Mila University Center Institute of Science and Technology	Computer Science – Semester 5 **COMPILATION – TD N° 04**

Exercise 1.

The FIRST and FOLLOW set for non-terminals A and B in the following productions:
P:

A → BAa | ε

B → bBc | AA

	First	**Follow**
A	ε a b	# a b c
B	ε a b	a b c

Exercise 2.

We define the following grammar G = <N,T,P,S>:

P : **S → A a A b | B b B a**

A → ε

B → ε

a) Construction of the LL(1) parsing table for G.

- The FIRST and FOLLOW sets

	First	Follow
S	a b	#
A	ε	a b
B	ε	a b

- The LL(1) parsing table for G.

	a	b	#
S	**S → A a A b**	**S → B b B a**	
A	**A → ε**	**A → ε**	
B	**B → ε**	**B → ε**	

- This grammar is LL(1) because the LL(1) parsing table is unambiguous (single-defined).

b) Analysis of the string: abb

Stack	Remaining to be analyzed	Action
#S	abb#	Pop (S), stack (bAaA)
#bAaA	abb#	Pop (A)
#bAa	abb#	Pop (a), Move Forward()
#bA	bb#	Pop (A)
#b	bb#	Pop (b), Move Forward()
#	b#	" error "

Analysis of the string: ab

Stack	Remaining to be analyzed	Action
#S	ab#	Pop (S), stack (bAaA)
#bAaA	ab#	Pop (A)
#bAa	ab#	Pop (a), Move Forward()
#bA	b#	Pop (A)
#b	b#	Pop (b), Move Forward()
#	#	"String accepted"

Exercise 3.

1. The grammar is **not LL(1)** because it is left-recursive.

2. **Elimination of left recursion:**

The left-recursive production rules **$T \rightarrow T,S \mid S$** are replaced by: **$T \rightarrow ST'$**

$T' \rightarrow ,ST' \mid \varepsilon$

The grammar becomes: **$S \rightarrow a \mid b \mid (T)$**

$T \rightarrow ST'$

$T' \rightarrow ,ST' \mid \varepsilon$

Grammar factorization: The resulting grammar is already **factored**.

3**. Proof that the grammar obtained in step 2 is LL(1):**

Compute the **FIRST** and **FOLLOW** sets of the grammar G**:**

Non-terminal symbol	First	Follow
S	a b (	# ,)
T	a b (	)
T '	, ε	)

Proof:
For every pair of rules A→a | bA \rightarrow a \ | \ bA→a | b, we have:
FIRST(a.FOLLOW(A)) ∩ FIRST(b.FOLLOW(A)) = ∅

- For the rules **S → a | b**:
  FIRST(a.FOLLOW(S)) ∩ FIRST(b.FOLLOW(S)) = ∅
- For the rules **S → a | (T)**:
  FIRST(a.FOLLOW(S)) ∩ FIRST((T).FOLLOW(S)) = ∅
- For the rules **S → b | (T)**:
  FIRST(b.FOLLOW(S)) ∩ FIRST((T).FOLLOW(S)) = ∅
- For the rules **T' → ,ST' | ε**:
  FIRST(,ST'.FOLLOW(S)) ∩ FIRST(ε.FOLLOW(T')) = ∅

Therefore, the grammar obtained in step 2 is **LL(1)**.

<u>The grammar analysis table.</u>

	has	**b**	**(**	**)**	**,**	**#**
S	S →a	S →b	ST →)			
T	T →ST '	T →ST '	T →ST '			
T '				T '→ε	T '→,ST '	

