pour faire un langage besoin : - d'un alphabet
- d'une grammaire
- de la sémantique

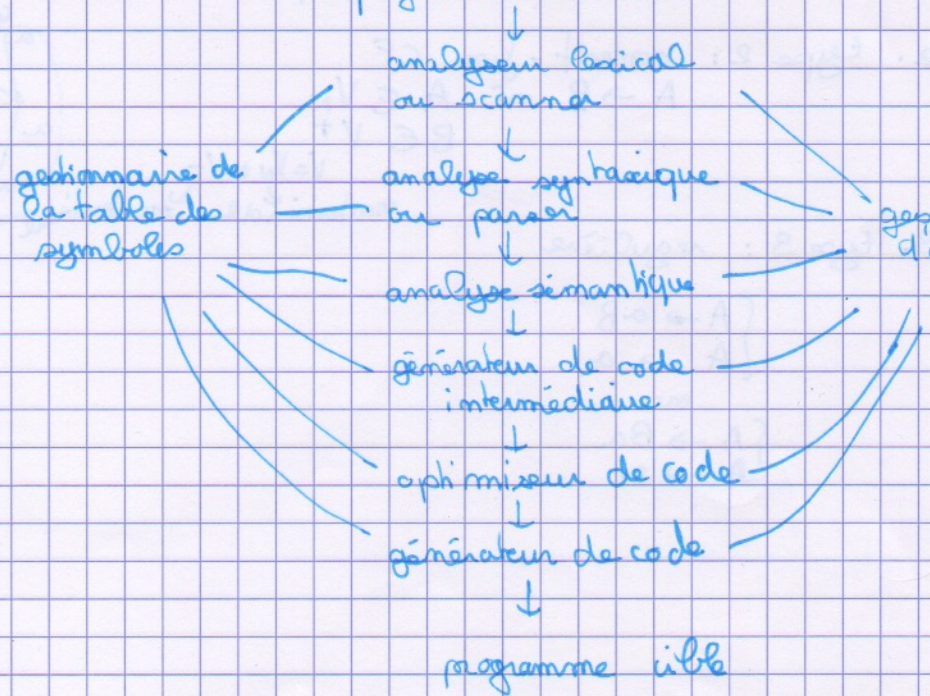
• Définition simplifiée : un compilateur est un programme qui lit un prgm écrit dans un premier langage source et le traduit en un prgm equivalent écrit dans un autre langage - le langage cible

• deux parties : - analyse : construit une représentation intermédiaire
- synthèse : construit le pgm cible à partir de 5

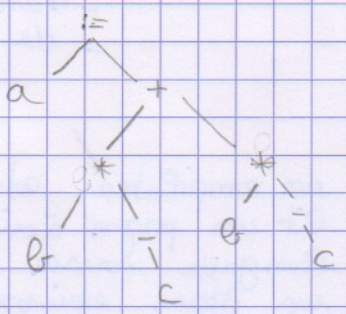
• Analyse : ① analyse linéaire = analyse lexicale
② " hiérarchique = " syntaxique
③ " sémantique

① décompose les instructions en symbole, signe, identificateur, ... les blancs sont éliminés

```
y := 234A      ← erreur lexicale ①
y := (3+4*2    ← OK - erreur syntaxique ②
boolean B
real R         → OK ①② sauf ③
R := B
```

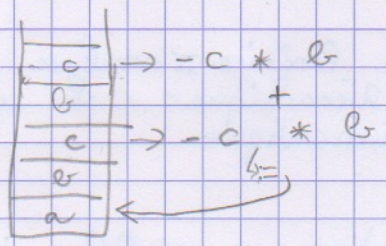


a := b * - c + b * - c



notation polonaise (postfixée) : pas d'enfant, on affiche, sinon on regarde les enfants.

a b c moins U * b c moins U * + affecter



Rappel classif grammaire

pas type 3 mais ok les autres

0. type 0 : aucune contrainte ni à gauche ni à droite

1. type 1 : context-sensitive CS
$\alpha \to \beta$ si $|\alpha| \leq |\beta|$

2. type 2 : context-free CF
$A \to B$ où $A \in V_N$
$B \in V^+$
$V = V_N + V_T$
non terminaux / voc. terminal

3. type 3 : régulière

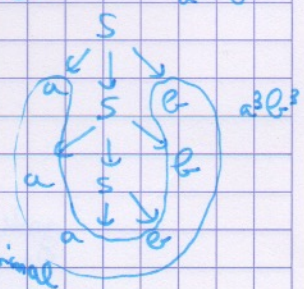
$\begin{cases} A \to aB \\ A \to a \end{cases}$

ou

$\begin{cases} A \to Ba \\ A \to a \end{cases}$

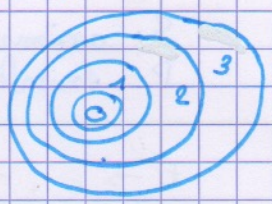
$V_N = \{S\}$
$V_T = \{a, b\}$

Grammaire G1 $\begin{cases} S \to aSb \\ S \to ab \end{cases}$ $a^n b^n$



$L(G) = \{x \in V_T^* \mid S \overset{*}{\Rightarrow} x\}$

$L(G_1) = \{x \in V_T^* \mid S \overset{*}{\Rightarrow} a^n b^n\}$



$L_1 = \{a^m b^n c^p \mid m \geq 1, p \geq 0\}$
$L_2 = \{a^m b^q c^q \mid q \geq 1, m \geq 0\}$

