0)$ de la grammaire G en figure 1 et sa table d'analyse SLR en figure 2.

Question 13. Dérouler l'analyse SLR sur l'entrée `[ x ; [ ] ]` .

Question 14. Expliquer de façon claire et détaillée comment la ligne relative à l'état `1` dans la table SLR est obtenue. Même question pour la ligne de l'état `7`.

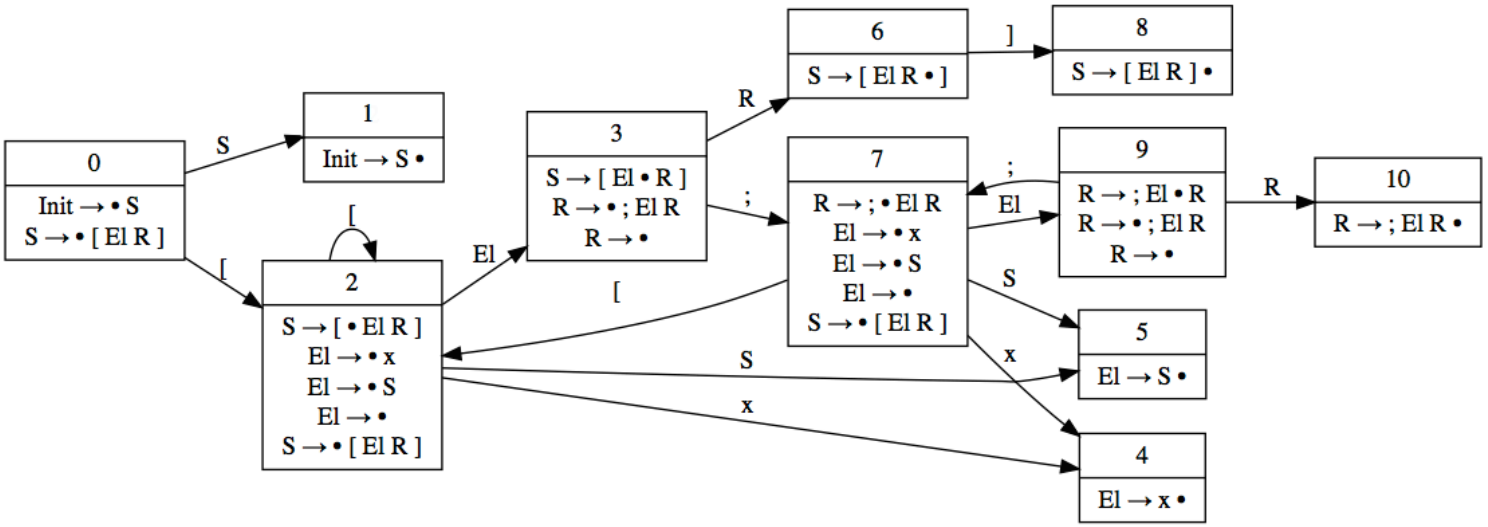

Figure 1: Automate fini caractéristique des items $LR(0)$ de la grammaire G

State	[	]	x	;	$	S	El	R
0	shift(2)					1		
1					**accept**			
2	shift(2)	reduce(El $\rightarrow \epsilon$)	shift(4)	reduce(El $\rightarrow \epsilon$)		5	3	
3		reduce(R $\rightarrow \epsilon$)		shift(7)				6
4		reduce(El $\rightarrow$ x)		reduce(El $\rightarrow$ x)				
5		reduce(El $\rightarrow$ S)		reduce(El $\rightarrow$ S)				
6		shift(8)						
7	shift(2)	reduce(El $\rightarrow \epsilon$)	shift(4)	reduce(El $\rightarrow \epsilon$)		5	9	
8		reduce(S $\rightarrow$ [El R])		reduce(S $\rightarrow$ [El R])	reduce(S $\rightarrow$ [El R])			
9		reduce(R $\rightarrow \epsilon$)		shift(7)				10
10		reduce(R $\rightarrow$; El R)						

Figure 2: Table SLR de la grammaire G