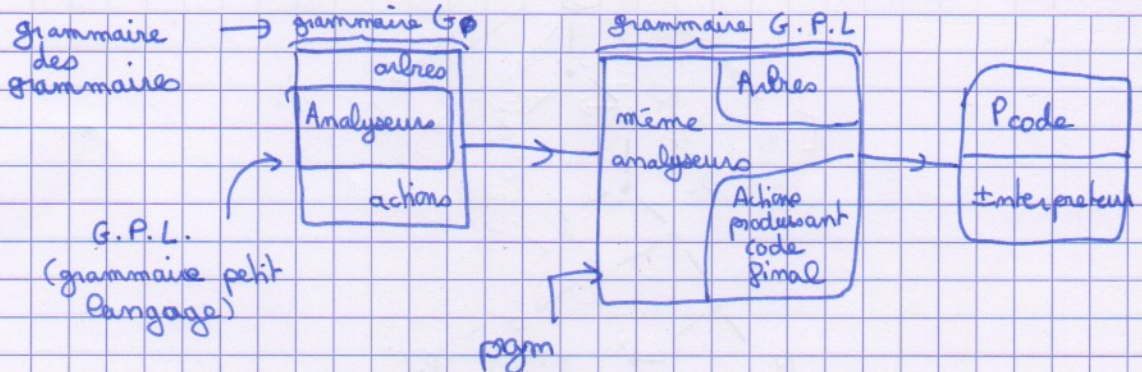
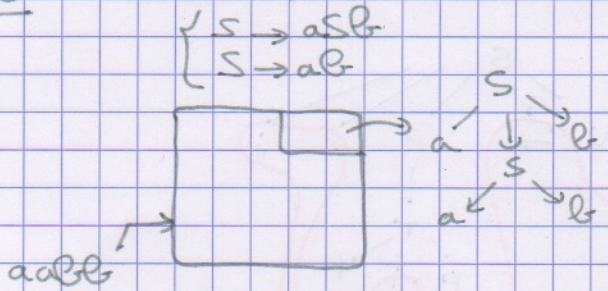


ex :



Notation BNF :

1°) ::= $\Leftrightarrow$ '→' (S::= aSb S→"aSb")

2°) [x] $\Leftrightarrow$ $\underbrace{x.x\ldots x}_{m \text{ fois}}$ $m \geq 0$

3°) (/x/) $\Leftrightarrow$ x ou vide

4°) / $\Leftrightarrow$ + ou

5°) concatenation $\Leftrightarrow$.

6°) Pour différencier terminaux et nonterminaux, "Terminaux"

Grammaire des grammaires G0 :

Regles type 2

R1 S → [N.'→'.E.';'].';', définit la forme des règles

R2 N → 'IDNTER'

R3 E → T.['+'.T] un terme ou plusieurs

R4 T → F.['.'.F]

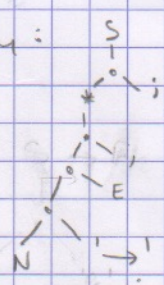
R5 F → 'IDNTER' + 'ELter' + '('.E.')' + '['.E.']' + '(/'.E.'/)';