- Donner les grammaires G_1 et G_2 correspondant à L_1 et L_2
- Trouver $L(G) = L_1 \cap L_2$

G_1 :
- $N \to SA$
- $S \to aSb$
- $S \to ab$
- $A \to cA$
- $A \to c$
- $A \to \varepsilon$

G_2 :
- $S \to aS$
- $S \to A$
- $A \to bAc$
- $A \to bc$

$L(G) = \{a^m b^m c^n \mid m \geq 1\}$ CF type

conv : G_1 :
- $S \to A \mid Sc$
- $A \to aAb \mid ab$

type 2 conv

G_2 :
- $S \to B \mid aS$
- $B \to bBc \mid bc$

type 2 conv

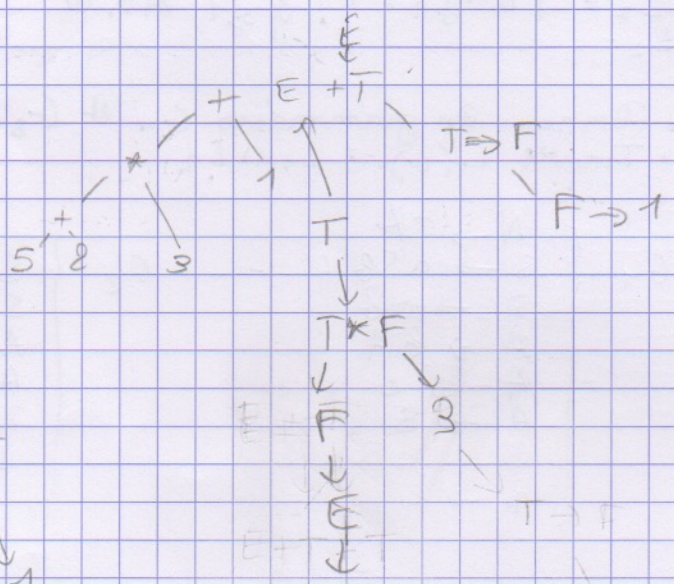
Chapitre II

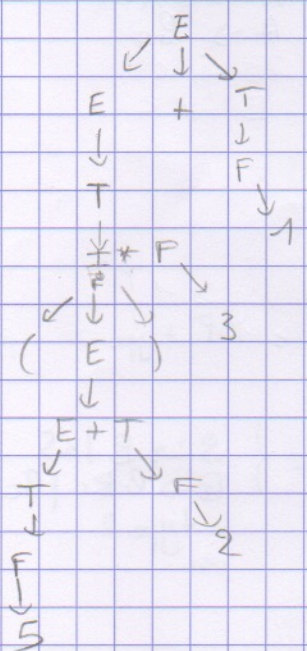
$G = (V_T, V_N, S, P)$

S : axiome
P : ensemble des règles de réécriture

$E \to E + T \mid T$ $\quad F \to (E) \mid 0 \ldots 19$
$T \to T * F \mid F$

$(5+2) * 3 + 1$





analyseur LL
(Left - Right)

(Left - Right)
↓
parcours des règles

analyseur descend.
↓
. part du start symbole

LR — analyseur desc
↓
. remonte jusqu'au start symbole

$S \to +A \mid -A \mid +G \mid -G$

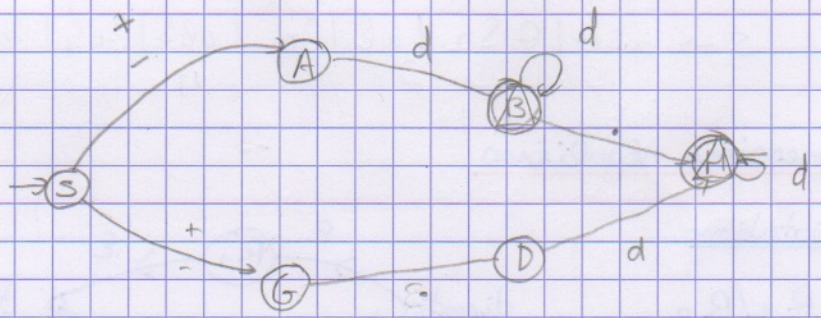
$A \to dB$

$B \to dB \mid .H \mid \Lambda$

$G \to .D$

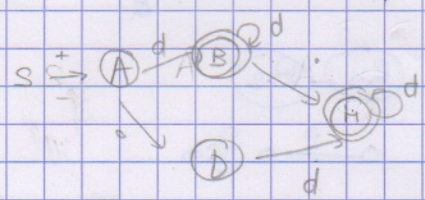
$D \to dH$

$H \to dH \mid \Lambda$



$\delta(S,+) = \{A, G\}$

$\delta(S,-) = \{A, G\}$



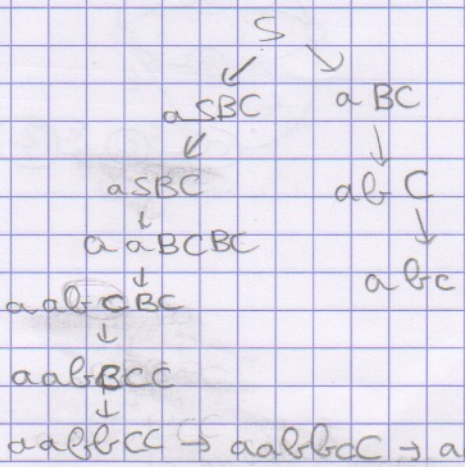
représentation matrice

TD:

$S \to aSBC$ (1)
$S \to aBC$ (2)
$CB \to BC$ (3)
$aB \to ab$ (4)
$bB \to bb$ (5)
$bC \to bc$ (6)
$cC \to cc$ (7)

type G ? type 1
$L(G)$?

