

Compilation M1

Félix Jamet, Mica Ménard

Avril 2018

Contents

1	Projet compil	2
1.1	Définitions	2
1.2	Schémas	2
1.3	Processus divers	3
1.3.1	Scan G_0	3
1.3.2	Scan GPL	3
1.3.3	Action G_0	3
1.4	Construction de la grammaire G_0	3
1.4.1	Notation B.N.F.	3
1.4.2	Règle 1	4
1.4.3	Règle 2	4
1.4.4	Règle 3	4
1.4.5	Règle 4	4
1.4.6	Règle 5	4
1.5	Structure de données	4
1.6	Construction des 5 Arbres	5
1.6.1	Fonctions Gen*	5
1.6.2	Arbres	6
1.7	Scan G_0	7
1.8	Action G_0	8
1.9	Exemple	9
1.9.1	Pile	9
1.9.2	Dictionnaires	9
1.9.3	Compilation	9
1.9.4	Arbre GPL	10
2	Grammaires LL(k)	11
2.1	First(N)	11
2.2	Follow(N)	11
2.3	Grammaire LL(1)	11
3	Tables S.R.	11
3.1	Algorithme Table Analyse L.R.	11
3.2	Génération automatique de la table SR	11
3.2.1	Opérateurs \doteq , $>$, et $<$	11
3.3	Exemple de génération de table S.R.	12
4	Types des grammaires	12

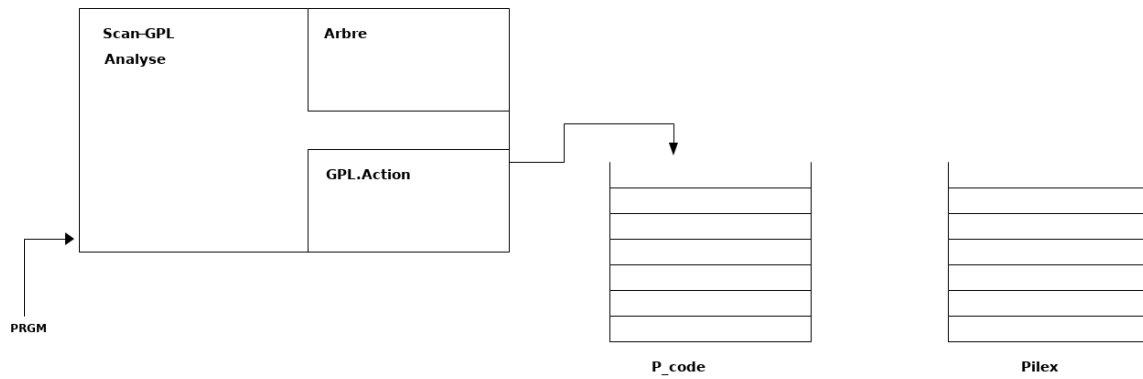


Figure 3: Actions de la grammaire GPL

1.3 Processus divers

1.3.1 Scan G_0

Scanne les

- éléments terminaux
- éléments terminaux

1.3.2 Scan GPL

Scanne les

- identificateurs
- nombres entiers
- symboles ($>$, $\#$, $[$, etc.)

1.3.3 Action G_0

Construit l'arbre GPL

1.4 Construction de la grammaire G_0

1.4.1 Notation B.N.F.

- $::= \Leftrightarrow \rightarrow$
- $[X] \Leftrightarrow X.X.X...X(n \text{ fois}), n \geq 0$
- $(/X/) \Leftrightarrow X \text{ ou Vide}$
- $/ \Leftrightarrow +$
- $concat \Leftrightarrow .$
- ' X ' correspond à un élément terminal

1.4.2 Règle 1

$$S \rightarrow [N.' \rightarrow' .E.',']';',$$

Une grammaire est forcément composée de plusieurs règles, séparées par des ‘,’ et terminée par un ‘;’.

1.4.3 Règle 2

$$N \rightarrow' IDNTER',$$

‘IDNTER’ signifie identificateur non terminal.

1.4.4 Règle 3

$$E \rightarrow T.['+' .T],$$

E est une expression qui peut être un terme ou un autre.

1.4.5 Règle 4

$$T \rightarrow F.['.' .F],$$

Un terme T peut être composé d’un seul facteur F ou de facteurs concaténés.

