Polytech Marseille

Automates, Langages et Compilation
Lecture 6 - Corrigé des exercices

N. Baudru

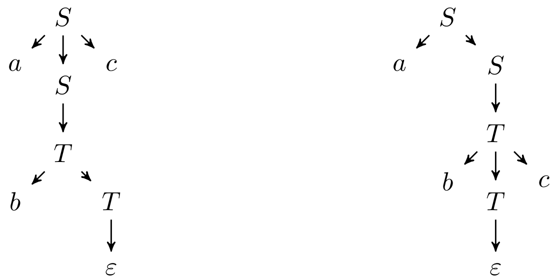
Informatique - 1e année

13 Analyseur syntaxique

Exercice 13.1 ☆ La grammaire suivante est-elle ambiguë ?

$$P = \left| \begin{array}{l} S \to aSc \mid aS \mid T \\ T \to bTc \mid bT \mid \varepsilon \end{array} \right.$$

G est ambigüe pour le mot abc car ce mot possède deux arbres de dérivation :



Exercice 13.2 ☆ Soit la grammaire $G = (\{S, A, M\}, \{a, b\}, S, P)$ ayant pour règles :

$$P = \left| \begin{array}{l} S \to A\$ \\ A \to aM \mid \varepsilon \\ M \to aAbb \mid \varepsilon \end{array} \right.$$

Déterminez les premiers et les suivants de S, A et M et montrez que G est $LL(1)$.

Nous allons calculer les premiers et les suivants des non-terminaux de G. Commençons par les premiers. Pour M, nous obtenons en applicant les règles de calcul :

$premiers(M) = premier(aAbb) \cup premier(\varepsilon)$	(règle 3)
$premier(\varepsilon) = \varepsilon$	(règle 1)
$premier(aAbb) = premier(a) = \{a\}$ car $\varepsilon \notin premier(a)$	(règle 4 et 2).

Au final $premiers(M) = \{a, \varepsilon\}$.

De manière complètement similaire, nous obtenons $premiers(A) = \{a, \varepsilon\}$.

Calculons maintenant les suivants. Pour A, il faut regarder dans quelle partie droite de règles apparait le non-terminal A . Ici, deux possibilités. La règle $S \to A\$$ implique que $\$ \in suivants(A)$. Et on obtient pour $M \to aAbb$ que $suivants(A) \supseteq premiers(bb) = \{b\}$ car $\varepsilon \notin premiers(bb)$.

Pour calculer $suivants(M)$, il faut regarder la règle $A \to aM$. Dans cette règle, rien ne suit le symbole M. Il faut donc considérer que ε joue ici le rôle du v de la formule de calcul des suivants. On en déduit que donc $suivants(M) = (premiers(\varepsilon) \setminus \{\varepsilon\}) \cup suivants(A)$. Et donc $suivants(M) = \{b, \$\}$.

Montrons maintenant que la grammaire est $LL(1)$.

- Clairement il n'y a pas de récursivité gauche.
- Rien à faire pour la première règle car il n'y a pas de choix.
- $possibles(A \to aM) \cap possibles(A \to \varepsilon) = \{a\} \cap \{\$, b\} = \emptyset$.
- $possibles(M \to aAbb) \cap possibles(M \to \varepsilon) = \{a\} \cap \{\$, b\} = \emptyset$.

Exercice 13.3 ☆ Construisez la table prédictive et le programme récursif correspondant à la grammaire de l'exercice 13.2.

Les règles sont numérotées de 0 à 4. La table prédictive :

	a	b	$\$$
S	$A\$, 0$		$A\$, 0$
A	$aM, 1$	$\varepsilon, 2$	$\varepsilon, 2$
M	$aAbb, 3$	$\varepsilon, 4$	$\varepsilon, 4$
a	$shift$		
b		$shift$	
$\$$			$success$

Le programme correspondant :

```
void parseS()  {
    nextSymbol();
    trace = null;

    if ( currentSymbol in possibles(S -> A$) ) {
        addTrace(0);
        parseA();
        if (currentSymbol == '$') SUCCESS; else ERROR;
    }
    else ERROR
}
```

```
parseA() {
    if ( currentSymbol in possibles(A -> aM) ) {
        addTrace(1);
        if (currentSymbol == 'a') nextSymbol();  else ERROR;
        parseM();
    }
    else if ( currentSymbol in possibles(A -> epsilon ) ) {
        addTrace(2);
    }
    else ERROR
}
```

```
parseM() {
    if ( currentSymbol in possibles(M -> aAbb) ) {
        addTrace(3);
        if (currentSymbol == 'a') nextSymbol();  else ERROR;
        parseA();
        if (currentSymbol == 'b') nextSymbol();  else ERROR;
        if (currentSymbol == 'b') nextSymbol();  else ERROR;
```