$L(G) = \{a^n b^m c^n / m \geq 1\}$



$S \overset{n-1}{\Rightarrow} a^{n-1} S(BC)^{n-1}$

$\overset{(1)}{\Rightarrow} a^{n-1} B(BC)^{n-1} C$

$\overset{n-1\ (2)}{\Rightarrow} a^{n-1} a B^n C^n$

$\overset{(3)}{\Rightarrow} a^{n-1} a b B^{n-1} C^n$

$\overset{(4)}{\Rightarrow} a^n b^n C^n$

$\overset{n-1\ (5)}{\Rightarrow} a^n b^{n-1} bc C^{n-1}$

$\overset{(6)}{\Rightarrow} a^n b^n c^n$

$\overset{n-1\ (7)}{\Rightarrow}$

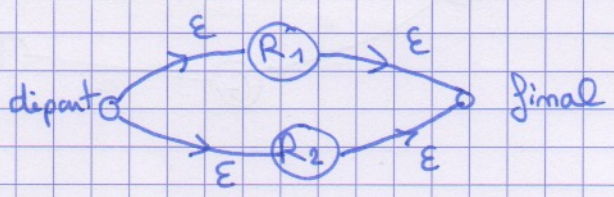
$L(G) = \{ w \mid |w|_a = |w|_b , |w| \geq 2 \}$

$S \to aSb \mid bSa \mid ab \mid ba \mid abS \mid baS \mid Sba \mid Sab$

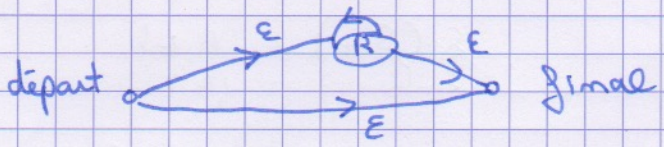
Expressions régulières

Notations

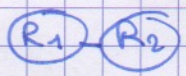
R_1 / R_2



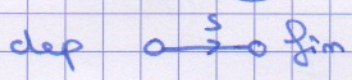
R^*



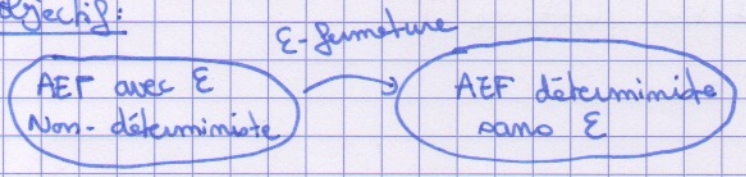
$R_1 R_2$



S



Objectif :



Exemple $(a|b)^* abb$

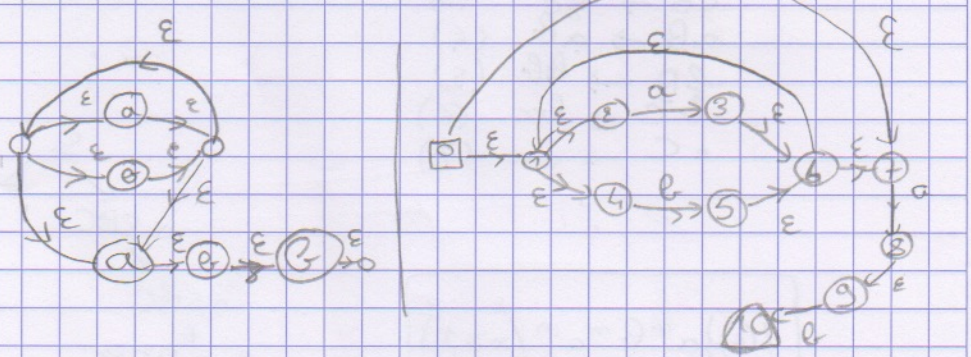
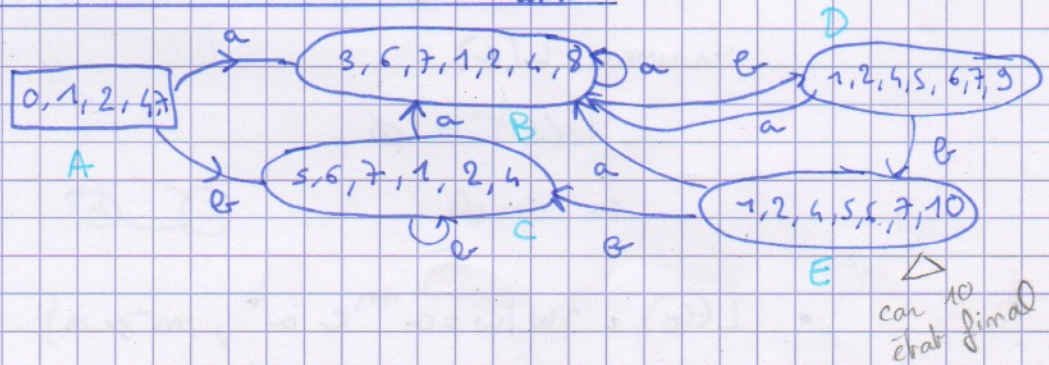