1.4.6 Règle 5

$$F \rightarrow' IDNTER' + ' ELTER' + ' ('.' .E.')' + ' ['.' .E.']' + ' (/' .E.' /), ;$$

1.5 Structure de données

Syntaxe maison...

```
Type Atomtype = (Terminal, Non-Terminal);
Operation = (Conc, Union, Star, UN, Atom); ##Atom = {IDNTER, ELTER}
PTR = ↑Node

Node = Enregistrement
    case operation of
    Conc: (left, right : PTR);
    Union: (left, right : PTR);
    Star: (stare: PTR);
    UN: (UNE : PTR);
    ATOM: (COD, Act : int ; AType: Atomtype);
    EndEnregistrement

A: Array [1..5] of PTR;
```

1.6 Construction des 5 Arbres

1.6.1 Fonctions Gen*

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

```
Fonction GenUnion(P1, P2 : PTR) : PTR;  
  var P : PTR;  
  début  
    New(P, union);  
    P↑.left := P1;  
    P↑.right := P2;  
    P↑.class := union;  
    GenUnion := P;  
  fin
```

```
Fonction GenStar(P1 : PTR) : PTR; ##0 ou n fois  
  var P:PTR;  
  début  
    New(P, star);  
    P↑.stare := P1;  
    P↑.class := star;  
    GenStar := P;  
  fin
```

```
Fonction GenUn(P1 : PTR) : PTR; ##0 ou une fois  
  var P:PTR;  
  début  
    New(P, un);  
    P↑.une := P1;  
    P↑.class := un;  
    GenUn := P;  
  fin
```

```
Fonction GenAtom(COD, Act : int, AType : Atomtype) : PTR  
  var P:PTR;  
  début  
    New(P, atom);  
    P↑.COD := COD;  
    P↑.Act := Act;  
    P↑.AType := AType;  
    GenAtom := P;  
  fin
```

1.6.2 Arbres

1. S

```
A[S] :=
  GenConc(
    GenStar(
      GenConc(
        GenConc(
          GenConc(GenAtom('N', '', NonTerminal),
            GenAtom('->', 5, Terminal)
          ),
          GenAtom('E', '', NonTerminal)
        ),
        GenAtom(',', , Terminal)
      ),
      GenAtom('; ', , Terminal)
    );
```

2. N

##Ajouts de ma part, je ne suis pas sûr des résultats :

```
A[N] := GenAtom('IDNTER', , Terminal);
```

3. E

```
A[E] := GenConc(
  GenAtom('T', '', NonTerminal),
  GenStar(
    GenConc(
      GenAtom('+', ?, Terminal),
      GenAtom('T', '', Terminal)
    )
  )
)
```

4. T

```
A[T] := GenConc(
  GenAtom('F', '', NonTerminal),
  GenStar(
    GenConc(
      GenAtom('.', ?, Terminal),
      GenAtom('T', '', Terminal)
    )
  )
)
```

5. F

```
A[F] := GenUnion(
  GenUnion(
    GenUnion(
      GenUnion(
        GenAtom('IDNTER', , Terminal),

```

```

        GenAtom('ELTER', , Terminal)
    ),
    GenConc(
        GenConc(
            GenAtom('(', ?, Terminal),
            GenAtom('E', '', NonTerminal)
        ),
        GenAtom(')', ?, Terminal)
    )
),
GenConc(
    GenConc(
        GenAtom('[', ?, Terminal),
        GenAtom('E', '', NonTerminal)
    ),
    GenAtom(']', ?, Terminal)
)
),
GenConc(
    GenConc(
        GenAtom('(', ?, Terminal),
        GenAtom('E', '', NonTerminal)
    ),
    GenAtom(')', ?, Terminal)
)
)
)

```

1.7 Scan G_0

```

fonction Analyse(P : PTR) : booléen
début
    case P↑.class of
        Conc: if Analyse(P↑.left) then Analyse := True
                else Analyse := Analyse(P↑.right);
        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
            Terminal: if P↑.cod = code then #cod = code ASCII
                début
                    Analyse := true;
                    if P↑.act != 0 then GO-action(P↑.act)
                    scanG0;
                fin
                    else Analyse := false;
            Non-Terminal: if Analyse(A[P↑.cod]) then
                début

```

```

        if P↑.act != 0 then G0-action(P↑.act);
        Analyse := true;
    fin
else Analyse := false;

fin

Main() #vérifie si une grammaire est correcte
{
    scan;
    if Analyse(A[s]) then write('OK');
}

```

1.8 Action G_0

De quoi a-t-on besoin ?

