Exercices corrigés

L. Ben Ayed

$\{a^n b^n, n>=0\}$ est un langage non contextuel et non régulier

W0 = ε
W1 = ab = aW0b
W2 = aabb = aW1b

Wi = $a^i b^i$ = awi-1b

S □ ε
S □ aSb

```
S(n)
{  si n!=0 alors
   écrire(a) S(n-1)
écrire(b)
}
```

Exemples de grammaires et d'arbres syntaxiques

G = (V, T, S, R) V = {S}, T = {a, b}

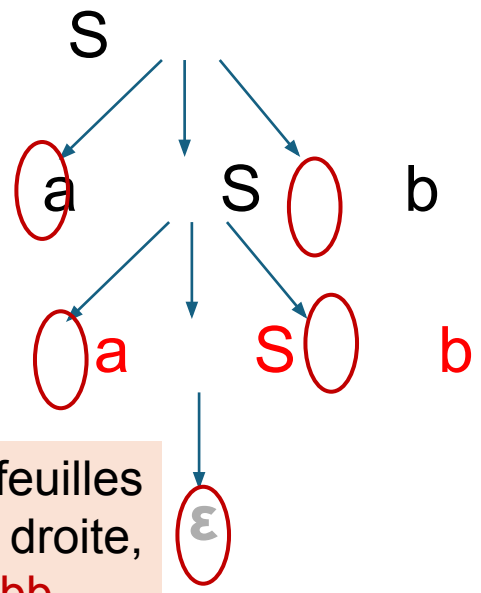
R= {S □ ε

S □ aSb }

S => aSb

=> a**aSb**b

=> a**a**ε**b**b

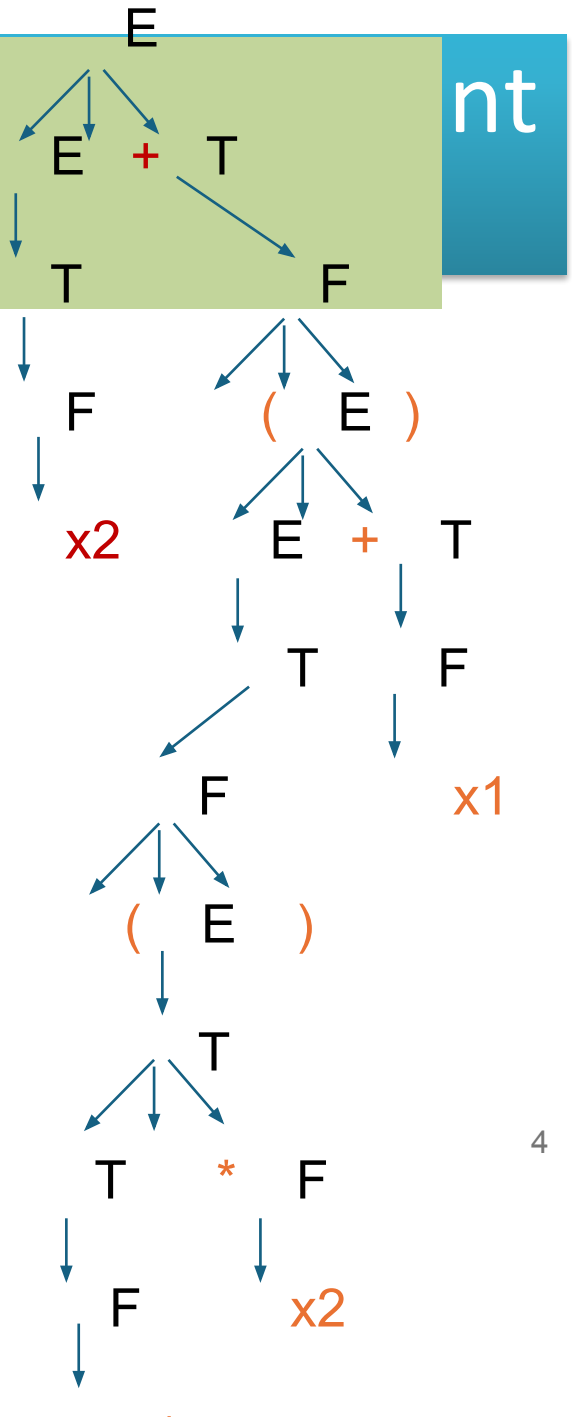


Montrer que le mot **aabb** est généré par la grammaire avec les règles de productions données ci-dessus