table d'etats

	ε	a	e
0	1,7	—	—
1	2,4	—	—
2	—	3	—
3	6	—	—
4			
⋮			

construction de l'AEF déterministe sans ε

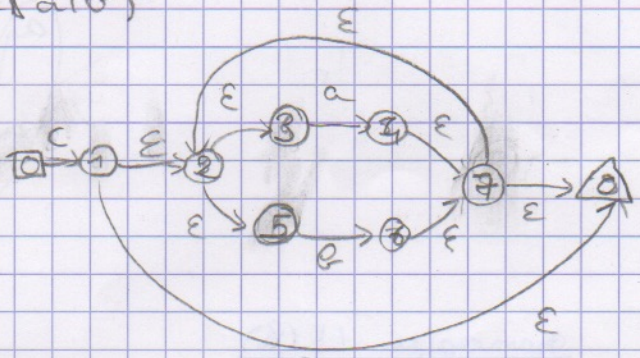


	a	e
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

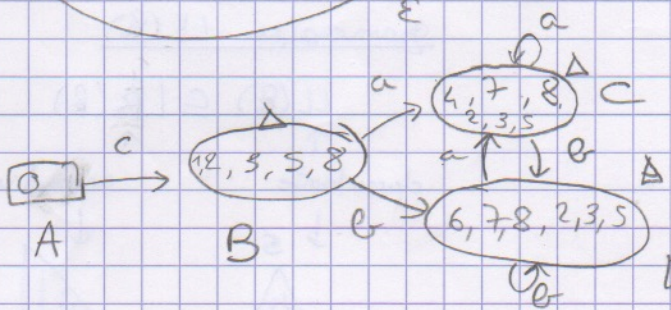
Exemple 2: c(a|e)*

table d'etat

	a	e	c	ε
0	—	—	1	—
1	—	—	—	2,8
2	—	—	—	3,5
3	4	—	—	—
4	—	—	—	7
5	—	6	—	—
6	—	—	—	7
7	—	—	—	2,8



	a	e	c
A	—	—	B
B	C	D	—
C	C	D	—
D	C	D	—



$L(G) = \{ w / w = a^p b^q c^r , p+q \geq r \}$

trouver la grammaire G correspondante

type 2

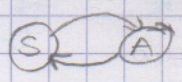
$S \to aSc \mid aAc \mid aS \mid \varepsilon \mid A$
$A \to bAc \mid bc \mid bA \mid e$

- Soit G: $S \to aAB$
$B \to ab \mid SA$
$bB \to a$
$Aa \to SaB$
$Ab \to SBb$

trouver $L(G)$

$L(G) = \emptyset$

la boucle

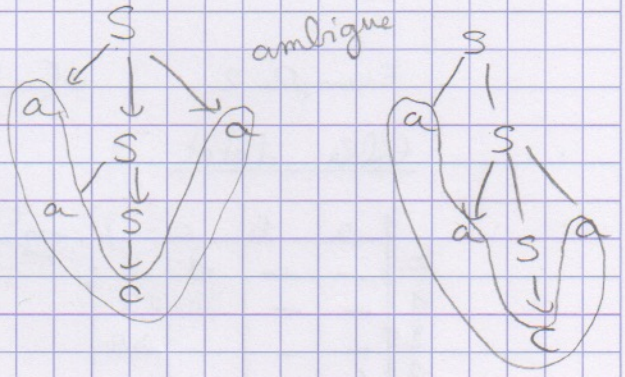


- $L(G) = \{ w / w = a^m c a^n , m \geq n \}$

Trouver G?

$S \to aSa \mid aS \mid c$ aaca

$S \to aSa \mid aA$
$A \to aA \mid c$

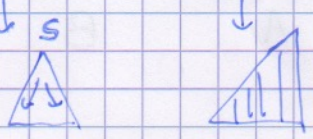


Grammaire LL(k)

$LL(k) \subset LR(k)$ k = profondeur

descendante ↓ S

ascendante ↓



Exo: type 2

$S \to U\$$
$U \to aAB$
$A \to cA$
$A \to c$
$B \to bB$
$B \to b$

$ac^m b^m \$$
$m, m \geq 1$

$S \Rightarrow U\$$
$\Rightarrow aAB\$$
$\Rightarrow acAB\$$
$\Rightarrow accB\$$
$\Rightarrow accb\$$

LL(1) = aucune ambiguité pour appliquer les règles.

First (Premiers) et Follow (suivants):

FirstA(N):

1) Si $N \to A \ldots$ alors First(N) = First(A)
2) Si $N \to c \ldots$ alors First(N) = {c}
3) Si $N \to A.B.c$ et si $A \Rightarrow \varepsilon$
   Alors First(N) = First(B)

Ex1: $S \to E\$$
$E \to E+T$
$E \to T$
$E \to T*F$
$T \to F$
$F \to (E)$
$F \to id$

First(S) = First(E) = First(T) =
First(F) = { (id }

Exo 2:

② ↑ ↗ ③②

$M \to Ad/e$
$A \to b^* c^*$

First(A) = { b c }

First(M) = { First(A) d e }
= { b c d e }

Follow:

1) Si $A \to \ldots Nc \ldots$ Alors Follow(N) = {c}
2) Si $A \to \ldots NB \ldots$ Alors Follow(N) = First(B)
3) Si $A \to \ldots N$ Alors Follow(N) = Follow(A)

Follow Ex 1:

Follow(E) = { $ +) }
Follow(T) = { * follow(E) } = { * $ +) }
Follow(F) = { follow(T) } = { * $ +) }
& E

mais pas besoin de le mettre car déjà dans Follow(T)

Grammaire LL(1):

1) $A \to \alpha_1 | \alpha_2 | \dots | \alpha_n$ $\quad First(\alpha_i) \cap First(\alpha_j) = \phi$ pour $i \neq j$

2) $A \overset{*}{\Rightarrow} \varepsilon$ on doit avoir $\quad First(A) \cap Follow(A) = \phi$

Exo 3:

$S \to Aa$
$A \to b^* | a$ $\quad$ Est-elle LL(1) ?

