

# $Traduction \ de \ languages$

M1 Informatique — Développement Logiciel Semestre 7

# Table des matières

2.1  .	1	Ana	alyse Lexicale – Système d'équations de langages	3
1.1.2       1.1.3 Langage ALGOL60       4         1.2 Systèmes d'équations algébriques de langages       5         1.2.1 G1       7         1.2.2 G2       8         2 Gramaires LL(1) – Descente récursive       9         2.1 Firsts       9         2.1.2 Follows       9         2.2 Clubres       9         2.2 Vérifions pour R, T et P       10         2.2.3 Procédures de descente récursive       12         2.3 Procédure de descente récursive       12         3 Génération de code en quadruplets       14		1.1	Analyse lexicale	3
1.1.3 Langage ALGOL60       4         1.2 Systèmes d'équations algébriques de langages       2         1.2.1 G1       2         1.2.2 G2       8         2 Gramaires LL(1) – Descente récursive       9         2.1       9         2.1.1 Firsts       9         2.1.2 Follows       9         2.2       10         2.2.1 LL(1)?       10         2.2.2 Vérifions pour R, T et P       10         2.2.3 Procédures de descente récursive       11         2.3       12         2.3.1 Procédure de descente récursive       12         3 Génération de code en quadruplets       12			1.1.1	3
1.2       Systèmes d'équations algébriques de langages         1.2.1       G1         1.2.2       G2         2       Gramaires LL(1) – Descente récursive         2.1       9         2.1.1       Firsts         2.1.2       Follows         2.2       10         2.2.1       LL(1)?         2.2.2       Vérifions pour R, T et P         2.2.3       Procédures de descente récursive         2.3       12         2.3.1       Procédure de descente récursive         3       Génération de code en quadruplets			1.1.2	3
1.2.1 G1       3         1.2.2 G2       8         2 Gramaires LL(1) – Descente récursive       9         2.1			1.1.3 Langage ALGOL60	4
1.2.2 G2       8         2 Gramaires LL(1) – Descente récursive       9         2.1		1.2	Systèmes d'équations algébriques de langages	7
2 Gramaires LL(1) – Descente récursive       9         2.1       9         2.1.1 Firsts       9         2.1.2 Follows       9         2.2.1 LL(1)?       10         2.2.2 Vérifions pour R, T et P       16         2.2.3 Procédures de descente récursive       17         2.3       15         2.3.1 Procédure de descente récursive       12         3 Génération de code en quadruplets       14			1.2.1 G1	7
2.1       9         2.1.1 Firsts       9         2.1.2 Follows       9         2.2       10         2.2.1 LL(1)?       10         2.2.2 Vérifions pour R, T et P       10         2.2.3 Procédures de descente récursive       12         2.3       12         2.3.1 Procédure de descente récursive       12         3 Génération de code en quadruplets       14			1.2.2 G2	8
2.1.1 Firsts       9         2.1.2 Follows       9         2.2       10         2.2.1 LL(1)?       10         2.2.2 Vérifions pour R, T et P       10         2.2.3 Procédures de descente récursive       12         2.3       15         2.3.1 Procédure de descente récursive       15         3 Génération de code en quadruplets       14	2	Gra	$ m amaires~LL(1)-Descente~r\'{e}cursive$	9
2.1.2 Follows       9         2.2       10         2.2.1 LL(1)?       10         2.2.2 Vérifions pour R, T et P       10         2.2.3 Procédures de descente récursive       11         2.3       12         2.3.1 Procédure de descente récursive       12         3 Génération de code en quadruplets       14		2.1		9
2.2       10         2.2.1       LL(1)?       10         2.2.2       Vérifions pour R, T et P       10         2.2.3       Procédures de descente récursive       12         2.3       12         2.3.1       Procédure de descente récursive       12         3       Génération de code en quadruplets       14			2.1.1 Firsts	9
2.2.1 LL(1)?       10         2.2.2 Vérifions pour R, T et P       10         2.2.3 Procédures de descente récursive       12         2.3       12         2.3.1 Procédure de descente récursive       12         3 Génération de code en quadruplets       14			2.1.2 Follows	9
2.2.2 Vérifions pour R, T et P       10         2.2.3 Procédures de descente récursive       12         2.3       15         2.3.1 Procédure de descente récursive       15         3 Génération de code en quadruplets       14		2.2		10
2.2.3 Procédures de descente récursive       15         2.3			2.2.1 LL(1)?	10
2.3       15         2.3.1       Procédure de descente récursive       15         3       Génération de code en quadruplets       14			2.2.2 Vérifions pour R, T et P	10
2.3.1 Procédure de descente récursive			2.2.3 Procédures de descente récursive	11
3 Génération de code en quadruplets 14		2.3		12
			2.3.1 Procédure de descente récursive	12
$3.1  \dots  1^{2}$	3	Gén	nération de code en quadruplets	14
		3.1		14