Lorsqu'on parcourt les feuilles de l'arbre de gauche à droite, nous trouvons le mot aabb.
Ainsi la grammaire génère aabb
aabb est syntaxiquement correcte
aa**ε**bb = aabb
ε Est l'élément neutre de la

La nt

G génère la chaine x2 + ((x1*x2)+ x1)
par la dérivation suivante



Exemple :

R = { E → E + T (1)
E → T (2)
T → T *F (3)
T → F (4)
F → (E) (5)
F → x1 (6)
F → x2 (7)
}

G = (V, X, E, R)
V = {T, F, E}
X = {x1, x2, +, *, (,)}

La nt

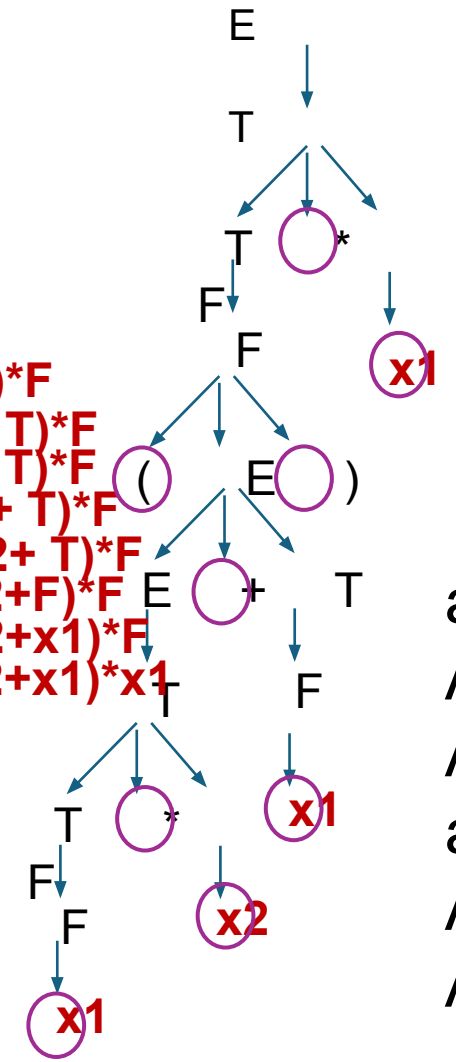
G génère la chaine **(x1*x2+x1)*x1** par la dérivation suivante

Exemple :

R = {
- **E → E + T (1)**
- **E → T (2)**
- **T → T *F (3)**
- **T → F (4)**
- **F → (E) (5)**
- **F → x1 (6)**
- **F → x2 (7)**

}

E => T
=>
=>
=> (E)*F
=> (E+
=> (T+ T)*F
=> (T*F+ T)*F
=> (F*F+ T)*F
=> (x1*F+ T)*F
=> (x1*x2+ T)*F
=> (x1*x2+F)*F
=> (x1*x2+x1)*F
=> (x1*x2+x1)*x1



G = (V, X, E, R)
V = {T, F, E}
X = {x1, x2, +, *, (,)}

a*
A □ ε
A □ aA
a+
A □ a
A □ aA

La du contexte nt

G génère la chaine **(x1*(x2+x1))*x1** par la dérivation suivante

Exemple :