$First(A) \cap Follow(A) = \phi$

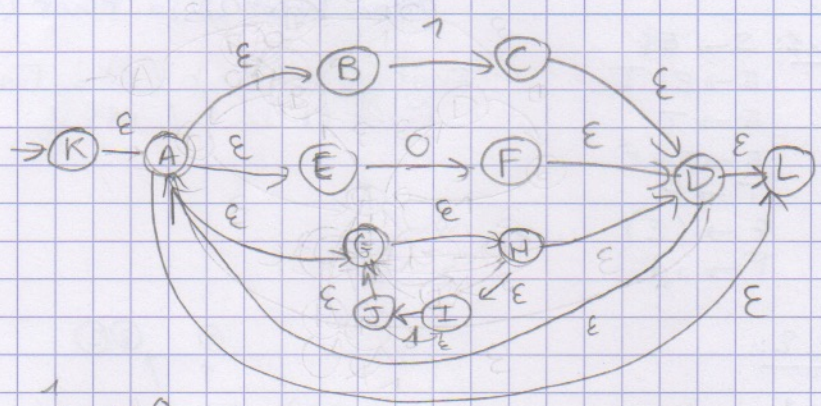
$First(A) = \{ b, a \}$
$Follow(A) = \{ a \}$

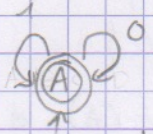
donc Non LL(1)

Toutes les grammaires récursives à gauche ne sont pas LL(1).

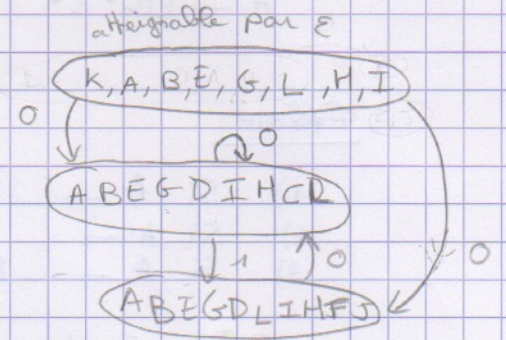
EX:

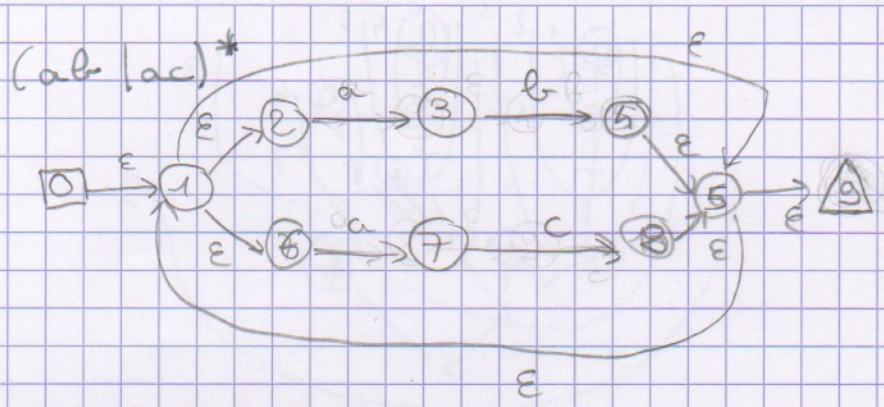
$(0|1|1^*)^*$ $\quad$ AEF avec ε-fermeture

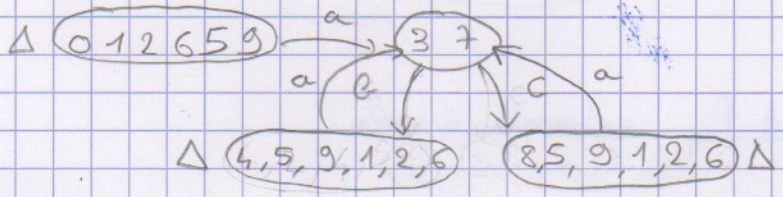


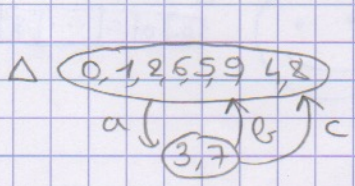


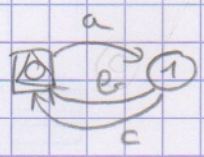
	ε	0	1
A	B,E,G,L	—	—
B	—	—	C
C	D	—	—
D	L,A	—	—
E	—	F	—
F	D	—	—
G	H	—	—
H	I	—	—
I	—	—	J
J	G	—	—
K	A	—	—
L	—	—	—