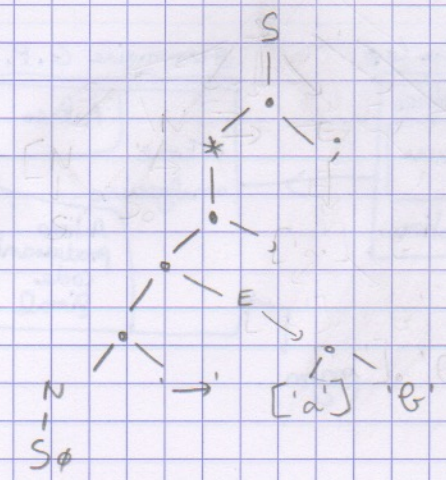
IDNTer = identificateur non terminal

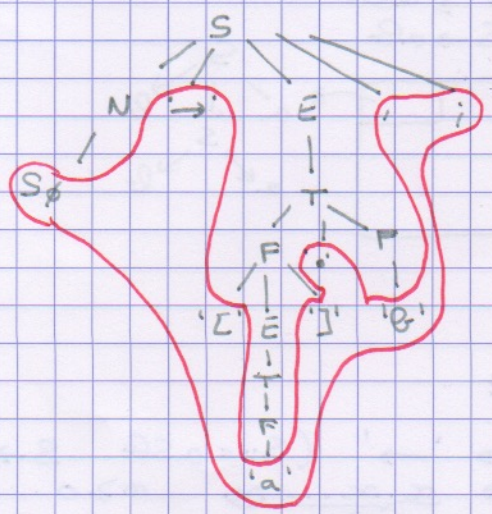
facteur

montrer que G1 ∈ G0 :

G1 } Sϕ → ['a'].'b',;







Inst → 'if' . N . 'then' . I . (/'else' . I /) . ';' . ';',
N → 'true' + 'false' + E .
I → 'ins'

aef=
gram reg

Inst → 'if' . EB . 'then' . Inst . (/ 'else' . Inst /),
EB → PG . '=' . PD,
→ P1 . 'et' . P2,
⋮
→ 'for' ...
→ 'repeat' ...
⋮

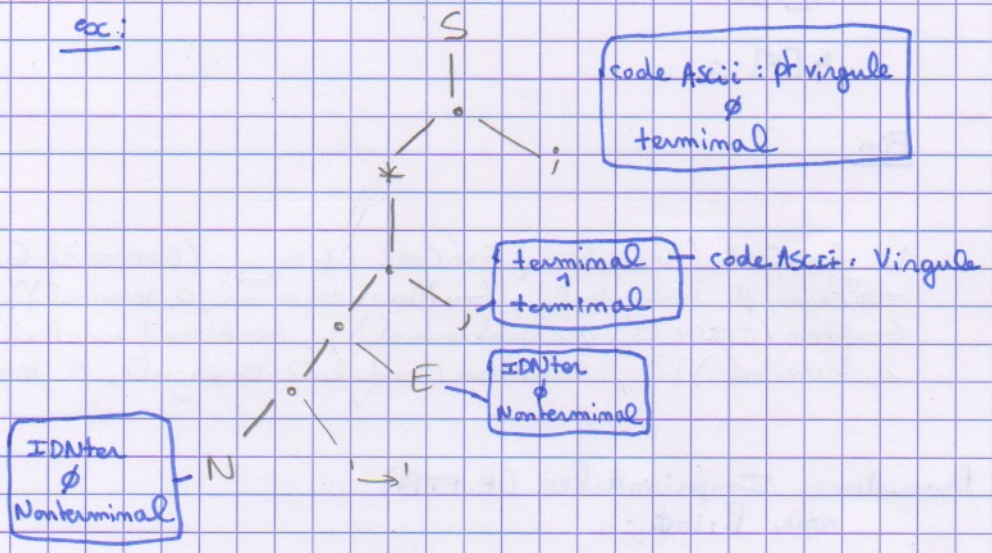
Structures de données:

Type ATomtype = (Terminal, NonTerminal)
Operation = (conc, union, star, un, atom)
↓ conc: •
↓ union: +
↓ star: [ ]*
↓ un: (/ /)
↓ atom: 'IDNter' 'Elter'

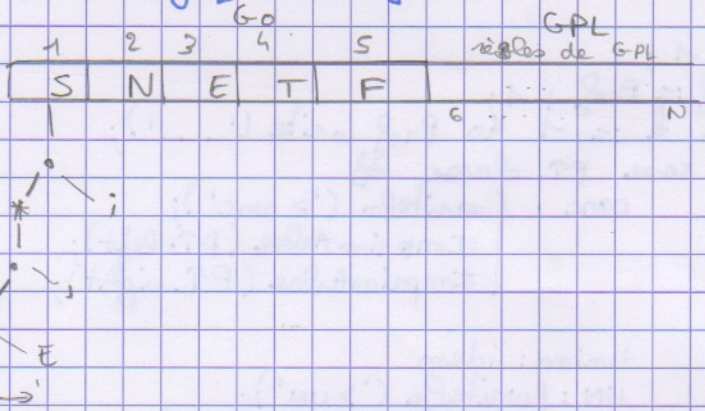
PTR = ↑Node
Node = Enregistrement
Case operation of
conc : (left, right: PTR).
union : (left, right: PTR);

```
Stan : (stance : PTR);
UN : (UNE : PTR);
Atom : (COD, Act : int, Atype : Atomtype);
FIN
```