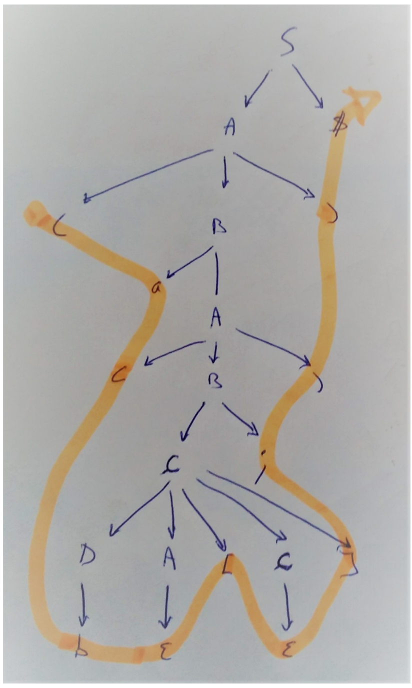
```
        }
        else if ( currentSymbol in possibles(A -> epsilon ) ) {
            addTrace(4);
        }
        else ERROR
}
```

Exercice 13.4 ✰✰ Soit la grammaire $G = (\{A, B, C, D\}, \{a, b, (,), [,], ;\}, A, P)$ avec A pour axiome et P l'ensemble des règles de productions suivantes :

$S \to A\$$
$A \to (B) \mid \varepsilon$
$B \to C; \mid aA$
$C \to DA[C] \mid \varepsilon$
$D \to b \mid \varepsilon$

1. Donnez l'arbre de dérivation gauche correspondant au mot $(a(b[];))$.
2. Calculez les premiers et les suivants de chaque non-terminal.
3. Vérifiez que les conditions LL(1) (les conditions de Knuth) sont satisfaites pour G.
4. Ecrivez dans un langage algorithmique un analyseur descendant récursif pour G.

Question 1 :



Question 2 :

	$premiers$	$suivants$
S	$(, \$$	
A	$(, \varepsilon$	$\$, [,)$
B	$b, (, [, ;, a$	$)$
C	$b, (, [, \varepsilon$	$;,]$
D	b, ε	$(, [$

Question 3 :
Montrons maintenant que la grammaire est $LL(1)$.
— Clairement il n'y a pas de récursivité gauche.
— Rien à faire pour la première règle car il n'y a pas de choix.
— $possibles(A \to (B)) \cap possibles(A \to \varepsilon) = \{(\} \cap \{\$, [,)\} = \emptyset$.

— $possibles(B \to C;) \cap possibles(B \to aA) = \{b, (, [, ;\} \cap \{a\} = \emptyset$.
— $possibles(C \to DA[C]) \cap possibles(C \to \varepsilon) = \{b, (, [\} \cap \{;,]\} = \emptyset$.
— $possibles(D \to b) \cap possibles(D \to \varepsilon) = \{b\} \cap \{(, [\} = \emptyset$.

Question 4 : Les règles sont étiqutées de 0 à 8.

```
void parseS()  {
    nextSymbol();
    trace = null;

    if ( currentSymbol in possibles( S -> A$ ) ) {
        addTrace(0);
        parseA();
        if (currentSymbol == '$') SUCCESS; else ERROR;
    }
    else ERROR
}
```

```
parseA() {
    if ( currentSymbol in possibles( A -> (B) ) ) {
        addTrace(1);
        if (currentSymbol == '(') nextSymbol();  else ERROR;
        parseB();
        if (currentSymbol == ')') nextSymbol();  else ERROR;
    }
    else if ( currentSymbol in possibles( A -> epsilon ) ) {
        addTrace(2);
    }
    else ERROR
}
```

```
parseB() {
    if ( currentSymbol in possibles(B -> C;) ) {
        addTrace(3);
        parseC();
        if (currentSymbol == ';') nextSymbol();  else ERROR;
    }
    else if ( currentSymbol in possibles(B -> aA ) ) {
        addTrace(4);
        if (currentSymbol == 'a') nextSymbol();  else ERROR;
        parseA();
    }
    else ERROR
}
```

```
parseC() {
    if ( currentSymbol in possibles(C -> DA[C]) ) {
        addTrace(5);
        parseD();
        parseA();
        if (currentSymbol == '[') nextSymbol();  else ERROR;
        parseC();
        if (currentSymbol == ']') nextSymbol();  else ERROR;
    }
    else if ( currentSymbol in possibles(C -> epsilon ) ) {
```

```
        addTrace(6);
    }
    else ERROR
}
```

```
parseD() {
    if ( currentSymbol in possibles(D -> b) ) {
        addTrace(7);
        if (currentSymbol == 'b') nextSymbol();  else ERROR;
    }
    else if ( currentSymbol in possibles(D -> epsilon ) ) {
        addTrace(8);
    }
    else ERROR
}
```