# 1

# Analyse Lexicale – Système d'équations de langages

# 1.1 Analyse lexicale

$$C = \{0, 1, \cdots, 8, 9\}$$

### 1.1.1

$$L = O^*2\$(0+1)^+ + 0^*3\$(0+1+2) + \dots + 0^*10\$(0+1+\dots+9)^+ + (0+\dots+9)^+ (1.1)$$
  
=  $C^+\$c^+ + c^+ = cc^*\$cc^* + cc^*$  (1.2)

- R
- Solution (1.1) est à déterminiser
- Solution (1.2) il faudra ajouter des contraintes

## 1.1.2

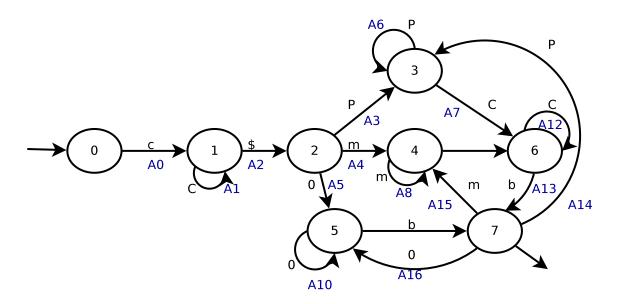


FIGURE 1.1 – Automate fini

```
carrcour ; val(carrcour)
  variables:
    base; -- valeur décimale de la base avant '$'
     ecart; -- Valeur décimale d'un écart
     cpt; -- Compter les '+' ou '-' ou '0' successifs
5
     signe; -- Indique si on a des '+' ou '-' ou '0'
         +1 si on a '+'
7
         -1 si on a '-'
        0 si on a '0'
        {base := base * 10 + val(valcarcour);}
11
  A2:
        \{\emptyset\}
  A3:
        \{ cpt := 1; signe := 1; \setminus \}
       {cpt := 1 ; signe := -1;\}
       {cpt := 1; signe := 0; \}
  A6: {cpt := cpt + 1;\}
  A7: {ecart := val(carcours)\}
  A8 = A10 = A6
  A9 = A7
  A12: \{\text{ecart := ecart * 10 + val(carcours)}\}
  A13: {
21
       tantque cpt != 0 faire
         printf(base + (signe * ecart));
23
         cpt := cpt - 1;
      fin tanque;
      }
   A11 = A13
27
   A14 = A3
    A15 = A4
    A16 = A5
```

# 1.1.3 Langage ALGOL60

 ${f R}$  Une écriture en grammaire est également une écriture en langages, ainsi :

$$P = \begin{cases} A & \to & \alpha_1 \\ A & \to & \alpha_2 \\ & \dots \\ A & \to & \alpha_n \end{cases} \Leftrightarrow L(A) = \alpha_1 + \alpha_2 + \dots + \alpha_n$$

