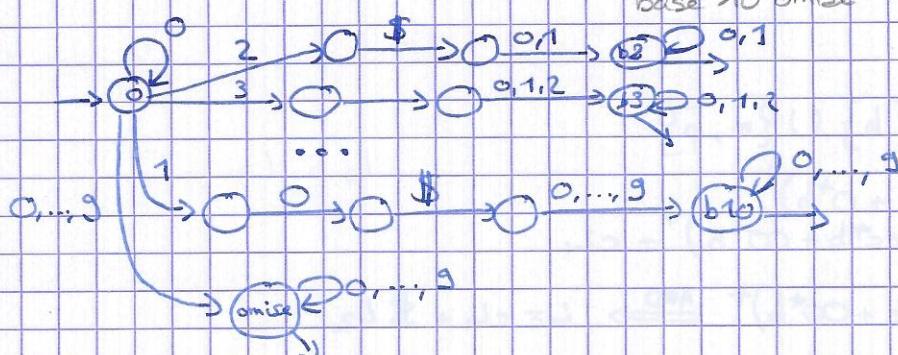


## A - Analyse lexicale

$$r^+ = r^- + \lambda \quad r^+ = r^-$$

$$1) a \cdot L = 0^* 2 \$ (0+1)^+ + 0^* 3 \$ (0+1+2)^+ + \dots + 0^* 9 \$ (0+1+\dots+8)^+$$

$$+ 0^* 10 \$ (0+\dots+8+9)^+ + 0^* (0+1+\dots+9)^+$$



non déterministe  
mais on sait déterminiser  
et on aura un auto fini  
déterministe équivalent

base \$ valeur  
suite finie                    idem  
non vide de  
chiffres

Automate fini déterministe qui reconnaît  $L' \cap L \subseteq L'$  ( $L'$  est un surlangage)

$$X = \{0, 1, \dots, 8, 9, 10\}$$

$$C = \{0, \dots, 9\}$$

$$\begin{aligned} L' &= c^+ \otimes c^+ + c^+ \\ L' &= cc^* \otimes cc^* + cc^* \end{aligned} \quad \text{avec } c \in C$$

$$L' = cc^* + cc^* + cc^*$$

$$b - L' = cc^* \$ cc^* + cc^*$$

$$= cL.$$

$$L_i = c^* \$cc^* + cc^*$$

$$= c^* (\$cc^* + cc^*) \quad \text{ARDEN} \Rightarrow L_1 = cl_1 + \$cc^* + \lambda = cl_1 + \$L_2 + \lambda$$

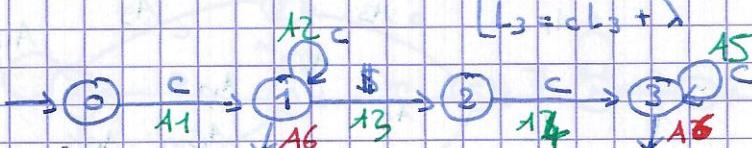
$$L_2 = cC^* = cL_3$$

$$l_3 = c^* = c^* \cdot \lambda$$

5 6

## Syst. d'équations de langage :

$$\begin{cases} L' = cL_1 \\ -1 = cL_1 + \$L_2 + \lambda \\ -2 = cL_3 \\ -3 = cL_3 + \lambda \end{cases}$$



Auto fini déterministe qui reconnaît  $L'$ , sur langage de  $L$

- Ajouter des actions sémantiques sur les transitions

- Actions sémantiques écrites dans un pseudo-langage :

On a: caracour = code ASCII du symbole

`val = une fonction tq val(cancer) = valeur décimale du caractère courant`

attribut(s) sémantique(s): la valeur qui contiendra la valeur décimale du nombre (u)

*Callouts* *multiple*

## symbolic w

constante MAX qui contient le nombre maximum qui doit être reconnu

variable base qui contient la valeur de la base reconnue

A1: { base := val(carcour); } Ast init de la base x/

A2. { base := base \* 10 + val (caractere) } /\* Schéma de Horner pour calculer la valeur décimale de la

A3  $\{ \text{if } (\text{base} < 2) \text{ or } (\text{base} > 10) \text{ then ERREUR, (1)} \}$  \* base trop grande ou trop petite \*/

A4: { valeur := val(carcour) /\* initialiser la valeur \*/  
 if valeur  $\geq$  base then ERREUR(2) } /\* chiffre  $\notin$  base \*/

A5: { if val(carcour)  $\geq$  base then ERREUR(2)  
 else valeur := valeur + base + val(carcour); }

A6: { if valeur > VMAX then ERREUR(3)  
 else return valeur; }

$$2) X = \{0, \dots, 9\} \cup \{\$\} \cup \{b\} \cup \{\bar{m}, \bar{p}\}$$

$$\begin{aligned} L_0 &= c^+ \$ (m^+ c^+ b + p^+ c^+ b + 0^+ b)^+ \\ &= cc^* \$ (mm^* cc^* b + pp^* cc^* b + oo^* b)^+ = cL_1 \end{aligned}$$

$$L_1 = c^* \$ (mm^* cc^* b + pp^* cc^* b + oo^* b)^+ \xrightarrow{\text{ARD}} L_1 = cL_1 + \$ L_2$$

$$\begin{aligned} L_2 &= (mm^* cc^* b + pp^* cc^* b + oo^* b)^+ \\ &= (mm^* cc^* b + pp^* cc^* b + oo^* b)(mm^* cc^* b + pp^* cc^* b + oo^* b)^* \\ &= mm^* cc^* b ( )^* + pp^* cc^* b ( )^* + oo^* b ( )^* \\ &= mL_4 + pL_3 + oL_5 \end{aligned}$$

$$L_3 = p^* cc^* b ( )^* \xrightarrow{\text{ARD}} L_3 = pL_3 + cc^* b ( )^* = pL_3 + cL_6$$