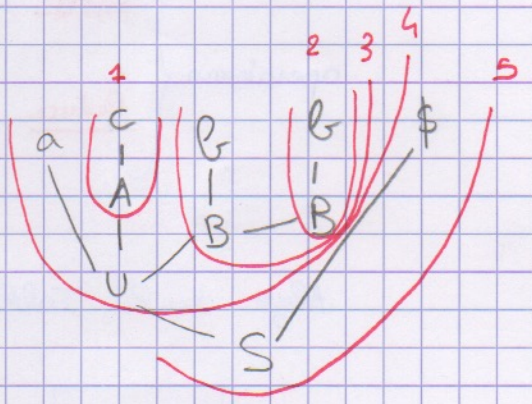




Grammaire LR(1):

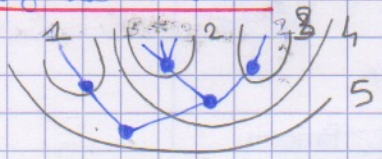
S → U$ ⇒ aAB$
⇒ aAbB$
⇒ aAbb$
⇒ acbb$

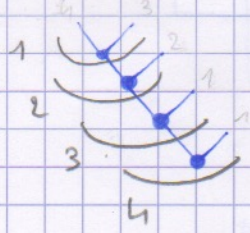
S → U$
U → aAB
A → cA
A → c
B → bB
B → b



Handles = Poignées
⇒ reconnaissance d'une règle

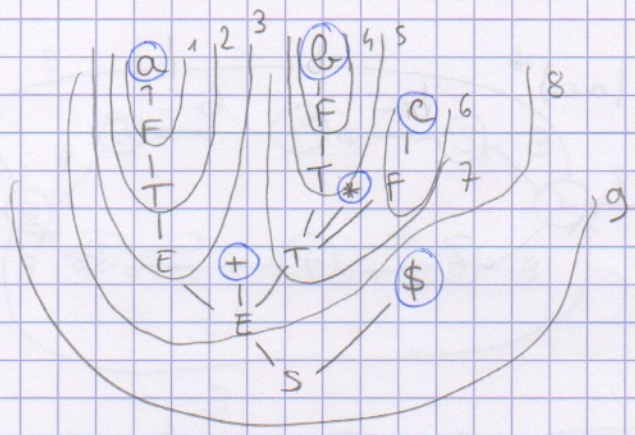
Def des Handles:



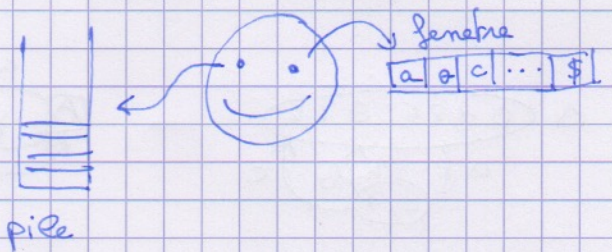


S → E$
E → E+T
E → T
T → T*F
T → F
F → (E)
F → id

construire arbre et poignées de a+b*c$

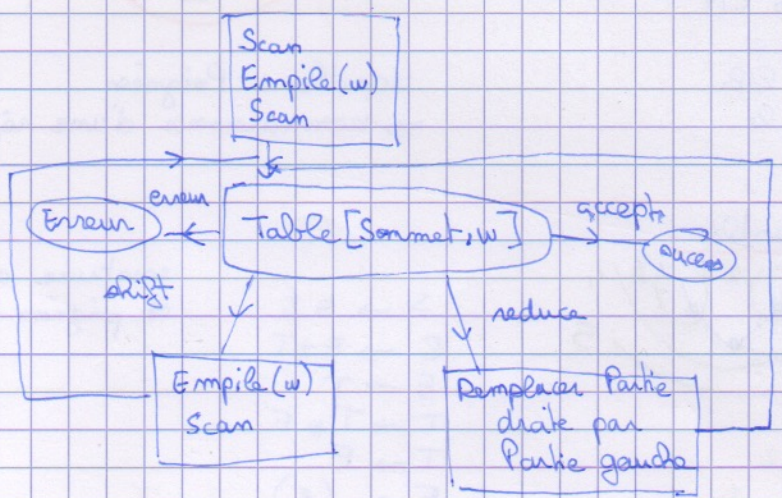


A.A.P : automate à pile



operations
- **Shift** → Empiler le caractère + Scan
- **Reduce** → Remplacer la partie droite par la partie gauche

Algo de la Table S-R



Exc : $S \to A\$$
$A \to aAb$ $\quad a^n c b^n \$$
$A \to c$

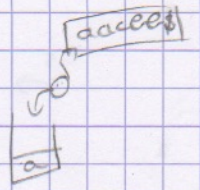
aacBb$

Table d'analyse SR

pile \ entrée	a	b	c	$
a	S	e	S	e
b	e	R	e	R
c	e	R	e	R
A	e	S	e	A

↑ pour savoir ou mettre reduce regarder la pile

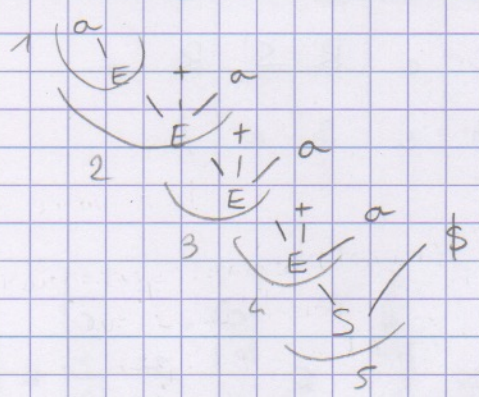
S = shift
R = reduce
e = Error
A = accept