ex :



A : Array [1, .. 5] of PTR tableau d'arbre



Construire les 5 arbres :

```
Fonction GenConc (P1, P2 : PTR) : PTR
    var P : PTR ;
debut
    New (P, conc) ;
    P↑.left := P1 ;
    P↑.right := P2 ;
    P↑.class := conc ;
  Gen conc := P
fin
```

```
Procedure Foret;
Debut
    A[S] := 1)
    A[N] :=
    A[E] :=
    A[T] :=
    A[F] :=
Fin
```

1) GenConc (GenStar (GenConc (GenConc (GenConc (GenAtom (IDNTerm, ∅, Nonterminal), GenAtom (terminal, 0, terminal)), GenAtom (IDNTerm, ∅, Nonterminal)), GenAtom (codeAscii virgule, 1, terminal))), GenAtom (codeAscii ; virgule, ∅, terminal))

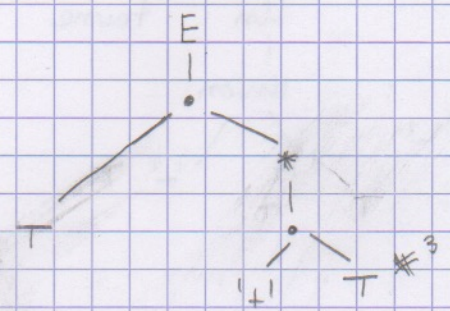
```
Procedure ImprimArbre (P.PTR)
    var k:int;

debut:                                   (Prof := 0)

    k:= 1 ;
    Prof := Prof + 1;
    for k := 1 to Prof write ('___');
        case P↑. classe of
            conc : { writeln ('> conc');
                     ImprimArbre (P↑. left);
                     ImprimArbre (P↑.right);

            Union: idem
            UN : { writeln ('> UN').
                   ImprimArbre (P↑.Une);

            Star : Idem
            Atome : Write('> atome')
                IF P↑. Atype = Terminal then
                        writeln ('cod =' P↑ cod, 'adresse =' P↑. ad,
                          'Terminal' = P↑. Atype);
                Else  ——
                      ——
                      —— (non terminal)

        EndCase
            Prof := Prof - 1

Fin
```

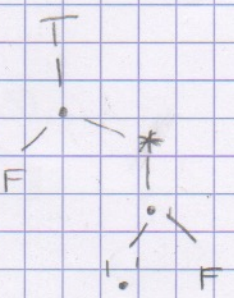
Exemple:

Exp → (/ '+' + '-') . Terme . [(/ '+' + '-') . Terme]) ; i

S	N	E	T	E

A(3) SNE , ImprimArbre
↳ E



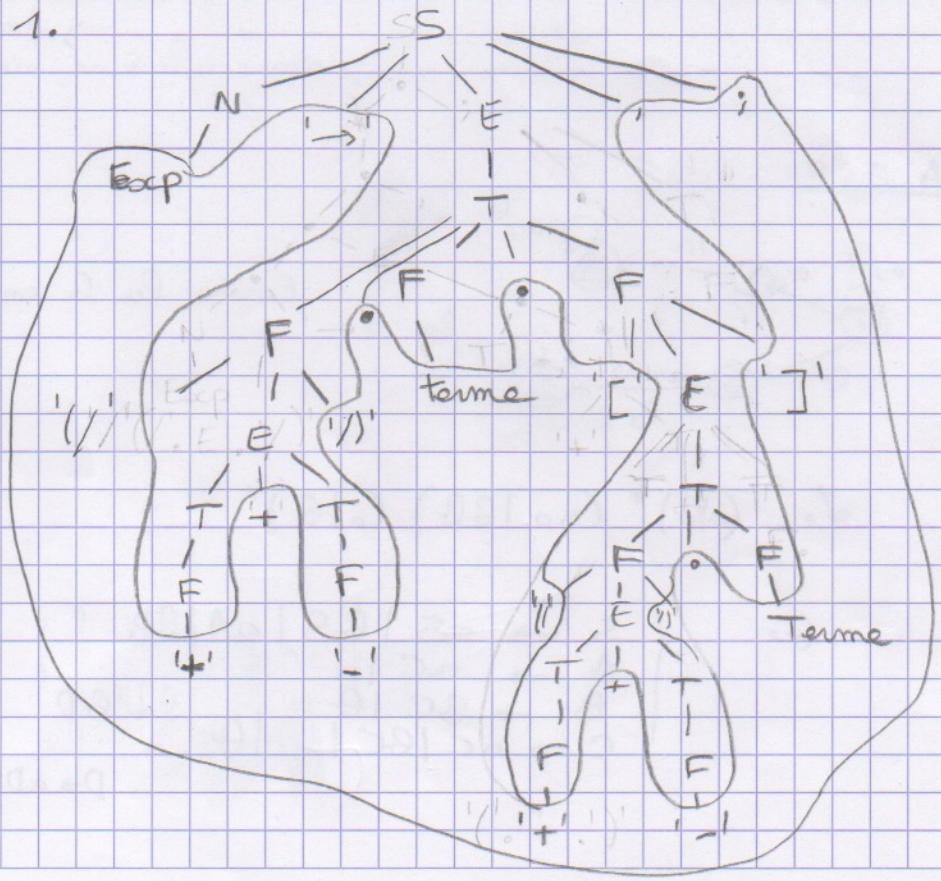


imprimArbre (E)

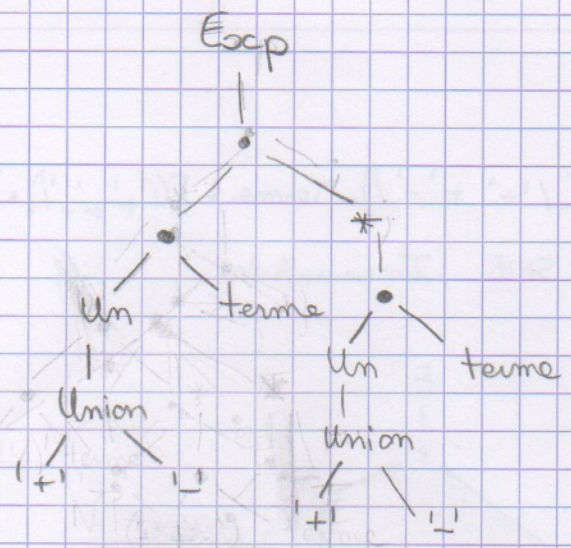
---> conc
---> atome cod = T action = 1 (imprime T) terminal = mon_terminal
---> Star
---> conc
---> atome code = + action = 0 terminal = terminal
---> atome cod = T action = 3 terminal = mon_terminal

1. à l'aide de GO montrons que GPL ∈ GO
2. représenter l'arbre
3. imprimer l'arbre