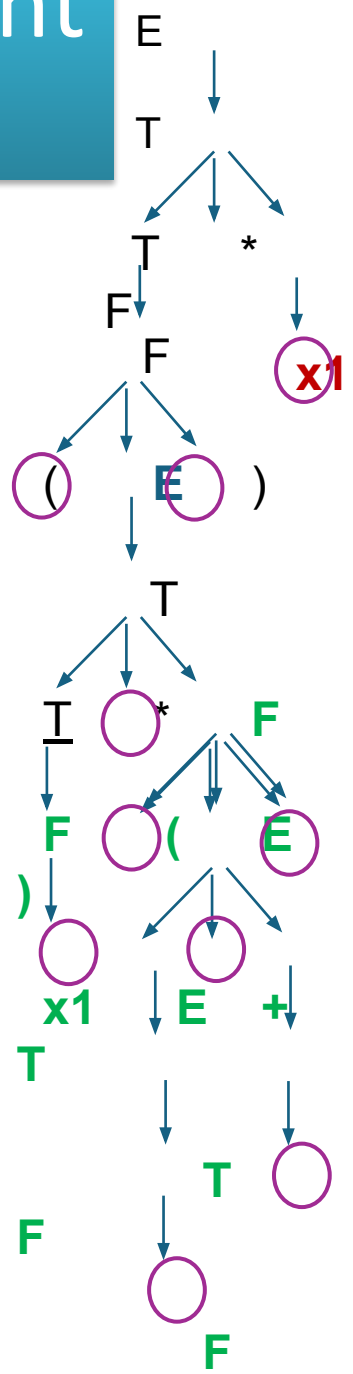
R = {
- **E → E + T (1)**
- **E → T (2)**
- **T → T *F (3)**
- **T → F (4)**
- **F → (E) (5)**
- **F → x1 (6)**
- **F → x2 (7)**

}

G = (V, X, E, R)

V = {T, F, E}

X = {x1, x2, +, *, (,)}



Langage hors contexte ou indépendant du contexte

Exemple :

R = { **P → Begin L_I end. (1)**
L_I → I (2)
L_I I L_I (3)
I → id := E; (4)
E → E op E (5)
E → id (6)
E → nb (7)
}

G = (V, X, **P**, R)
V = { P, L_I, I, E}
T = {Begin, end, ., id, :=, ; ,op, nb}
Montrer que le mot m suivant est généré par G:
m : Begin id := id op nb; id := nb; end.

R = { **P → Begin L_I end.**
L_I → I | I L_I
I → id := E;
E → E op E | id | nb
}
Op représente + ou * ou/ ou -