#### 1.1.3.1 Définition de la grammaire

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

$$N = \{I, J, K, L, M, N, O\}$$

$$S = O$$

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

#### 1.1.3.2 Système d'équation de langage égaux à G

$$\begin{cases}
I &= c + cI \\
J &= I + sI \\
K &= I \\
L &= \underbrace{I + K + IK}_{Problèmeder\'egularit\'e?} \\
M &= eJ \\
N &= £ + M + LM \\
O &= N + sN
\end{cases}$$

I Entier signé

**c** 
$$c \in \{0 \cdots 9\}$$

J Entier

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

K Partie fractionnaire

L Nombre décimal

 $\mathbf{M}$  Partie exponentiel

N Nombre non signé

O Nombre



Un automate fini ne peut reconnaître que les langages réguliers, qui sont engendrés par des grammaires linéaires à droite.

L'union de langages régulier engendre un langage régulier, de même le produit de deux langages réguliers donnent un langage régulier.

Dans notre cas, L ne pose donc aucun problème de régularité étant donné que I et K sont des langages réguliers ainsi, l'union et le produit de langages réguliers engendrant des langages réguliers, L sera régulier.

Le problème pourrait se poser pour L mais aussi pour N et M: la réponse étant la même.

O est régulier, on peut donc trouver un automate finis le reconnaissant.

$$\begin{split} I &= c(\lambda + I)cP \\ P &= I + \lambda = cP + \lambda \\ J &= I + sI = cP + sI \\ K &= .I \\ L &= I + K + IK = \underline{c}P + .I + \underline{c}PK = \underline{c}(\underline{P + PK}) + .I \\ &= cQ + .I \\ Q &= P + PK = cP + \lambda + (cP + \lambda)K = cP + \lambda + cPK + K = \underline{c}(\underline{P + PK}) + .I + \lambda \\ &= cQ + .I + \lambda \\ M &= eJ \\ N &= L + M + LM = cQ + .I + eJ + (cQ + .I)M = cQ + .I + eJ + cQM + .IM + .IM \\ &= \underline{c}(\underline{Q + QM}) + .(\underline{I + IM}) + eJ \\ R &= Q + QM = cQ + .I + \lambda + cQM + .IM + eJM = \underline{c}(Q + QM) + .(I + IM) + eJ + \lambda \\ S &= I + IM = cP + cPM = \underline{c}(P + PM) \\ T &= P + PM = cP + \lambda + cPM + \underbrace{\frac{M}{eJ}}_{eJ} = \underline{c}(P + PM) = e + \lambda \\ Q &= N + sN = cR + .S + eJJ + sN \end{split}$$

D'où le système d'équation de langage suivant :

$$\begin{cases}
I &= cP \\
P &= cP + \lambda \\
J &= cP + sI \\
K &= J \\
L &= cQ + J \\
Q &= cQ + J + \lambda \\
M &= eJ \\
N &= cR + S + eJ \\
R &= cR + s + eJ + \lambda \\
S &= cT \\
T &= cT + eJ + \lambda \\
O &= cR + s + eJ + sN
\end{cases}$$

L'axiome O est un état initial et  $\{T,R,Q,P\}$  sont des états finaux.

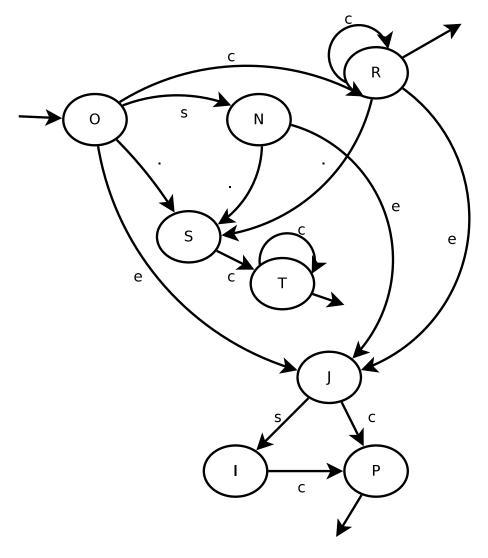


FIGURE 1.2 – Automate fini

# 1.2 Systèmes d'équations algébriques de langages

**R** 3

3 théorèmes :