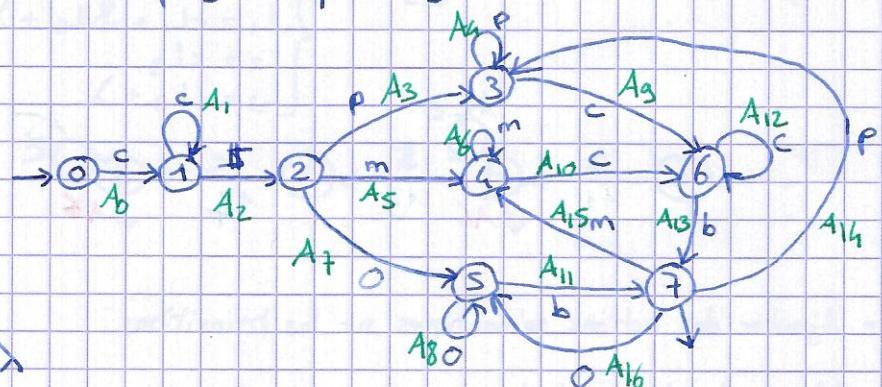
$$L_4 = m^* cc^* b ( )^* \xrightarrow{\text{ARD}} L_4 = mL_4 + cc^* b ( )^* = mL_4 + cL_6$$

$$L_5 = o^* b ( )^* \xrightarrow{\text{ARD}} L_5 = oL_5 + b ( )^* = oL_5 + bL_7$$

$$L_6 = c^* b ( )^* \xrightarrow{\text{ARD}} L_6 = cL_6 + b ( )^* = cL_6 + bL_7$$

$$L_7 = ( )^* = ( )^+ + \lambda = L_2 + \lambda = pL_3 + mL_4 + oL_5 + \lambda$$

$$\left\{ \begin{array}{l} L_0 = cL_1 \\ L_1 = cL_1 + \$L_2 \\ L_2 = pL_3 + mL_4 + oL_5 \\ L_3 = pL_3 + cL_6 \\ L_4 = mL_4 + cL_6 \\ L_5 = oL_5 + bL_7 \\ L_6 = cL_6 + bL_7 \\ L_7 = pL_3 + mL_4 + oL_5 + \lambda \end{array} \right.$$



A0 { base := val(carcour); }

A1 { base := base + 10 + val(carcour); }

A2 Ø

A3 { cpt := 1; signe := 1; } A4

A5 { cpt := 1; signe := -1; } A15

A7 { cpt := 1; signe := 0; } A16

A4 { cpt := cpt + 1; } A6 A8

A9 { ecart := val(carcour); } A10

A12 { ecart := ecart + 10 + val(carcour); }

A13 { r := base + signe \* ecart; }

while not (r  $>$  1000) loop { print r; wait 1; r := r + 1; } A11

TL-  
TD1

$$\begin{aligned} ③ \quad I &\rightarrow c \mid cI \\ J &\rightarrow II \mid SI \\ K &\rightarrow .I \\ L &\rightarrow II \mid K \mid IK \\ \Pi &\rightarrow eJ \\ N &\rightarrow L \mid M \mid LM \\ O &\rightarrow N \mid sN \end{aligned}$$

$$\begin{aligned} c &\in [0, \dots, 3] \\ s &\in [+,-] \end{aligned}$$

$$X = \{0, \dots, 3, +, -, \circ, e\}$$

Axiome: O

Syst. d'équations de langage équivalent à la grammaire d'axiome O

$$\begin{cases} I = c + cI \\ J = I + SI \\ K = .I \\ L = I + K + IK \\ \Pi = eJ \\ N = L + \Pi + LM \\ O = N + sN \end{cases}$$

Linéaire à droite:  $I_0 \rightarrow aI_0 \mid bI_1 \mid \lambda$   
 Linéaire à gauche:  $S \rightarrow Sa \mid \lambda$

Auto fini  $\Leftrightarrow$  grammaire linéaire à droite  
 $L(O)$  est-il bien un langage régulier?

{ L'union de langages réguliers est un langage régulier  
 Le produit }

$$L(I) = c^+ \text{ régulier}$$

$$L(J) = c^+ + sc^+ \text{ régulier}$$

$$L(K) = .c^+ \text{ régulier}$$

$$L(L) \Rightarrow \text{union de langages réguliers régulier}$$

$$L(\Pi) = e(c^+ + sc^+) \text{ régulier}$$

$$L(N) \Rightarrow \text{union de langages réguliers régulier}$$

$\Rightarrow L(O)$  est régulier, il est possible de construire un auto fini:

$$I = c + cI = c(\lambda + I) = cP$$

$$P = I + \lambda = cP + \lambda$$

$$J = I + SI = cP + SI$$

$$K = .I$$

$$\begin{aligned} L = I + K + IK &= cP + .I + cPK \\ &= c(P + PK) + .I \\ &= cQ + .I \end{aligned}$$

$$\begin{aligned} Q = P + PK &= cP + \lambda + (cP + \lambda)K = cP + \lambda + cPK + \lambda \\ &= c(P + PK) + \lambda + .I \\ &= cQ + \lambda + .I \end{aligned}$$

