Analyse Syntaxique - Annee 2009-2010

Devoir Surveillé

(Durée 1 heure 30)

Exercice 1

En utilisant la syntaxe de `lex`, donnez une expression régulière pour les *noms* des nombres en Français (de 1 : "un" à 9999 : "neuf mille neuf cent quatre vingt dix neuf"). Les conventions utilisées sont les suivantes :

- on utilise l'alphabet en minuscules
- le caractère de séparation est toujours " " (espace),
- "vingt", "cent" et "mille" sont invariables ("quatre vingt", "deux cent", "trois mille"),
- on distingue les spécificités linguistique du Français, dont "treize", "vingt et un", etc.

Vous pouvez alléger l'écriture de votre solution en utilisant des points de suspension, toutefois les expressions régulières du type :
"vingt" | "vingt et un" | ... | "quatre vingt dix neuf",
"mille" | ... | "neuf mille neuf cent quatre vingt dix neuf"
ne rapporteront aucun point.

Exercice 2

Soit la grammaire suivante (`$` est un marqueur de fin de chaîne) :

```
S': S $
S : 'si' E 'alors' S X
  | 'a'
X : 'sinon' S
  |
E : 'b'
```

1. Calculer pour chaque symbole non terminal de cette grammaire les ensembles *Premier* et *Suivant* correspondants.
2. Cette grammaire n'est pas LL(1). Pourquoi ?
3. Habituellement, pour résoudre le probème on fait correspondre tout `'sinon'` au `'si'` le plus proche possible. Dans le cas de conflit entre deux règles pendant l'analyse LL(1), laquelle faut-il choisir pour appliquer ce principe ?

Exercice 3

On considère la grammaire suivante :

```
L : 'id' ';' L
  | 'id' ';'
```

1. Décrire informellement le langage correspondant.
2. Montrer que cette grammaire n'est pas LR(0) mais qu'elle est SLR(1).
3. Proposer une grammaire équivalente qui est LR(0).
4. En modifiant légèrement cette grammaire, donner une grammaire équivalente qui n'est pas SLR(1).

Exercice 4

On considère une grammaire simplifiée pour des déclarations de tableaux en langage C. `TYPE`, `ID`, et `NUM` sont des tokens (terminaux). Les règles sont numérotées de 0 à 7. `$accept` est l'axiome, `$end` un marqueur de fin de chaîne.

```
0  $accept : S $end
1  S : TYPE E ';'
2  E : D
3    | '*' E
4  D : ID
5    | D '[' A ']'
6  A : NUM
7    |
```

Vous trouverez en annexe l'automate produit par `yacc` pour la cette grammaire. Donner la suite de configurations de la pile (en utilisant les numéros d'états donnés par `yacc`) pendant une analyse SLR(1) du texte suivant :

```
int * f [ 10 ] ;
```

Annexe

```
state 0
$accept : . S $end  (0)

TYPE  shift 1
.  error
```

```
S  goto 2

state 1
S : TYPE . E ';'  (1)

ID  shift 3
'*'  shift 4
.  error

E  goto 5
D  goto 6

state 2
$accept : S . $end  (0)

$end  accept

state 3
D : ID .  (4)

.  reduce 4

state 4
E : '*' . E  (3)

ID  shift 3
'*'  shift 4
.  error

E  goto 7
D  goto 6

state 5
S : TYPE E . ';'  (1)

';'  shift 8
.  error

state 6
E : D .  (2)
D : D . '[' A ']'  (5)
```

```
'['  shift 9
';'  reduce 2

state 7
E : '*' E .  (3)

.  reduce 3

state 8
S : TYPE E ';' .  (1)

.  reduce 1

state 9
D : D '[' . A ']'  (5)
A : .  (7)

NUM  shift 10
']'  reduce 7

A  goto 11

state 10
A : NUM .  (6)

.  reduce 6

state 11
D : D '[' A . ']'  (5)

']'  shift 12
.  error

state 12
D : D '[' A ']' .  (5)

.  reduce 5

9 terminals, 5 nonterminals
8 grammar rules, 13 states
```