Arden  $X = r_1X + r_2 \Rightarrow X = r_1^*r_2$ Arden Bis  $X = Xr_1 + r_2 \Rightarrow X = r_2r_1^*$ 

 $\mathbf{AnBn} \ X = Yr_1 + r_{\Rightarrow} X = r_2 r_1^*$ 

# 1.2.1 G1

$$S = S \underbrace{a}_{r_1} + \underbrace{b}_{r_2} \Rightarrow S$$

$$\stackrel{Ardenbis}{=} ba* = ba^n n \ge 0$$

# 1.2.2 G2

$$\begin{cases} S = aSb + T \stackrel{AnBn}{\Longrightarrow} S = a^n Tb^n, n \ge 0 = a^n b^+ b^n n \ge 0 \\ T = Tb + b \stackrel{Ardenbis}{\Longrightarrow} \end{cases}$$

# Gramaires LL(1) – Descente récursive

2.1

#### 2.1.1 Firsts

$$S' \rightarrow S\$$$
 (2.1)  
 $S \rightarrow bRS$  (2.2)  
 $S \rightarrow RcSa$  (2.3)  
 $S \rightarrow \lambda$  (2.4)  
 $R \rightarrow acR$  (2.5)  
 $R \rightarrow b$  (2.6)

$$\begin{array}{rcl} first_1(R) & = & \{a,b\} \\ first_1(S) & = & \{a,b,\lambda\} \\ first_2(R) & = & \{ac,b\} \\ first_2(S) & = & \{\underbrace{\lambda}_{2.3},\underbrace{bb}_{2.2+R}\} \end{array}$$

Pour le  $first_2(R)$ , ac est le préfixe de longueur 2 des mots prouits par R, et b est le mot de longueur inférieur ou égale à 2 produit par R.

Nous avons chercher les first intuitivement, cependant nous nous sommes servis de la formule suivante :  $first_k(\alpha_1\alpha_2\cdots\alpha_n)=first_k(first_k(\alpha_a)first(\alpha_2)\cdots first_k(\alpha_n)$ 

## 2.1.2 Follows

 $R follow_k(y) = \cup first_k(Sfollow_k(x)))$ 

#### k = 2 Pour s:

- 1.  $2lookahead(S \rightarrow bRS)first_2(bRSfollow_2(S)) = bfirst_1(RSfollow_2(S)) = \{ba, bb\} = E_1$
- 2.  $2lookahead(S \rightarrow RcSa) = first_2(Sa = follow_2(S)) = \{ac, bc\}$
- 3.  $2lookahead(S \rightarrow RcSa) = first_2(RcSa = follow_2(S)) = first_2(first_2(R)c\cdots) = \{ac, bc\}$

### 2.2

Grammaire d'axiome  $\theta$  puis augmentée :

$$\begin{cases} S' & \to & \theta \$^R \\ \theta & \to & cR \mid .S \mid eJ \mid sN \\ R & \to & cR \mid .S \mid J \mid \lambda \\ S & \to & cT \\ J & \to & cP \mid sI \\ N & \to & cR \mid .S \mid eJ \\ T & \to & cT \mid eJ \mid \lambda \\ P & \to & cP \mid \lambda \\ I & \to & cP \end{cases}$$

### 2.2.1 LL(1)?

$$1lookahead(\theta \to cR) = \{c\} = \{0, 1, \cdots, 9\}$$
$$1lookahead(\theta \to .S) = \{.\}$$
$$1lookahead(\theta \to eJ) = \{e\}$$
$$1lookahead(\theta \to sN) = \{s\} = \{+, -\}$$

# 2.2.2 Vérifions pour R, T et P

#### 2.2.2.1 R?

$$1lookahead(R \to \lambda) = first_1(\lambda.follow_1(R)) = \{\$\}$$
  
 $1lookahead(R \to cR) = \{c\}$   
 $1lookahead(R \to .S) = \{.\}$   
 $1lookahead(R \to eJ) = \{e\}$ 

OK, pas de conflit

#### 2.2.2.2 *P*?

$$follow_2(P) = \{\$\}$$

Ok, pas de conflit.