$$\Pi = eJ$$

$$\begin{aligned} N = L + \Pi + LM &= cQ + .I + eJ + (cQ + .I)\Pi \\ &= cQ + .I + eJ + cQM + .IM\Pi \\ &= c(Q + Q\Pi) + .(I + IM) + eJ \\ &= cR + .S + eJ \end{aligned}$$

$$\begin{aligned} R = Q + Q\Pi &= cQ + .I + \lambda + (cQ + .I + \lambda)\Pi \\ &= cQ + .I + \lambda + cQ\Pi + .IM + \Pi \\ &= c(Q + Q\Pi) + .(I + IM) + eJ + \lambda \\ &= cR + .S + eJ + \lambda \end{aligned}$$

$$\begin{aligned} S = I + IM &= cP + cPM \\ &= c(P + PM) \\ &= cT \end{aligned}$$

$$\begin{aligned} T = P + PM &= cP + \lambda + (cP + \lambda)M \\ &= cP + \lambda + cPM + M \\ &= c(P + PM) + eJ + \lambda \\ &= cT + eJ + \lambda \end{aligned}$$

$$O = N + sN = cR + .S + eJ + sN$$

$$I = cP$$

$$J = cP + SI$$

$$K = .I$$

$$L = cQ + .I$$

$$\Pi = eJ$$

$$N = cR + .S + eJ$$

$$O = cR + .S + eJ + sN$$

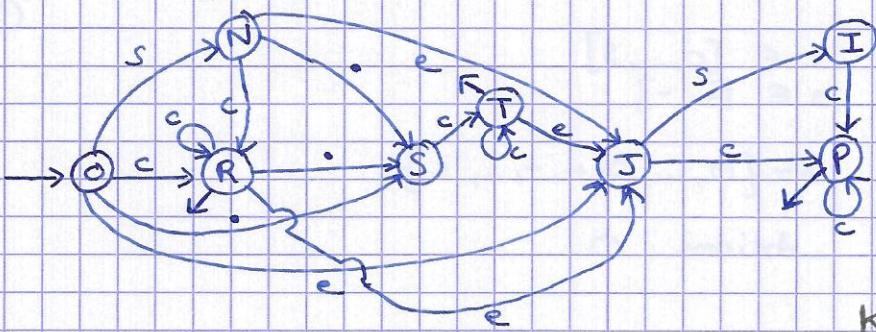
$$P = cP + \lambda$$

$$Q = cQ + .I + \lambda$$

$$R = cR + .S + eJ + \lambda$$

$$S = cT$$

$$T = cT + eJ + \lambda$$



K L Q  $\cap$  sont des états non atteignables depuis O  $\Rightarrow$  inutile

$$R \sim 3S + 3S - 3S$$

$$T \sim 3S \cdot 6 + 3S \cdot 6 - 3S \cdot 6 \cdot 67 \doteq 67$$

$$P \sim 3Se + 3 - 3S \cdot 6e^{-2}$$

### B - Systèmes d'équations algébriques de langages

$$G_1: S = Sa + b \xrightarrow{\text{ARD bis}} S = ba^+$$

$$G_2: S \xrightarrow{\quad} aSb \quad ) \quad S = aSb + T \quad S \text{ est hors contexte} \neq \text{régulier} \quad S = a^n T b^n \ n \geq 0$$

$$S \xrightarrow{\quad} T$$

$$T \xrightarrow{\quad} Tb \quad ) \quad T = Tb + b \xrightarrow{\text{ARD bis}} T = bb^+ = b^+$$

$$T \xrightarrow{\quad} b$$

$$\Rightarrow S = a^n b^+ b^n \ n \geq 0$$

$$G_2: S = a^m b^n \ m > n \geq 0$$

$$G_3: S \xrightarrow{\quad} SS \mid a \quad S = SS + a$$

Vérifier si  $a^+$  est solution de  $S = SS + a$  c'est à dire  $a^+ = a^+ a^+ + a$  ?

$$a^+ = \sum_{i \geq 1} a^i ; a^+ a^+ = \sum_{i \geq 1} a^i \sum_{j \geq 1} a^j = \sum_{i \geq 1, j \geq 1} a^{i+j} = \sum_{k \geq 2} a^k$$

$$\text{donc } a^+ a^+ + a = \sum_{k \geq 2} a^k + a^1 = \sum_{p \geq 1} a^p = a^+$$

Donc  $a^+$  est solution de l'équation  $S = SS + a$

Donc  $L(G_3) = a^+$

$a^+$  est solution. Y en a-t-il d'autres ?

$a^+$  solution ?

Vérifions si  $a^+ = a^+ a^* + a$  :  $a^+ a^* = a^*$  or  $a \in a^+$  donc  $a^+ a^* + a = a^+$

$\forall \Sigma \subset X^* \text{ t q } a \in \Sigma \text{ alors } \Sigma^* \text{ est solution de } S = SS + a \text{ (algébriquement)}$

$a^+$  est la solution "naturelle" de  $S = SS + a \iff$  c'est le plus petit point fixe de l'équation, il peut être trouvé en résolvant le système d'éq. de lang. par approximations successives. On part de l'ensemble vide, on applique  $E(\varphi) = E^{-1}(\varphi)$ . On itère :  $E^i(\varphi) = E(E^{i-1}(\varphi))$ . Le plus petit point fixe est  $\lim_{i \rightarrow \infty} E^i(\varphi)$

$E(\varphi) = \dots$

$$1) \begin{cases} S' \rightarrow S \$^k \\ S \rightarrow bRS \mid Rcsa \mid \lambda \\ R \rightarrow acR1b \end{cases}$$