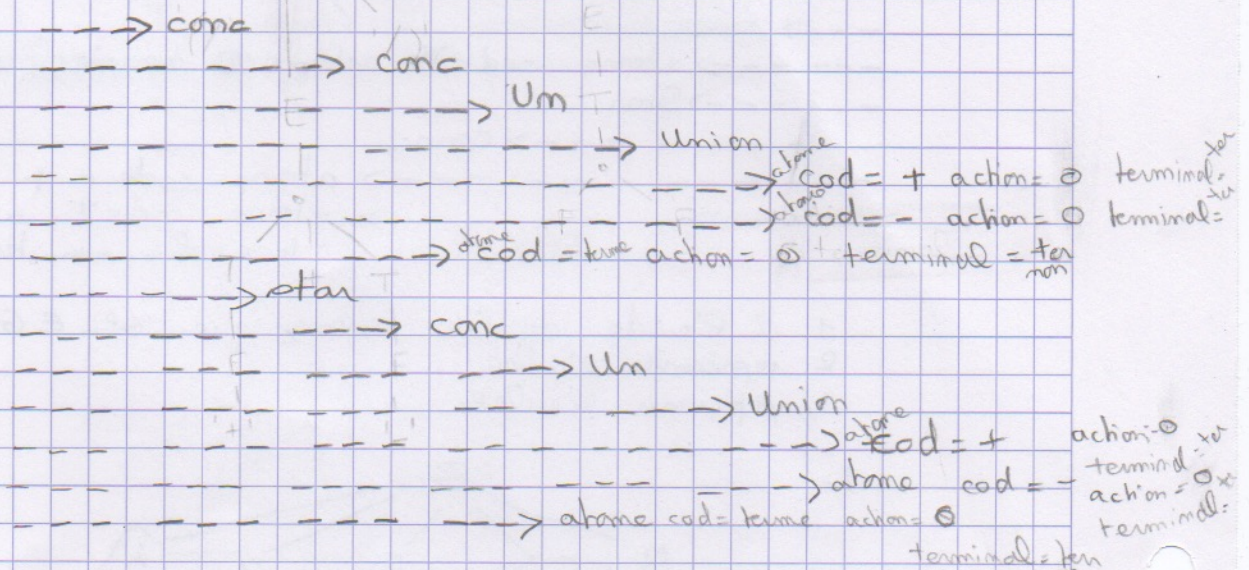
1.



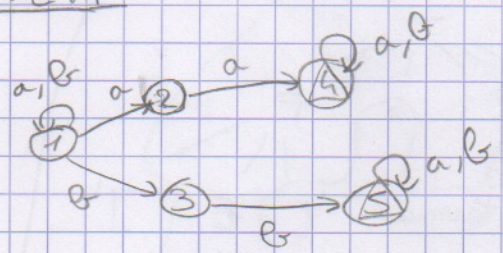
2.



3.



A.E.F



Ecrire la G correspondante

$\mathcal{L} = (a|b)^* (aa|bb) (a|b)^*$

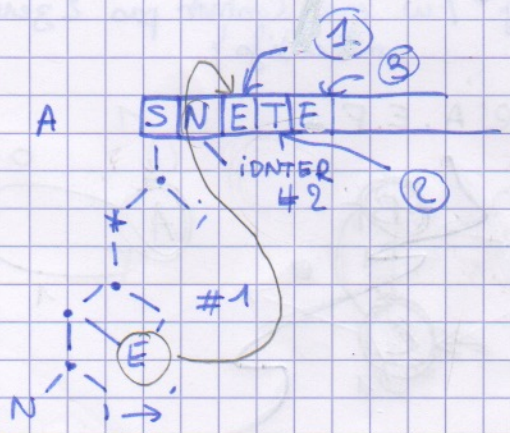
$$G: \begin{cases} S \to aS \mid bS \mid aA \mid bB \\ A \to aC \mid a \\ B \to bC \mid b \\ C \to aC \mid bC \mid a \mid b \end{cases}$$