- Deux dictionnaires : *DicoT*, *DicoNT*
- Tableau *pile*[*I*] : Tableau de pointeurs

Remarque : les nombres du case correspondent aux actions associées aux numéros inscrits dans les arbres.

```

Procédure Action G0(Act : int);
var T1, T2 : PTR;
début
    case Act of
    1: Dépiler(T1);
        Dépiler(T2);
        A[T2↑.cod + 5] := T1; ##Arbres GPL commencent à 6
    2: Empiler(GenAtom(Recherche(DicoT), Action, CAtype)) ##donne la
        ##partie gauche d'une règle
        ##Recherche() stocke le token si non stocké dans dico
    3: Dépiler(T1);
        Dépiler(T2);
        Empiler(GenUnion(T2,T1))
    4: Dépiler(T1);
        Dépiler(T2);
        Empiler(GenConc(T2,T1))
    5: if CAtype = Terminal then
        Empiler(GenAtom(Recherche(DicoT), Action, Terminal))
        else
        Empiler(GenAtom(Recherche(DicoNT), Action, Terminal))
    6: Dépiler(T1);
        Empiler(GenStar(T1));
    7: Dépiler(T1);
        Empiler(GenUn(T1));

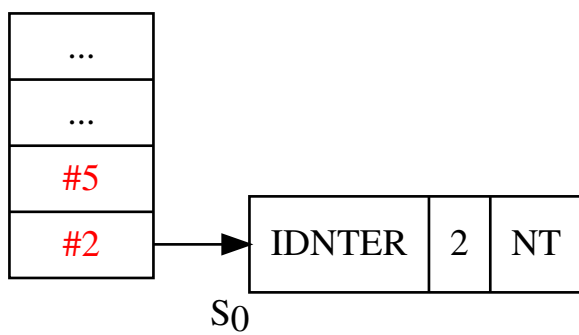
Pile : Array[1..50] : PTR;
DicoT, DicoNT: Dico;
Dico : Array[1..50] : String[10];

```

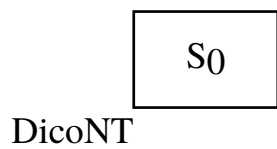
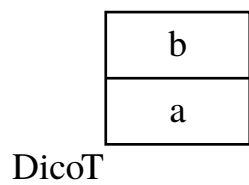