•  $k=0$  G est-elle LL(0) ? Non, car 2 règles pour R et 3 règles pour S

•  $k=1$  G LL(1) ? Pour R, on sait choisir la règle selon qu'on voit a ou b

$$\begin{aligned} \text{first}_1(R) &= \{a, b\} \\ \text{first}_1(S) &= \{\lambda, a, b\} \\ \text{follow}_1(S') &= \{\lambda\} \end{aligned}$$

$$\begin{aligned} \text{follow}_1(S) &= \{a, \$\} \\ \text{règle } 0: &= \text{first}_1(\$). \text{follow}_1(S') = \{\$^k\} \\ \text{règle } 1: &= \text{first}_1(\lambda). \text{follow}_1(S) = \{a\} \\ &\Rightarrow \text{ne sert à rien} \end{aligned}$$

$$\begin{aligned} \text{follow}_1(R) &= \text{règle } 1: \text{first}_1(S). \text{follow}_1(S) = \text{first}_1(S) \cdot \text{first}_1(\text{follow}_1(S)) \\ &= \{a, b, c, \$\} \\ &= \{\$, a, b\} \end{aligned}$$

$$\text{règle } 2: = \text{first}_1(cSa). \text{follow}_1(S)$$

a calculer pour  $\text{first}_1(S) = \lambda$

$$\begin{aligned} &= \{c\} \\ \text{règle } 4: &\text{ ne sert à rien} \end{aligned}$$

$$1\text{-lookahead } (S \rightarrow bRS) = \text{first}_1(bRS \cdot \text{follow}_1(S)) = \{b\}$$

$$1\text{-lookahead } (S \rightarrow Rcsa) = \text{first}_1(Rcsa \cdot \text{follow}_1(S)) = \text{first}_1(R) \cdot c \cdot \text{first}_1(S) \dots = \{a, b\}$$

$$\{b\} \cap \{a, b\} = \{b\} \neq \emptyset \text{ non LL(1)}$$

•  $k=2$  G LL(2) ?

$$\begin{aligned} \text{first}_2(R) &= \{b, ac\} \\ \text{first}_2(S) &= \{ba, bb, ac, bc, \lambda\} \\ \text{follow}_2(S') &= \{\lambda\} \end{aligned}$$

$$\begin{aligned} \text{follow}_2(S): &r_0 \rightarrow \{\$\$\} \\ &= \{\$, aa, ab\} \\ &r_2 \rightarrow \text{first}_2(a \cdot \text{follow}_2(S)) \\ &= a \cdot \text{first}_2(\text{follow}_2(S)) = \{aa, ab\} \\ &r_1 \rightarrow \text{ne sert pas} \end{aligned}$$

$$\text{follow}_2(R): r_1: \text{first}_2(S \cdot \text{follow}_2(S))$$

$$\begin{aligned} &= \text{first}_2(\text{first}_2(S) \cdot \text{first}_2(\text{follow}_2(S))) \\ &= \{ba, bb, ac, bc, \$\$, ab, aa\} \end{aligned}$$

r<sub>4</sub>: ne sert à rien

$$r_2: \text{first}_2(cSa \cdot \text{follow}_2(S))$$

$$\begin{aligned} &= c \cdot \text{first}_2(S \cdot \text{follow}_2(S)) \\ &= \{ca, cb\} \end{aligned}$$

$$\begin{aligned} &= \{\$, aa, ab, ac, ba, bb, \\ &\quad bc, ca, cb\} \end{aligned}$$

Pas de problème pour S', ni pour R dès que  $k \geq 1$

$$\text{Pour } S, k=2: 2\text{-lookahead } (S \rightarrow bRS) = \text{first}_2(bRS \cdot \text{follow}_2(S))$$

$$= b \cdot \text{first}_1(RS \cdot \text{follow}_2(S)) = \{ba, bb\} = E_1$$

$$\{ac, bc\} \cap \{ba, bb\} = \emptyset$$

$$2\text{-lookahead } (S \rightarrow Rcsa) = \text{first}_2(Rcsa \cdot \text{follow}_2(S)) = \{ac, bc\} = E_2$$

$$2\text{-lookahead } (S \rightarrow \lambda) = \text{first}_2(\lambda \cdot \text{follow}_2(S)) = \{\$, ab, aa\} = E_3$$

$$E_1 \cap E_2 = \emptyset$$

$$E_1 \cap E_3 = \emptyset$$

$$E_2 \cap E_3 = \emptyset$$

$$\Rightarrow G \text{ est LL(2)}$$

Pour construire la table d'analyse

- Pour  $S'$ : 2-lookahead ( $S' \rightarrow S \cdot \$^2$ ) = {ba, bb, ac, bc,  $\$^2$ }

- Pour  $R$ : 2-lookahead ( $R \rightarrow \text{acc}R$ ) = {ac}

2-lookahead ( $R \rightarrow b$ ) =  $\text{first}_2(b \cdot \text{follow}_1(R)) = \{\text{bb}, \text{ba}, \text{bc}, b\$^2\}$

### Table d'analyse

		ba	bb	ac	bc	$\$^2$	b\$	a\$	aa
		s'	S\$	S\$	S\$	S\$	S\$	S\$	S\$
		o	o	o	o	o	o	o	o
S	bRS	bRS	RcSa	RcSa	$\lambda$	$\lambda$	$\lambda$	$\lambda$	
	1	1	2	2	3		3	3	
R	b	b	b	b	b				
	s	s	s	s	s				
a	pop			pop	pop				
b	pop	pop	pop	pop					
c									
\$					accept				