Ex: S → E$
E → E+a
E → a

1) a+a+a+a$ (arbre + Handles)
2) table SR
3) config de la pile

1)



2)

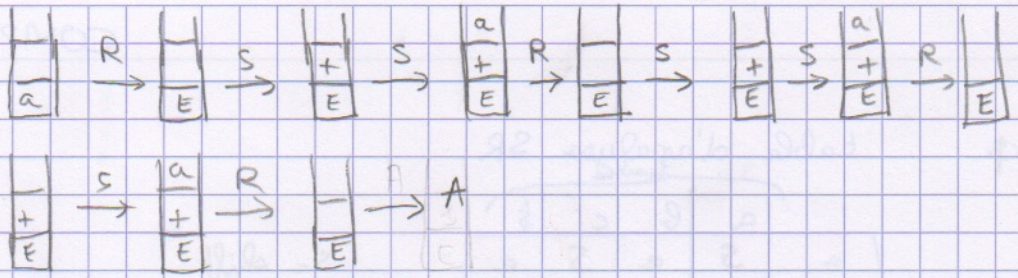
	a	+	$
a	e	R	R
+	S	e	e
E	e	S	A

reduce = remplacer partie droite règle par partie gauche

3)

[a]	a	a+a+a+a
R [E]		
S [E+]	a+	
S [E+a]	a+a	
R [E]	a+a	
S [E+]	a+a+	
S [E+a]	a+a+a	
R [E]	a+a+a	
S [E+]		

S [E+a]	a+a+a+a
R [E]	a+a+a+a
A [E$]	a+a+a+a$

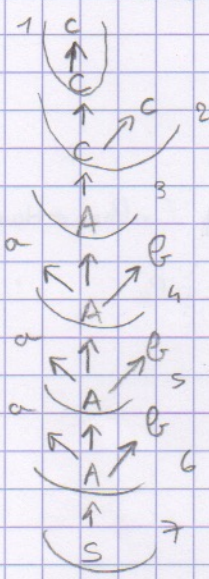


partypes car (2)
type 2 car
règles de
car # elem
à gauche $\in V_N$

$S \to A\$$
(2) $A \to aAb$
$A \to C$
$C \to Cc$
$C \to c$

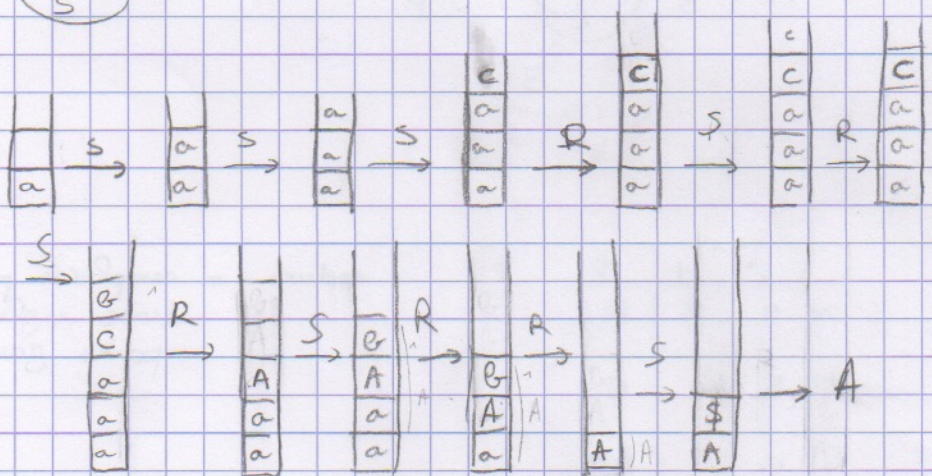
aaacc~~bbb~~$

1) arbre + handles
2) table SR
3) pile



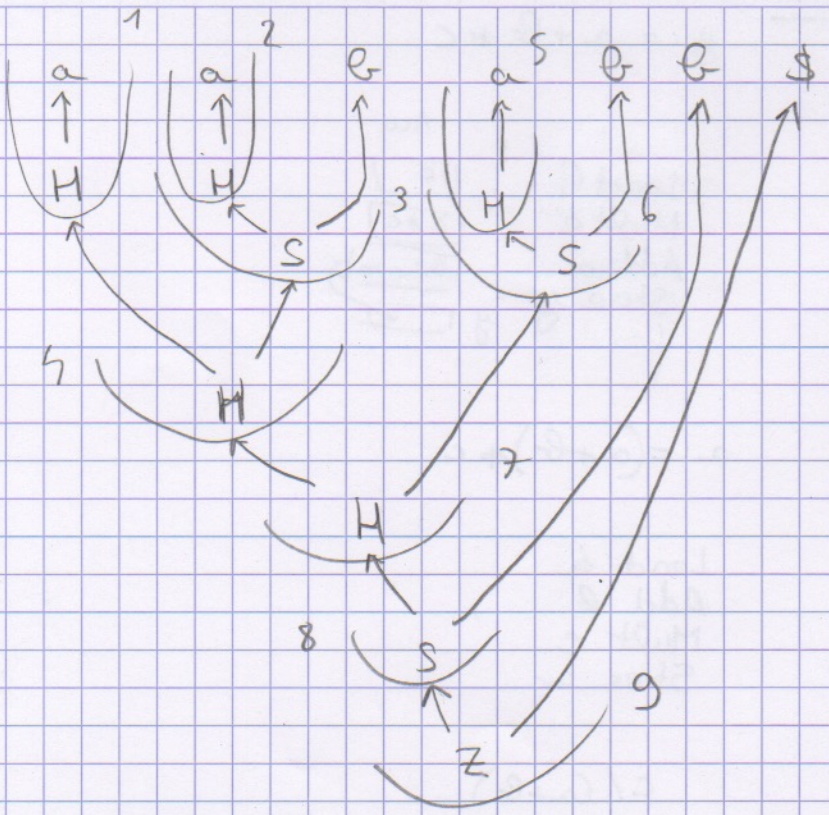
	a	b	c	$
a	S	e	S	e
b	e	R	e	R
c	e	R	R	R
C	e	R	S	R
A	e	S	e	A