#### 2.2.2.3 T?

$$1 lookahead(T \to \lambda) = first_1(\lambda follow_1(T)) = \{\$\}$$
  
 
$$\Rightarrow follow_1(T) = follow_1(S) = follow_1(\theta) \cup follow_1(R) \cup follow_1(N) = \{\$\}$$

Ok, pas de conflit



Si on a une grammaire linéaire à droite, on est sûr qu'elle est LL(1).

Quelque soit la règle de l'ensemble des règles de production P,  $\forall A: A \to \lambda, A \to xB$ 

#### 2.2.3 Procédures de descente récursive

A chaque  $A \in N$  on associe une procédure de nom A qui contient « l'image de  $\beta$  ».

$$Si \ A \to \beta \ avec \begin{cases} \beta = x \in X \implies SKIP('x') \\ \beta = \lambda \implies NULL \\ \beta = B \in N \implies B \\ \beta_1 = \beta_1\beta_2 \cdots \beta_n \implies image(\beta_1); image(\beta_2); \cdots; image(\beta_n); \end{cases}$$

$$\begin{array}{c} Si \ A \ \rightarrow \ \beta_1 \\ Si \ A \ \rightarrow \ \beta_1 \\ Si \ A \ \rightarrow \ \beta_1 \\ Si \ A \ \rightarrow \ \beta_1 \end{array} \right\} \Longleftrightarrow \begin{array}{c} Ilookahead(A \rightarrow \beta_1) \ : \ image\beta_1 \\ Ilookahead(A \rightarrow \beta_2) \ : \ image\beta_2 \\ \vdots \\ Ilookahead(A \rightarrow \beta_n) \ : \ image\beta_n \\ Others \ : \ ERREUR: \end{array}$$

```
procedure \theta is
                                        begin
  procedure S' is
                                           switch NEXTS
  begin
                                              number : SKIP(number.val); R;
    scan;
                                             |. : SKIP('.'); S;
                                             | e : SKIP('e') ; J;
    SKIP('$');
                                              + : SKIP('+'); N;
 end S'
                                             | - : SKIP('-'); N;
                                             | others : ERREUR;
                                       end 	heta
  procedure R is
  begin
                                        procedure S is
    switch NEXTS
                                        begin
       number : SKIP(number.val); R;²
4
                                          SKIP(number.val);
      |. : SKIP('.'); S;
                                          Τ;
      |e : SKIP('e') ; J ;
6
                                        end S
      | $ : NULL;
8
      |others : ERREUR;
```

2.3

```
< program > \rightarrow program < suiteDct > begin < switchInst > end
< suiteInst > \rightarrow < inst > | < suiteInst >; < inst >
< inst > \rightarrow if < exp > then < suiteInst > else < suiteInst > endif
|while < exp > loop < suiteInst > endloop
|repeat < suiteInst > until < exp > endloop
< suiteDcl > \rightarrow \cdots
< exp > \rightarrow \cdots
```

```
 \begin{array}{l} \hline \textbf{R} \\ \hline - \text{ Non terminal} : < \cdots > \\ \hline - \text{ Terminal} : X = \{program, begin, end, if, \cdots \} \end{array}
```

#### 2.3.1 Procédure de descente récursive

#### 2.3.1.1 LL(1)?

```
Pour < program > : ok

Pour < inst > : ok

1 lookahead() = \{if\}
1 lookahead() = \{while\}
1 lookahead() = \{repeat\}
```