$(bc b c a a \$ \$, S') \xrightarrow{\cdot} (bc b c a a \$ \$, S \$) \xrightarrow{?} (bc b c a a \$ \$, R c S a \$) \xrightarrow{?} (bc b c a a \$ \$, b c S a \$)$

$\xrightarrow{\text{pop}} (cb c a a \$ \$, c S a \$) \xrightarrow{\text{pop}} (b c a a \$ \$, S a \$) \xrightarrow{?} (b c a a \$ \$, R c S a a \$) \xrightarrow{?} (b c a a \$ \$, b c S a a \$)$

$\xrightarrow{\text{pop}} (c a a \$ \$, c S a a \$) \xrightarrow{\text{pop}} (a a \$ \$, S a a \$) \xrightarrow{?} (a a \$ \$, a a \$) \xrightarrow{\text{pop}} (a \$ \$, a \$) \xrightarrow{\text{pop}} (\$, \$)$

Le mot bc b c a a est bien engendré par la grammaire et reconnu par l'automate à pile

$$2) S' \rightarrow O \$^L$$

$$O \rightarrow cR \mid .S \mid eJ \mid sN$$

$$R \rightarrow cR \mid S \mid eJ \mid \lambda$$

$$S \rightarrow cT$$

$$T \rightarrow cP \mid sI$$

$$N \rightarrow cR \mid .S \mid eJ$$

$$T \rightarrow cT \mid eJ \mid \lambda$$

$$P \rightarrow cP \mid \lambda$$

$$I \rightarrow cP$$

$$c \in \{0, \dots, 9\}$$

$$s \in \{+, -, \cdot\}$$

$$x = \{+, -, e, ., 0, \dots, 9\}$$

Grammaire LL(1)

• Vérifier pour  $R$ : 1-lookahead ( $R \rightarrow \lambda$ ) =  $\text{first}_1(\lambda \cdot \text{follow}_1(R)) = \{\$\$^2\}$

$$\text{follow}_1(R) = \text{follow}_1(N) + \text{follow}_1(O)$$

$$= \text{follow}_1(O)$$

$$= \{\$\$^2\}$$

Grammaire LL(1)

$$\text{follow}_1(T) = \{\$\$^2\}$$

...

TL  
TD2

procedure S' is  
begin

SCAN;  
O;  
SKIP('\$\$');

end S'

procedure R is

begin

switch NEXTS  
'number': SKIP(...), R;  
. : SKIP('.'); S;  
e: SKIP('e'); S;  
\$: NULL;  
others: ERREUR();

end  
end R

procedure J is

begin

switch NEXTS  
'number': SKIP(number.valeur), P;  
+ : SKIP('+'), I;  
- : SKIP('-'), I;  
others: ERREUR()

end  
end J

procedure T is

begin

Switch: NEXTS  
'number': —; T;  
e: SKIP('e'), J;  
\$: NULL;  
others: ERREUR()

end  
end T

procedure O is  
begin

switch NEXTS

'number': SKIP(number.valeur), R; ← - : SKIP(''); N;  
. : SKIP('.'); S;  
e: SKIP('e'); S;  
+ : SKIP('+'), N;  
others: ERREUR()

end

end O

Quand le scanner a trouvé un chiffre, l'US est 'number' et il a construit la valeur décimale correspondante dans l'attribut "valeur"

procedure S is

begin

SKIP (number.valeur);

T;

end S

procedure N is

begin

switch NEXTS  
'number': —; R;  
. : SKIP('.'); S;  
e: SKIP('e'); S;  
others: ERREUR()

end  
end N

procedure I is

begin

Skip(number.v);  
P;

end I

procedure P is

begin

switch NEXTS  
'number': —; P  
\$: NULL;  
others: ERREUR()

end  
end P;

3) 1-lookahead ( $S \rightarrow aSbS$ ) = first<sub>1</sub>(aSbS. follow<sub>1</sub>(S)) = {a} = E<sub>1</sub>

1-lookahead ( $S \rightarrow \lambda$ ) = first<sub>1</sub>(\lambda. follow<sub>1</sub>(S)) = {b, \$} = E<sub>2</sub>

$E_1 \cap E_2 = \emptyset \Rightarrow$  grammaire LL(1)

procedure S' is

begin

SCAN;  
S;  
SKIP('\$\$');

end S'

procedure S is

begin

switch NEXTS

a: SKIP('a'), S; SKIP('b'), S;  
b|\$|\$: NULL; //image de  $S \rightarrow \lambda$   
others: ERREUR()

end

end S

4) La grammaire est-elle LL(1) ?

⇒ récursive à gauche ( $\langle \text{suiteInst} \rangle \rightarrow \langle \text{suiteInst} \rangle, \langle \text{inst} \rangle$ ), pas LL(1)

⇒ il faut éliminer la récursivité à gauche  $\Leftrightarrow$  construire une grammaire équivalente qui ne soit pas récursive à gauche

Éliminer la récursivité à gauche

$$A \rightarrow A\alpha \mid \beta \Leftrightarrow A = A\alpha + \beta \xrightarrow{\text{ARDEN bis}} A = \underbrace{\beta \alpha^*}_{A'} = \beta A' \text{ avec } A' = \alpha^*$$

$$A' = \alpha^* \cdot \lambda \xrightarrow{\text{ARDEN}} A' = \alpha A' + \lambda$$

On part de  $\begin{cases} A \rightarrow A\alpha \\ A \rightarrow \beta \end{cases}$  et on a  $\begin{cases} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' \\ A' \rightarrow \lambda \end{cases}$

...

$$\begin{cases} \langle \text{suiteInst} \rangle \xrightarrow{1} \langle \text{inst} \rangle \langle \text{suiteInst}' \rangle \\ \langle \text{suiteInst}' \rangle \xrightarrow{2} ; \langle \text{inst} \rangle \langle \text{suiteInst}' \rangle \\ \langle \text{suiteInst}' \rangle \xrightarrow{3} \lambda \end{cases}$$

...