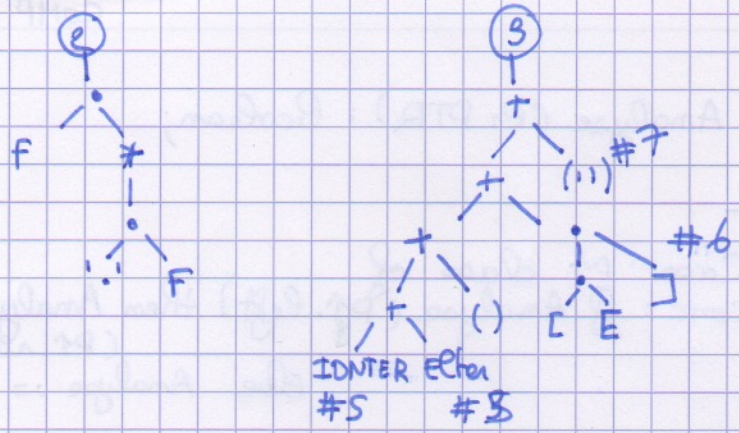
ou bD

pour bien faire on rajoute

$D \to aD \mid bD \mid a \mid b$

```
Fonction Analyse (P: PTR) : Boolean;
Debut:
    Case P↑ class of
    Conc : if Analyse (P↑.left) then Analyse := Analyse
                                              (P↑.right)
           Else Analyse := false;

    Union : if Analyse (P↑.left) then Analyse := true
            Else Analyse := Analyse
                            (P↑.right);

    Star : Analyse := true
           while Analyse (P↑.stare) do;

    UN:  Analyse := true
         if Analyse (P↑.UNE) then

    Atom : Case P↑.Atype of          remplis par Scan

    Terminal : if P↑.cod = code then
               debut
                   Analyse := true
                   if P↑.act ≠ 0 then
                       Go-action (P↑.act)
                   Scan
               Fin
               Else Analyse := false
    Non terminal : if Analyse (A[P↑.cod]) then
                   debut
                       if P↑.act ≠ 0 then
                           Go-action (P↑.act)
                       Analyse := true;
                   Fin
                   Else
                       Analyse := false
FIN
```





Scan recup une unité lexicale et la stock dans une var etc...

GPL : S$ → ['a'] 'b';; a*b

Main {}
Scan
If Analyse (A[S]) then
write ('OK')

G_0 : { S → aSb
S → aSb

G_{pl} : { aa cb
↑ Scan

Simulation de la GPL : S$ → ['a'] 'b';;