4. **<u>The analysis of the string (a,(b,a),a) :</u>**

Stack Contents	Remaining to be analyzed	Action
#S	(a,(b,a),a)#	Replace S with)T(

#)T(	(a,(b,a),a)#	Move forward
#)T	a,(b,a),a)#	Replace T with T'S
#) T'S	a,(b,a),a)#	Replace S with a
#) T'a	a,(b,a),a)#	Move forward
#) T'	,(b,a),a)#	Replace T' with T'S,
#) T'T'S,	,(b,a),a)#	Move forward
#) T'T'S	(b,a),a)#	Replace S with)T(
#) T'T')T(	(b,a),a)#	Move forward
#) T'T')T	b,a),a)#	Replace T with T'S
#) T'T') T'S	b,a),a)#	Replace S with b
#) T'T') T'b	b,a),a)#	Move forward
#) T'T') T'	,a),a)#	Replace T' with T'S,
#) T'T') T'S,	,a),a)#	Move forward
#) T'T') T'S	a),a)#	Replace S with a
#) T'T') T'a	a),a)#	Move forward
#) T'T') T'	),has)#	Pop (T ')
#) T'T')	),has)#	Move forward
#) T'T'	,has)#	Replace T' with T'S,
#) T'T'S,	,has)#	Move forward
#) T'T'S	has)#	Replace S with a
#) T'T'a	has)#	Move forward
#) T'T'	)#	Pop (T ')
#) T'	)#	Pop (T ')
#)	#)	Move forward

#	#	**“String accepted”**

Exercise 4.

We define the following grammar G = <N,T,P,S>:

P : **S → (A) | a**

A → SB

B → ,S | S a)

1. Verify whether the grammar obtained is LL(1) or not:

- Compute the **FIRST** and **FOLLOW** sets :

	First	Following
S	(a	a (, #)
A	(a	)
B	(a ,	)

- verification :

For every pair of rules A → α | β We have:

For: **S → (A) | a**

First ((A) .FOLLOW(S)) ∩ First (a **. FOLLOW (S)) = Φ**

For: **B → ,S | S a)**

First (,S . FOLLOW (B)) ∩ First (S a **. FOLLOW (B)) = Φ**

Therefore, the grammar **is LL(1)**

2. Writing the syntax analysis algorithm that recognizes the words of the language L(G)L(G)L(G) using the **recursive descent method**.

Add the rule **Z → S#**

```
Procedure Z( )
Begin
  S( ) ;
  if tc = '#' then
     "string accepted" ;
  else "Error" ;
 end if
End
```

```
Procedure A( )
Begin
  S( ) ;
  B( ) ;
End
```

```
Procedure S( )
Begin
if tc = '(' then
       tc ← ts ;
       A( ) ;
       if tc = ')' then
         tc ← ts ;
       else "Error"
       end if
else
    if tc = 'a' then
         tc ← ts ;
    else "Error"
    end if
end if
end
```

```
Procedure B( )
Begin
 if tc = ',' then
     tc ← ts ;
     S( ) ;
 else
     S() ;
     if tc = 'a' then
         Tc ← ts ;
          if tc = ') ' then
             Tc ← ts
          else "Erreur"
          end if
       else "Error"
       end if
 end if
end
```

3. Analysis of the string: ((a,a))#

	Remaining to be analyzed	Action
Z	((a,a))#	Call(S)
ZS	((a,a))#	Move forward (), Call(A)
ZSA	(a,a))#	Call(S)
ZSAS	(a,a))#	Move forward (), Call(A)
ZSASA	a,a))#	Call(S)
ZSASAS	a,a))#	Move forward(),Return(S)
ZSASA	,a))#	Call(B)
ZSASAB	,a))#	Move forward (), Call(S)
ZSASABS	a))#	Move forward (),Return(S)
ZSASAB	))#	Return (B)
ZSASA	))#	Return (A)
ZSAS	))#	Move forward (),Return (S)
ZSA	)#	Call(B)
ZSAB	)#	Call(s)
ZSABS	)#	" Error "

3+. Analysis of the string: (a,a)#

	Remaining to be analyzed	Action
Z	(a,a)#	Call(s)
ZS	(a,a)#	Move forward (), Call(A)
ZSA	a,a)#	Call(S)
ZSAS	a,a)#	Move forward (), Return(S)

ZSA	,a)#	Call(B)
ZSAB	,a)#	Move forward(),Call(S)
ZSABS	a)#	Move forward (), Return(S)
ZSAB	)#	Return(B)
ZSA	)#	Return(A)
ZS	)#	Move forward (), Return(S)
Z	#	“Accepted string”