• Vérifier que  $G'$  est bien LL(1)

1-lookahead ( $\langle \text{suiteInst}' \rangle \rightarrow ; \langle \text{inst} \rangle \langle \text{suiteInst}' \rangle$ ) = {; } =  $E_0$

1-lookahead ( $\langle \text{suiteInst}' \rangle \rightarrow \lambda$ ) = first<sub>1</sub>( $\lambda$ . follow<sub>1</sub>( $\langle \text{suiteInst}' \rangle$ )) =  $E_1$

$$E_0 \cap E_1 = \emptyset$$

$$\begin{aligned} \text{follow}_1(\langle \text{suiteInst}' \rangle) &= \text{follow}_1(\langle \text{suiteInst} \rangle) && (\text{d'après règle 1}) \\ &= \{\text{end, else, endif, endloop, until}\} &= E_1 \end{aligned}$$

1-lookahead (règle 4) = {if }

1-lookahead (règle 5) = {while }

1-lookahead (règle 6) = {repeat }

$$\left. \begin{array}{l} \text{1-lookahead (règle 4)} = \{\text{if}\} \\ \text{1-lookahead (règle 5)} = \{\text{while}\} \\ \text{1-lookahead (règle 6)} = \{\text{repeat}\} \end{array} \right\} \cap = \emptyset$$

$G'$  est LL(1)

procedure PROGRAM is

begin

SKIP('program');

SUITE DCL;

SKIP('begin');

SUITEINST;

SKIP('end');

end PROGRAM;

procedure INST is

begin

switch NEXTS

; : SKIP(';'); EXP; SKIP('Then'); SUITEINST; SKIP('else'); SUITEINST; SKIP('endif');  
while: SKIP('while'); EXP; SKIP('loop'); SUITEINST; SKIP('endloop');  
repeat: SKIP('repeat'); SUITEINST; SKIP('until'); EXP; SKIP('endloop');  
others: ERREUR();

end

end INST;

procedure SUITEINST is

begin

INST;

SUITEINST';

end SUITEINST;

procedure SUITEINST' is

begin

switch NEXTS

; : SKIP(';'); INST; SUITEINST';

endelse... : NULL;

others: ERREUR();

end SUITEINST';

TL  
TD 2

$$\left\{ \begin{array}{l} S' \xrightarrow{\circ} S\$ \\ S \xrightarrow{1} aAb \\ S \xrightarrow{2} bAb \\ A \xrightarrow{3} cAB \\ A \xrightarrow{4} \lambda \\ A \xrightarrow{5} a \\ B \xrightarrow{6} \lambda \end{array} \right] \rightarrow L(S) = aL(A)ab + bL(A)b$$

$$\left\{ \begin{array}{l} A \xrightarrow{1} \lambda \\ A \xrightarrow{2} a \\ A \xrightarrow{3} a \\ B \xrightarrow{6} \lambda \end{array} \right] \rightarrow L(A) = c^*(\lambda + a) = c^* + c^*a$$

$$\rightarrow L(B) = \{\lambda\}$$

$$\begin{aligned} L(G) &= L(S') = \\ &ac^*(\lambda + a)ab\$^k \\ &+ bc^*(\lambda + a)b\$^k \\ &= ac^*ab\$^k \\ &+ ac^*aab\$^k \\ &+ bc^*b\$^k \\ &+ bc^*ab\$^k \end{aligned}$$

$A \rightarrow cAB$

$A \rightarrow \lambda \Rightarrow A = cAB + \lambda + a$

$A \rightarrow a \quad A = \underbrace{cA}_{r_1 X} + \underbrace{\lambda + a}_{r_2} \xrightarrow{\text{ARDEN}} A = c^*(\lambda + a)$

Intuitivement :  $G \vdash LL(k), \forall k$ Pas de pb pour  $S'$  ni  $B$ Pour  $S$ , dès que  $k \geq 1$  on sait décider s'il faut utiliser la règle 1 ou 2.Pour  $A$ , dès que  $k \geq 1$  on sait choisir la règle 3 ou 5

ruban	pile
accab\\$ <sup>k</sup>	$S'$
accab\\$ <sup>k</sup>	$S\$^k$
accab\\$ <sup>k</sup>	<del><math>\lambda Aab\\$^k</math></del> 1
accab\\$ <sup>k</sup>	<del><math>\lambda ABab\\$^k</math></del> 3
cab\\$ <sup>k</sup>	<del><math>\lambda ABBab\\$^k</math></del> 3

Pb: choisir entre ④ et ⑤ si on a  $A$  au sommet de pile en regardant  $k$  symboles sur le ruban

④ ?      ⑤ ?      même en lisant  $ab\$^{k-2}$ , on ne sait pas choisir car on ne sait pas ce qu'il y a dans la pile au moment où on lit le ruban et le sommet de pile  $\rightarrow$  impossible de savoir si le a sera dépiler plus tard ou s'il faut le dépiler avec la règle ⑤

• Vérifier  $G \vdash LL(k), \forall k$  par calcul des klookahead

$$\bullet \text{klookahead } (A \rightarrow cAB) = \text{first}_k(cAB \cdot \text{follow}_k(\lambda)) = c \cdot \text{first}_{k-1}(AB \cdot \text{follow}_k(\lambda)) = E_0$$