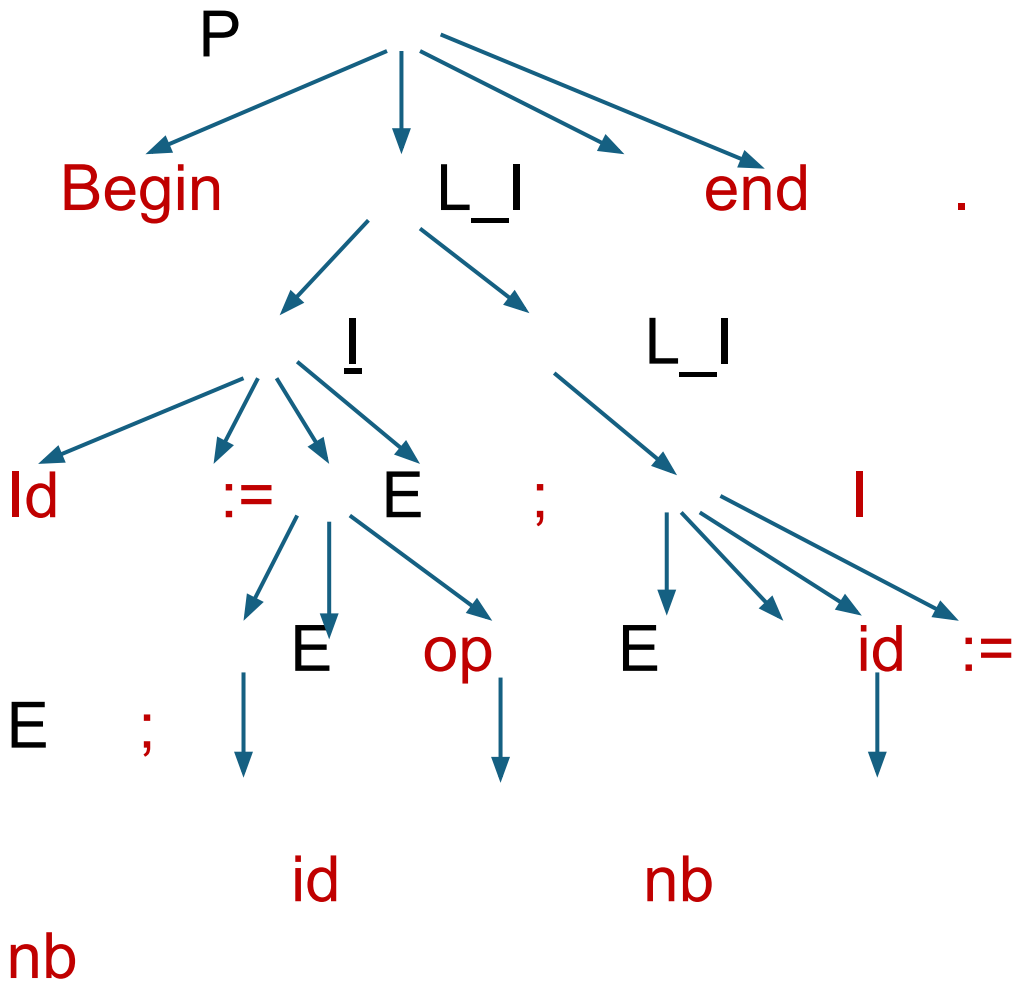
a* A e
A a A
a+ A a
A aA
L_I I
L_I I L_I

Exemple :

R = { **P → Begin L_I end. (1)**
L_I → I (2)
L_I I L_I (3)
I → id := E; (4)
E → E op E (5)
E → id (6)
E → nb (7)
}

Construire un arbre syntaxique pour le mot suivant:

Begin id := id op nb; id := nb; end.



Exercice1 :

Soit la grammaire G avec les règles de production suivantes

R = {

P → Begin L_I end. (1)

L_I → I (2)

L_I I L_I (3)

I → id := E; (4)

E → E op E (5)

E → id (6)

E → nb (7)

}

1. Donner de résultat de l'Analyseur lexical sur le mot suivant:

Begin a:= b1+ 12; b1:=20; end.

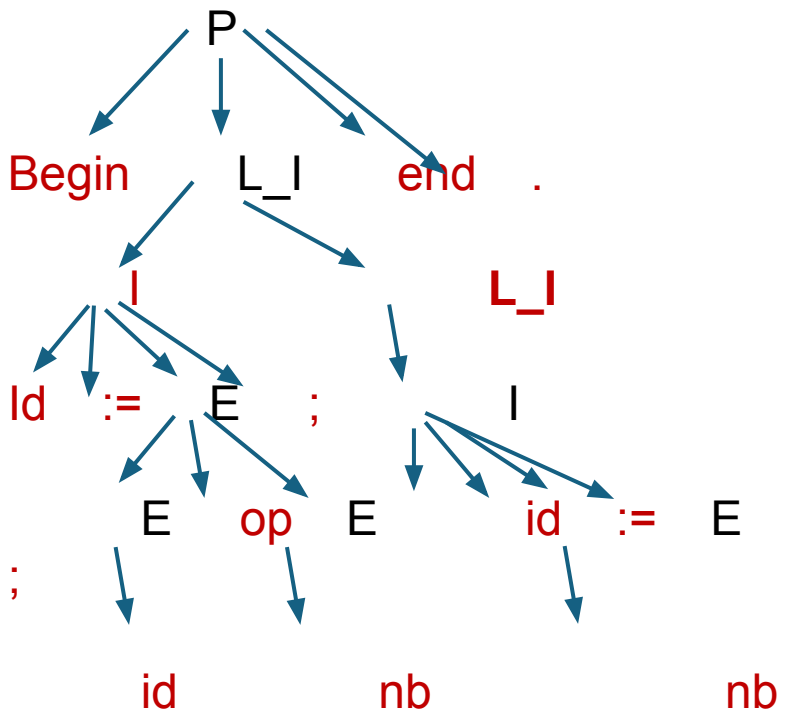
2. Montrer que le résultat retourné par l'analyseur lexical est syntaxiquement valide.