Scan : S$

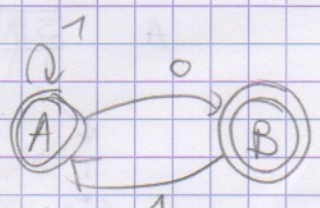
Analyse (S$) ⟹ IDNTER #2

G_0-action (2)
true

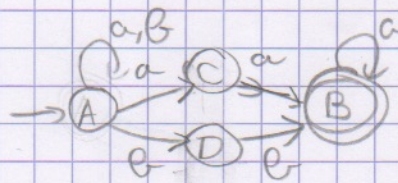
G_0-action 2 - 5 - 6 - 5 - 1

- A = {ω ∈ {0,1}* / ω ne contient pas 2 zéros consécutifs}

construire l'A.E.F

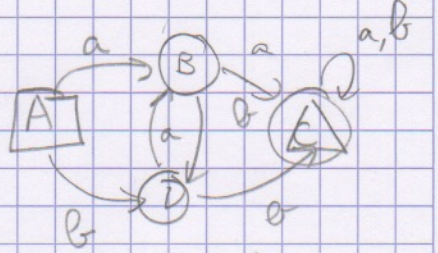


$L = \{ w \in \{a,b\}^* / w$ contient $2a$ ou $2b$ consécutifs $\}$



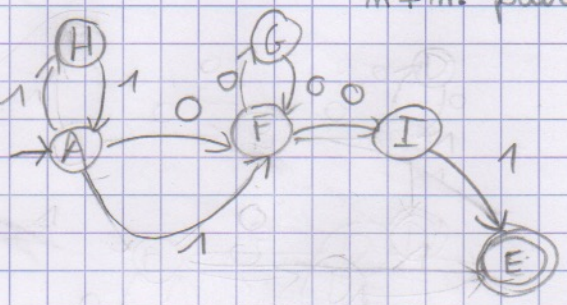
expreg

ou



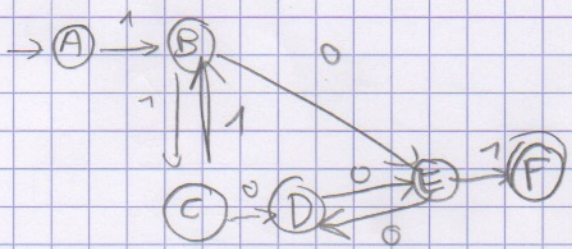
A

$L = \{ w \in \{0,1\}^* / w = 1^m 0^m 1 \}$
$m, m > 0$
$m+m$ pair



mon can 001

$$\begin{cases} A \to 0\,1H \mid 1F \mid 0F \\ H \to 1A \\ F \to 0G \mid 0I \\ G \to 0F \\ I \to 1 \end{cases}$$



$A \to 1B$
$B \to 1C \mid 0E$
$C \to 1B \mid 0D$
$D \to 0E$
$E \to 0D \mid 1$

Montrez que tout langage fini est régulier

$w = x_1 \dots x_n \to \begin{cases} S \to x_1 S_1 \\ S_1 \to x_2 S_2 \\ \\ S_n \to x_n \end{cases}$ grammaire reg

- projet: analyseur & Arbre de dep fini

Construction de la Go action:

De quoi a-t-on besoin ?
- Un tableau pile Pile [i]
- 2 dictionnaires :
  DICOT
  DICONT

```
Procedure Go Action (act : int);
    var T1, T2 : PTR;
    debut
        case action of
```

1 : debut
depiler (T1);
depiler (T2);
A[T2↑.code + 5] := T1;
fin

1: écrire chacune des règles

type caractère ou (ter ou nter)

2 : Empiler (Genatom (Recherche (DICONT), Action, CA-type)))
↳ sur le So.

genatom (code (So), Ø, Nonterm)

3 : debut
Depiler (T1);
Depiler (T2);
Empiler (GenUnion (T2, T1));
fin

génère genunion construit la fusion des 2 arbres

4 : debut
Depiler (T1);
Depiler (T2);
Empiler (GenConc (T2, T1));
fin

idem 3 mais conc

5 : if CA-type = Terminal then
Empiler (Genatom (Recherche (DICOT), act, terminal))
Else
Empiler (Genatom (Recherche (DICONT), act, non-terminal))

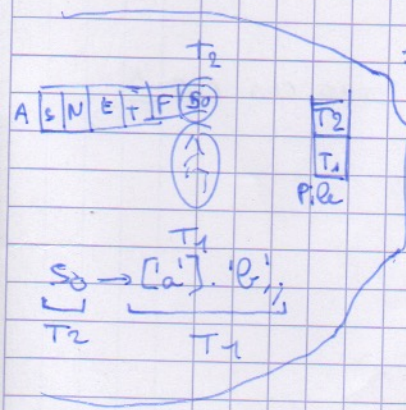
cherche code ascii et empiler après genatom

6 : debut
Depiler (T1); Empiler (GenStan (T1));
fin

genStan

7 : debut
Depiler (T1); Empiler (GenUn (T1));
fin

genUn

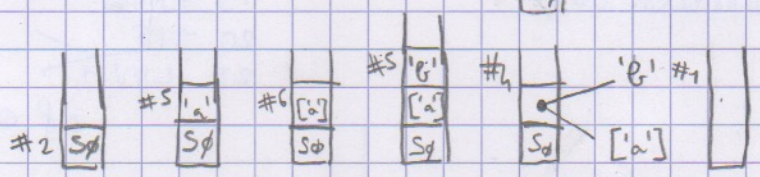


- Go action génère automatiquement Gencons, GenStar etc...
- Etant donné la petite grammaire GPL ... - montrer
  ↳ Go → fichier arbre Go.
  - remplissage DICOT, DICONT et donner état pile