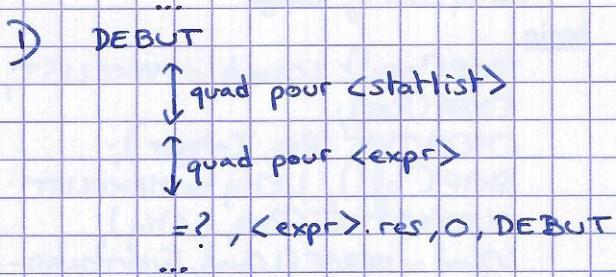
$$\bullet \text{klookahead } (A \rightarrow \lambda) = \text{first}_k(\lambda \cdot \text{follow}_k(\lambda)) = \{ab\$^{k-2}, b\$^{k-1}\} = E_1$$

$$\text{follow}_k(\lambda) = ab \cdot \underbrace{\text{follow}_k(S)}_{\$^{k-2}} + b\$^{k-1}$$

$$E_1 \cap E_2 = \{ab\$^{k-2}\}$$

$$\text{follow}_k(\lambda) = \{ab\$^{k-2}, b\$^{k-1}\}$$

$$\bullet \text{klookahead } (A \rightarrow a) = \text{first}_k(a \cdot \text{follow}_k(\lambda)) = \{aab\$^{k-3}, ab\$^{k-2}\} = E_2$$



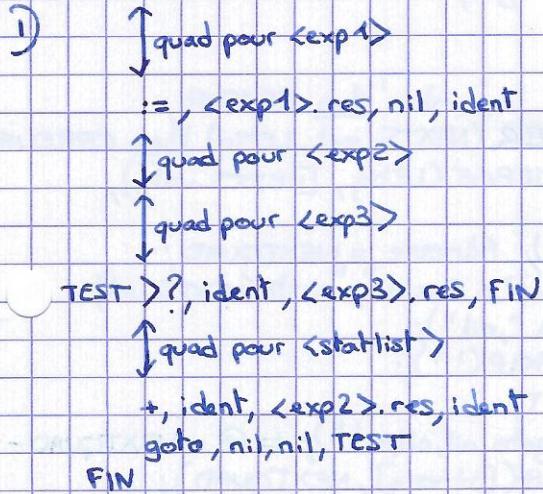
2). Vérifier le type de <expr>.res = booléen  
 $\Rightarrow \text{CHECKTYPE}$

- Référence en arrière à DEBUT  
 $\Rightarrow$  mémoriser l'e

### 3) procédure REPEATSTAT is

```

Debut: Integer;
Res: String;
begin
  SKIP ('repeat');
  Debut := NEXTQUAD;
  STATLIST;
  SKIP ('until');
  EXP (Res);
  CHECKTYPE (Res, Boolean);
  GEN ("=? ", " ∧ Res ∧ ", 0, " ∧ str(Debut)");
  SKIP ('endrepeat');
end REPEATSTAT;
  
```



2). Vérifier le type Integer pour <exp1>.res  
 $\Rightarrow \text{CHECKTYPE}$   
 $<\exp2>.res$   
 $<\exp3>.res$

- Référence en avant à FIN  
 $\Rightarrow$  mémoriser l'adresse dans une variable  
 $\Rightarrow$  engendrer le quadruplet incomplet  
 $\Rightarrow$  maj du champ adresse quand on connaît l'adresse de FIN  $\Rightarrow$  BACKPATCH

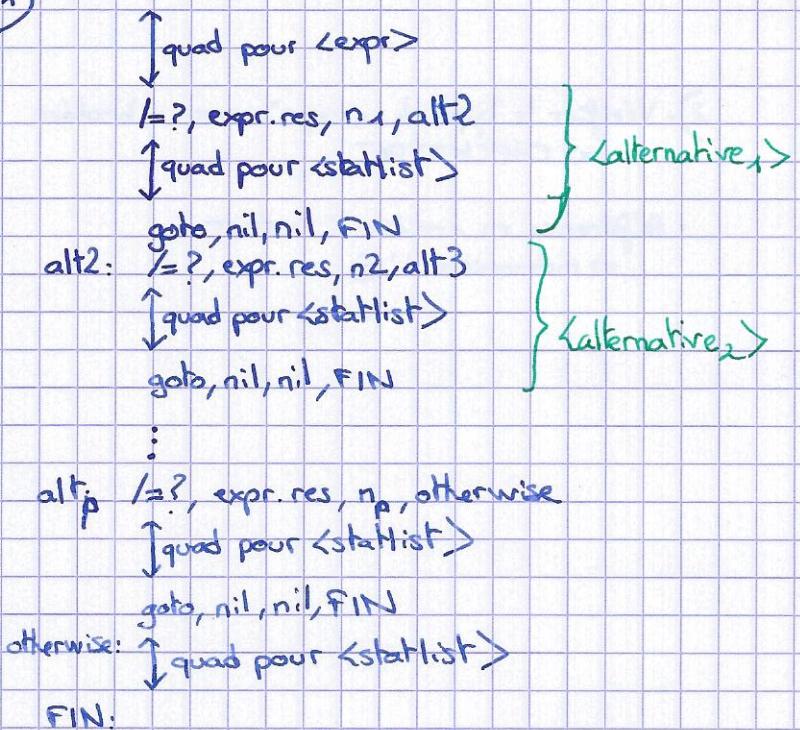
- Référence en arrière à TEST  
 $\Rightarrow$  mémoriser l'adresse dans une variable