1. Begin a := b1 + 12; b1 := 20; end.

Devient

Begin id := id op nb; id := nb; end.

2.



Exercice 1 : terminaux modifiés

Soit la grammaire G avec les règles de production suivantes

Aff à la place de := et pv à la place de ;

R = {

P → Begin L_I end. (1)

L_I → I (2)

L_I I L_I (3)

I → id aff E pv (4)

E → E op E (5)

E → id (6)

E → nb (7)

} aff représente :=

Pv représente ;

1. Donner de résultat de l'Analyseur lexical sur le mot suivant:

Begin a:= b1+ 12; b1:=20; end.

1. Begin a := b1 + 12; b1 := 20; end.

Devient

Begin id aff id op nb pv id aff nb pv end.

Exercice 2.

Soit la grammaire G avec les règles de production suivantes:

I→ si B alors I sinon I | id := E;
B → E opr E | non B | B et B | B ou B
E → nb| id | E opa E | (E)

1. Donner l'ensemble des terminaux (unités lexicales de G).
2. Donner le résultat de l'AL sur le mot suivant:
si b>10 alors si a>b alors a :=b; sinon b:=0; sinon a :=0 ;
3. Montrer que le résultat obtenu en 2 est syntaxiquement valide (généré par G).

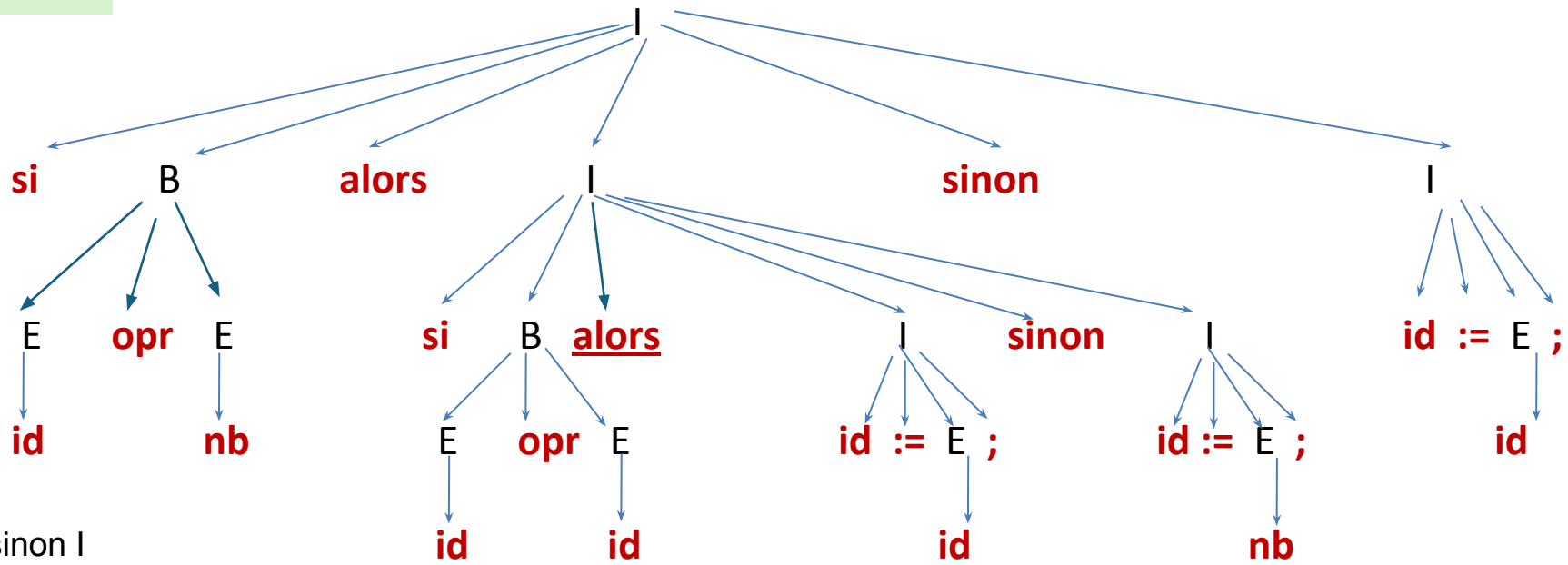
1.
T = {si, alors, sinon, id, nb, := ,;,
opr, non, et, ou, opa, (,)}
2.
si id opr nb alors si id opr id alors
Id := id ; sinon id := nb; sinon id := nb;