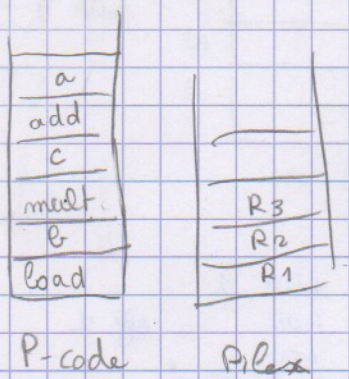
$S_0 \rightarrow ['a'] \cdot 'b';$

DICOT = {a b} ← 'b' 'a'

DICONT = {S_ϕ} ↖ S_ϕ



fonction Analyse pour construire la pile



registres

y := a + b * c

load b
mult c
add a
to y

Génération de P-code

3 inst. de chargements { LDA @ (charge @), LDV V, LDC C (charge la constante de l'adresse)

Opérateurs Arith { ADD + (plus), SUB - (moins), DIV /, MULT *, NEG - (unaire), INC (+1), DEC (-1)

inst. de Saut { TMP @ (prend contrainte telle quelle), JIF @ ← jump if false, JSR ← jump to subroutine (procédure), RSR

opérations relationnels { SUP >, SUPE ≥, INF <, INFE ≤, EG =, DIFF ≠

opérat. logiques { AND, OR, NOT

Inst pour I/O { RD, RDLN, WRT, WRTLN

AFF : affectation
STOP : inst. d'arrêt

LN = retour chariot

• Program Somme;
var I, S, N : int;

Debut

Read (N);
S := 0;
I := 1;

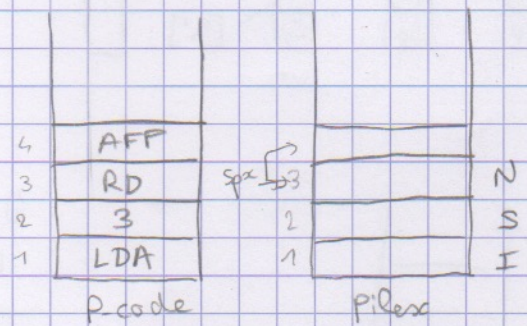
while I ≤ N do
debut
S := S + I;
I := I + 1;
fin
writeln (S);

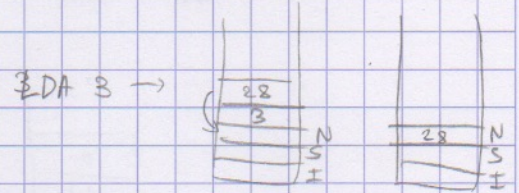
fin

n° dans la pile P.code

1	LDA 3	24	LDV 2
3	RD	26	ADD
4	AFF	27	LDC 1
5	LDA 2	29	INC
7	LDC 0	30	JMP 15
9	AFF	32	LDA 1
10	LDA 1	34	WRT
12	LDC 1	35	STOP
14	AFF		
15	LDA 1		
17	LDA 3		
19	INFE		
20	JIF		
22	LDV 1		

cf correction





P.code:

1.	LDA 3	26	LDV 1
3	RD	28	ADD
4	AFF	29	AFF
5	LDA 2	30	LDA 1
7	LDC 0	32	LDV 1
9	AFF	34	LDC 1
10	LDA 1	36	ADD
12	LDC 1	37	AFF
14	AFF	38	JMP 15
15	LDV 1	40	LDV 2
17	LDV 3	42	WRTLN
19	INFE	43	STOP
20	JIF 40		
22	LDA 2		
24	LDV 2		

aff 1

aff 2
ou avec INC

test

Procédure Exec;
debut
while P-code [C_0] ≠ Stop do
Interprete (Pcode (C_0))
fin

C_0 : compteur ordinal
(pointeur de P-code
= instruction courante)

```
Procedure Interprete (x : int);
    case x of

LDA : Spx := Spx + 1;
      Pile_ex (Spx) = P_code (Co + 1);   interprète Co + 1
      Co := Co + 2;                      puis + 2

LDV : Spx := Spx + 1;
      Pile_ex(Spx) = Pile_ex (P_code (Co + 1));
      Co := Co + 2;
```