quand on peut faire S et R tu fait R



$Z \to S\$$
$H \to HS$
$H \to a$
$S \to Hb$

aabaBb$



	a	e	$
a	R	R	e
e	R	R	R
H	S	S	e
S	R	R	A

I) Génération de code :

1) Mnémoniques associés à un ACCU

ACC

LOAD	A	prend contenu de l'ACCU et le met dans A
STO	A	
ADD	B	prend contenu Accu + contenu add B et le met dans l'Accu
SUB	D	soustrait D à la valeur de l'Accu, met le résultat dans l'Accu

Ex 1 :

$y := a + b * c$

	Acc
Load b	b
Mult c	b*c
Add a	b*c+a
Store y	y ← b*c+a

$a := (a+b) * c$

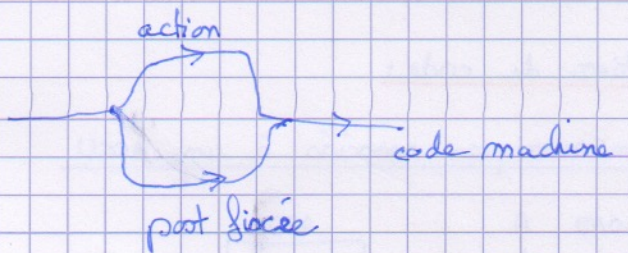
```
Load a
Add b
Mult c
Store a
```

c / (a+b)

```
Load a
Add b
Store d
Load c
div d
```

Pour toutes les opérations non commutatives ie faut passer par des variables intermédiaires.

c) Automatisation de ce processus :



$c/(a+b) \Rightarrow c\ a\ b\ +\ /$

II) Generation de code avec plusieurs registres

1) Opérations

mov { R / A } , { R / A }

R : registre
A : adresse

source — destination

Exemple :

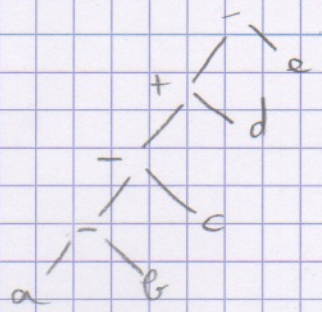
Add $R_1, R_2 \Leftrightarrow R_2 \leftarrow R_2 + R_1$

2) Combien a-t-on besoin de Registre R_i ?

$$(((a-b)-c)+d)-e$$

[] = @

```
mov [a], R1
sub [b], R1
sub [c], R1
add [d], R1
sub [e], R1
```

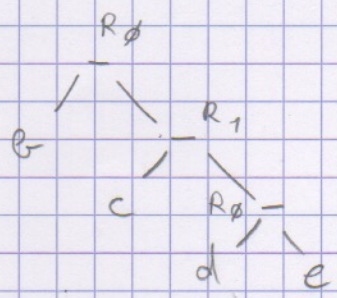


1 registre

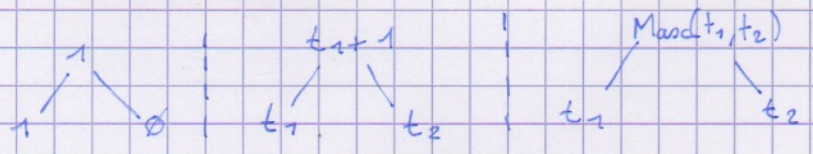
Ex 2:

$$b-(c-(d-e))$$

```
mov [d], Rø
sub [e], Rø
mov [c], R1
sub Rø, R1
mov [b] Rø
sub R1, Rø
```



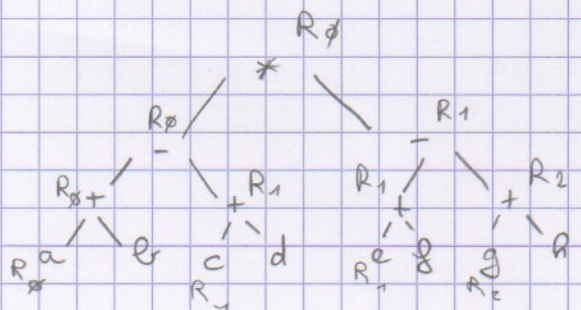
Règle générale :



$t_1 = t_2$ $\quad$ $t_1 \neq t_2$

$$((a+b)-(c+d)) * ((e+f)-(g+h))$$

3 registres




```
mov [a], Rø      sub R1, Rø      add [h], R2
add [b], Rø      mov [e], R1     sub R2, R1
mov [c], R1      sub [f] R1      mult R1, Rø
add [d], R1      mov [g] R2
```