Exercice 2.

3. Montrer que le résultat obtenu en 2 est syntaxiquement valide (généré par G).

I→ si B alors I sinon I | id := E;
B → E opr E | non B | B et B | B ou B
E → nb| id | E opa E | (E)

si id opr nb alors si id opr id alors Id := id ; sinon id := nb; sinon id := nb;



> si B alors I sinon I
> si E opr E alors I sinon I
> si id opr E alors I sinon I
> si id opr nb alors I sinon I
> si id opr nb alors si B alors I sinon I sinon I
> si id opr nb alors si E opr E alors I sinon I sinon I
> si id opr nb alors si id opr E alors I sinon I sinon I
> si id opr nb alors si id opr id alors I sinon I sinon I
> si id opr nb alors si id opr id alors id := E; sinon I sinon I
> si id opr nb alors si id opr id alors id := id; sinon I sinon I
> si id opr nb alors si id opr id alors id := id; sinon id:= E; sinon I
> si id opr nb alors si id opr id alors id := E; sinon id := nb; sinon I
> si id opr nb alors si id opr id alors id := E; sinon id := nb; sinon id := E;
> si id opr nb alors si id opr id alors id := E; sinon id := nb; sinon id := nb;

Exercice 3.

Construire une grammaire qui génère des séquences (non vides) d'instructions d'affectation.

G = (V, T, S_Inst, R)
V = {S_Inst, Inst, Exp}
T = {id, :=, ;, +, *, -, /, (,), nb}

R = { S_Inst □ Inst | Inst S_Inst
    Inst □ id := Exp;
    Exp □ Exp + Exp | Exp * Exp | Exp – Exp | Exp / Exp | (Exp)| id |nb

}
Id : l(l+c)*
nb:(c)+
l est une lettre alphabétique et c est un chiffre

Donner un pseudo code pour l'AS

I→ si B alors I sinon I | id := E;

B → E opr E | non B | B et B | B ou B

E → nb| id | E opa E | (E)

description	Non	Code
si	si	1
alors	alors	2
L(l+c)*	id	3
:=	aff	4
;	pv	5
<, <=, >, >=, =, <>	opr	6
Non	non	7
Et	et	8
C+	nb	9
+, *, -, /	opa	10
(	po	11
)	pf	12

```
    Si symbole = t alors symbole := symbole_suivant();
Proc I()
{
  si symbole = si alos
          accepter(si); B(); accepter(alors); I();
          accepter(sinon) ; I();
  sinon si symbole = id
            alors  accepter (id); accepter(aff); E();
            accepter(pv);
           sinon écrire(erreur si ou id manquants)
}

Proc E()
{
  Si symbole = nb) alors accepter(nb)
   sinon si symbole = id alors accepter(id)
    sinon si symbole = po alors
                                    accepter(po);
                                    E(); accepter(pf);
  sinon
     si symbole <> nb et symbole <> id et symbole <>
po alors écrire (erreur id ou nb ou ( manquante);
```