3) procédure FORSTAT is

```

Res1, Res2, Res3: String      ident: String
Test: Integer
begin
  for SKIP ('for');            $\rightarrow$  type de la prochaine US
  ident | if NEXTS != 'ident' then
            ERREUR();
            ident := NEXTS.string, attribut syntaxique
            qui contient la chaîne
            SCAN();
            := (SKIP (':='));
            (CALCULÉ PAR AL)
  exp1 | EXP (Res1);
        | CHECKTYPE (Res1, Integer);
        | GEN ("=? ", " ∧ Res1 ∧ ", nil, " ∧ ident"),
  step | SKIP ('step');
  exp2 | EXP (Res2);
        | CHECKTYPE (Res2, Integer);
  until | SKIP ('until');
  exp3 | EXP (Res3);
        | CHECKTYPE (Res3, Integer);
        | Test := NEXTQUAD
        | GEN (">?", " ∧ ident ∧ ", " ∧ Res3 ∧ ", " ∧ nil");
        | do (SKIP ('do'));
        | statlist (STATLIST;
        | GEN ("+", " ∧ ident ∧ ", " ∧ Res2 ∧ ", " ∧ ident");
        | GEN ("goto, nil, nil, " ∧ str(Test));
        | BACKPATCH ([Test], NEXTQUAD);
  endfor (Skip ('endfor'));
end FORSTAT.
  
```

1)

{ alternative<sub>1</sub> }  

{ alternative<sub>2</sub> }

3) procedure CASESTAT is

Res: String  
LETiq: List of Integer  
begin  
SKIP('case'); LQuad := MAKELIST;  
EXPR(Res);  
SKIP('of'); LETiq := MAKELIST;  
ALTERNATIVE(Res, LETiq);  
LQuad := MERGE(LQuad, [NEXTQUAD - 1]);  
while NEXTS = ';' loop  
SKIP(';'); ALTERNATIVE(Res, LETiq);  
LQuad := MERGE(LQuad, [NEXTQUAD - 1]);  
endloop  
SKIP('otherwise');  
STATLIST;  
SKIP('endcase');  
BACKPATCH(LQuad, NEXTQUAD);  
end CASESTAT;

voir proc alternative

2) Références en avant à FIN

⇒ engendrer une liste de quad incomplets  
+ BACKPATCH

Expr.res de type Integer ?

⇒ CHECKTYPE

Référence en avant à alt:

⇒ engendrer le quad incomplet  
+ BACKPATCH

Vérifier qu'il n'y a pas 2 fois le même nombre  
dans 2 alt différentes

⇒ liste d'étiquettes (nombres)  
+ MEMBER (rajouté dans boîte à outil)

procedure ALTERNATIVE(Res: in String,  
LETiq: in out List of Integer) is

Adresse: Integer;

begin

if NEXTS != 'number' then ERREUR;  
if MEMBER(NEXTS.val, LETiq) then ERREUR;  
LETiq := MERGE(LETiq, [NEXTS.val]);

Adresse := NEXTQUAD;  
GEN("I = ?" ^ Res ^ ", " ^ str(NEXTS.val)  
^ ", nil");  
SCAN; SKIP(':'');  
STATLIST;  
GEN("goto, nil, nil, nil");  $\Leftarrow @ = \text{NEXTQUAD} - 1$   
BACKPATCH([Adresse], NEXTQUAD);

end ALTERNATIVE;

## I - Méthode de la descente récursive

### 2) • Référence en arrière à Début

⇒ enregistrer l'adresse du 1<sup>er</sup> quad engendré

- Référence en avant à Fin

⇒ listes de quad + BACKPATCH

⇒ une liste pour chaque boucle

- Expr. Rés de type Booléen ?

⇒ CHECKTYPE

- Pas de boucles imbriquées de m nom (étiquette)

Sortie de boucle : le nom doit exister

⇒ gérer les noms de boucles

- À chaque nom de boucle

⇒ vérifier qu'on n'est pas déjà dans une boucle de m nom

⇒ lui associer le type Etiquette

⇒ lui associer une liste de quad incomplets pour aller à Fin de ce nom là

⇒ Table des Symboles (TDS)

Rajout Boîte à Outils : SEARCHBIS (S: String, P: out Integer, type)  
 ENTERBIS (..., type)  
 CANCELBS (..., type)

procedure LOOPSTAT is

P: Integer

Début: Integer

nomBoucle: String

begin

```

        if NEXTS != 'ident' then ERREUR;
        SEARCHBIS(NEXTS.chaine, P, 'Etiquette');
        if P != 0 then ERREUR;
        ENTERBIS(NEXTS.chaine, P, 'Etiquette');
        TDS[P].lqi := MAKELIST;
        nomBoucle := NEXTS.chaine
        SCAN; SKIP(':'), SKIP('loop');
        Début := NEXTQUAD;
        STATLIST; SKIP('endloop');
        GEN("goto, nil, nil, " & str(Début));
        BACKPATCH(TDS[P].lqi, NEXTQUAD);
        CANCELBS(nomBoucle, 'Etiquette');
    end LOOPSTAT;
```

TDS:	nom	type	lqi
1			
P	A	Etiquette	0 → [---]

procedure EXITSTAT is

Rés: String

P: Integer

begin

```

        SKIP('when');
        EXP(Rés); CHECKTYPE(Rés, Booléen);
        SKIP('exit');
        if NEXTS != 'ident' then ERREUR;
        SEARCHBIS(NEXTS.chaine, P, 'Etiquette');
        if P = 0 then ERREUR;
        SCAN;
        TDS[P].lqi := MERGE(TDS[P].lqi,
            [NEXTQUAD]);
        GEN("=?," & Rés & ", 1, nil");
    end EXITSTAT;
```