Pour < suiteInst > :

```
1look(< suiteInst > \rightarrow < inst >) = \{if, while, repeat\}
1look(< suiteInst > \rightarrow < suiteInst >; < inst >) = \{if, while, repeat\}
```

La grammaire est non LL(1) car elle est récursive à gauche!, à cause de <  $suiteInst> \rightarrow <$  suiteInst> ; < inst>

On peut la transformer en une grammaire linéaire à droite grâce à Arden.

$$A \rightarrow A\beta | \alpha \stackrel{ArgenBis}{\Longrightarrow} = \alpha \beta^*$$

Éliminer la récursivité à gauche :

$$L(A) = \alpha \beta^* = \alpha L(A')$$
  
 $L(A') = \beta^* \lambda \Longrightarrow \beta L(A') + \lambda$ 

Donc  $< suiteInst > \rightarrow < inst > | < suiteInst >; < inst > devient :$ 

$$\overbrace{\langle suiteInst\rangle}^{A} \rightarrow \overbrace{\langle inst\rangle \langle suiteInstPrime\rangle}^{\alpha} \\ \langle suiteInstPrime\rangle \rightarrow ; \langle inst\rangle \langle suiteInstPrime\rangle | \lambda$$

On vérifie de nouveau qu'elle soit bien LL(1);

Pour <suiteInst> pas de problème, car une seule règle.

Pour < suiteInstPrime > :

```
1lookahead(< suiteInstPrime > \rightarrow; < inst > \leftrightarrow) = \{;\}1lookahead(< suiteInstPrime > \rightarrow \lambda) = first_1(\lambda.follow_1(< suiteInstPrime >)= \{end, endif, else, endloop, until\} = E_2.
```

Donc G' est LL(1).

```
procedure SUITEINSTPRIME is

procedure SUITEINST is
begin

switch NEXTS

;': SKIP(';'); INST; SUITEINSTPRIME;

SUITEINSTPRIME;

end SUITEINST

tend SUITEINSTPRIME;

suitch NEXTS

ightharpoorus

switch NEXTS

ightharpoorus

ightharpoorus

switch NEXTS

ightharpoorus

switch NEXTS

ightharpoorus

ightharpoorus

switch NEXTS

ightharpoorus

ightharpoorus

switch NEXTS

ightharpoorus

ightharpoorus

switch NEXTS

ightharpoorus

ightharpoorus

ightharpoorus

switch NEXTS

ightharpoorus

ightharpoorus
```

#### k = 2 Pour s:

- 1.  $2lookahead(S \rightarrow bRS)first_2(bRSfollow_2(S)) = bfirst_1(RSfollow_2(S)) = \{ba, bb\} = E_1$
- 2.  $2lookahead(S \rightarrow RcSa) = first_2(Sa = follow_2(S)) = \{ac, bc\}$
- 3.  $2lookahead(S \rightarrow RcSa) = first_2(RcSa = follow_2(S)) = first_2(first_2(R)c\cdots) = \{ac, bc\}$

# Génération de code en quadruplets

Mettre en œuvre un processus de traduction en quadruplets des structures de contrôle <repeatstat>, <forstat> et <casestat> par la méthode de la descente récursive.

## 3.1

```
procedure REPEATSTAT is
     // declaration
     DEBUT : Integer
3
     TEST : Integer
     RES : String
5
     // instruction
7
     begin
       SKIP('repeat');
       DEBUT := NEXTQUAD;
       EXP(Res);
11
       GEN("=?,"^Res^", 0, nil");
       SKIP('until');
13
       EXPR(RES);
       CHECKTYPE(Res, Boolean);
15
       STATLIST;
       SKIP('endloop');
17
```

```
GEN("goto, nil, nil, " ^str(debut));
BACKPATCH([TEST], NEXTQUAD);
end REPEATSTAT
```