1.9 Exemple

GPL : $S_0 \rightarrow ['a']. 'b'$; ; Regex : a^nb

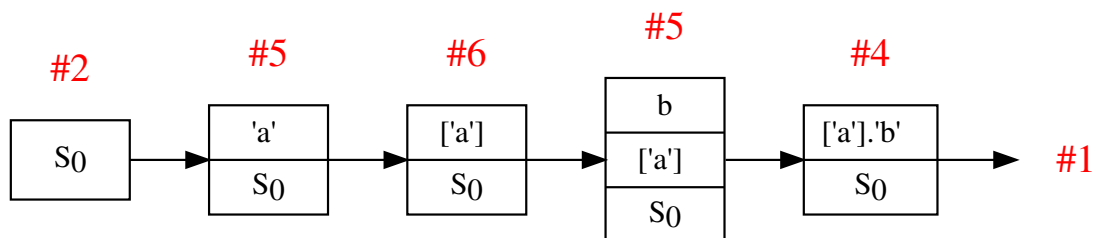
1.9.1 Pile



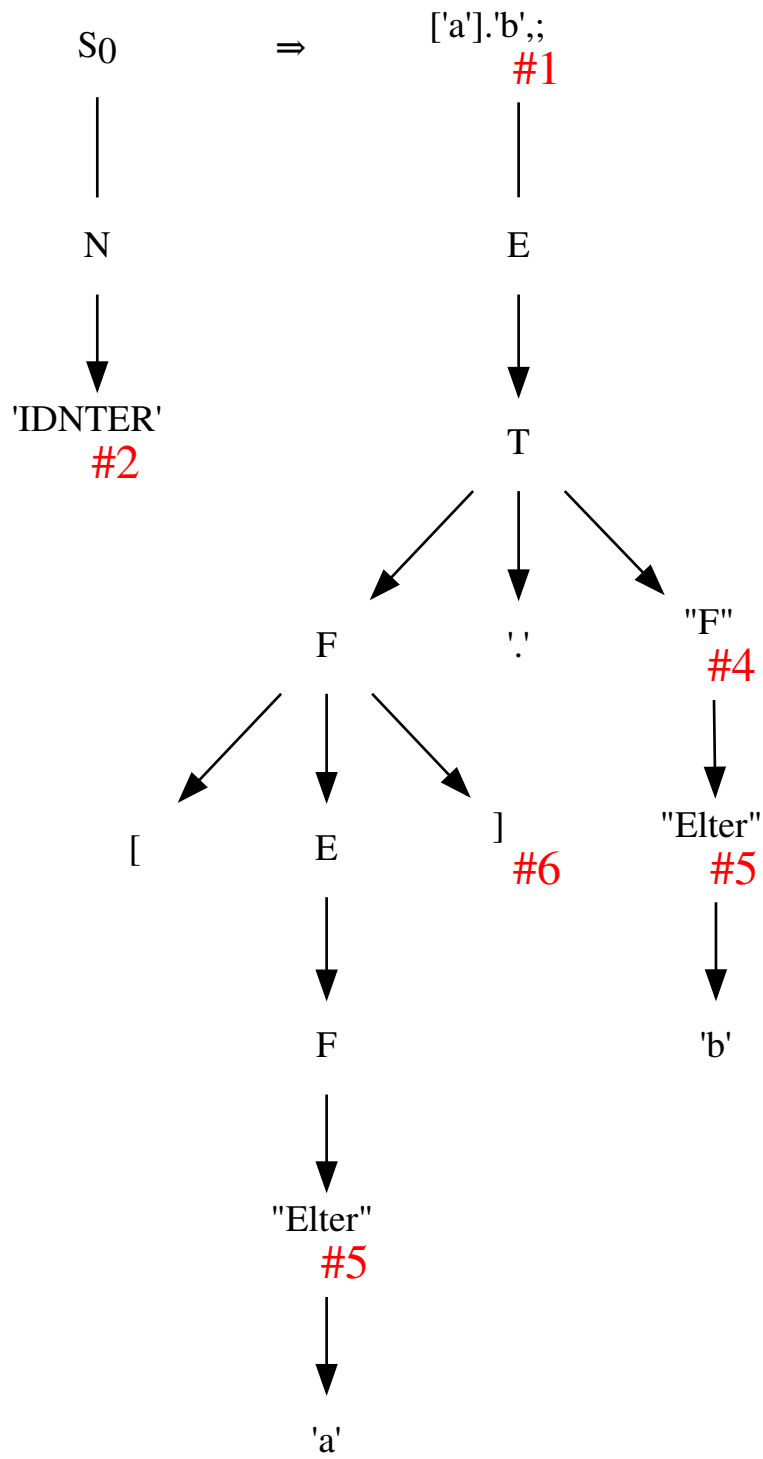
1.9.2 Dictionnaires



1.9.3 Compilation



1.9.4 Arbre GPL



2 Grammaires LL(k)

k est une mesure de l'ambiguïté. Représente le nombre de caractères qu'il est nécessaire de regarder pour déterminer quelle règle utiliser. Bien entendu, les règles LL(1) sont préférables.

2.1 First(N)

- Si $N \rightarrow A \dots$ alors $First(N) = First(A)$
- Si $N \rightarrow c \dots$ alors $First(N) = \{c\}$
- Si $N \rightarrow A.B \dots$ et si $A \xRightarrow{*} \epsilon$ alors $First(N) = First(B)$

Avec " $\xRightarrow{*}$ " signifiant "se derive en".

Il ne s'agit pas d'appliquer une règle à chaque fois, mais plutôt d'appliquer toutes les règles possibles.

2.2 Follow(N)

- Si $A \rightarrow \dots Nc \dots$ alors $Follow(N) = \{c\}$
- Si $A \rightarrow \dots NB \dots$ alors $Follow(N) = First(B)$
- Si $A \rightarrow N \dots$ alors $Follow(N) = Follow(A)$

Concernant la dernière règle, hippolyte a noté: - Si $A \rightarrow \dots N$ alors $Follow(N) = Follow(A)$

À déterminer.

2.3 Grammaire LL(1)

- si $A \rightarrow \alpha_1/\alpha_2/\dots/\alpha_n$ alors

$$Prem(\alpha_i) \cap Prem(\alpha_j) = \Phi, \forall i \neq j$$

- si $A \Rightarrow \epsilon$ on doit avoir $Prem(A) \cap Suiv(A) = \Phi$

Si une règle ne possède qu'une dérivation, la règle 1 ne s'applique pas. Si une règle ne possède pas de suiv, la règle 2 ne s'applique pas.

3 Tables S.R.

3.1 Algorithme Table Analyse L.R.

Shift Empiler le caractère; scan;

Reduce Remplacer la partie droite au sommet de la pile par la partie gauche ($A \rightarrow a$)

3.2 Génération automatique de la table SR

3.2.1 Opérateurs \doteq , $>$, et $<$

3.2.1.1 Shift

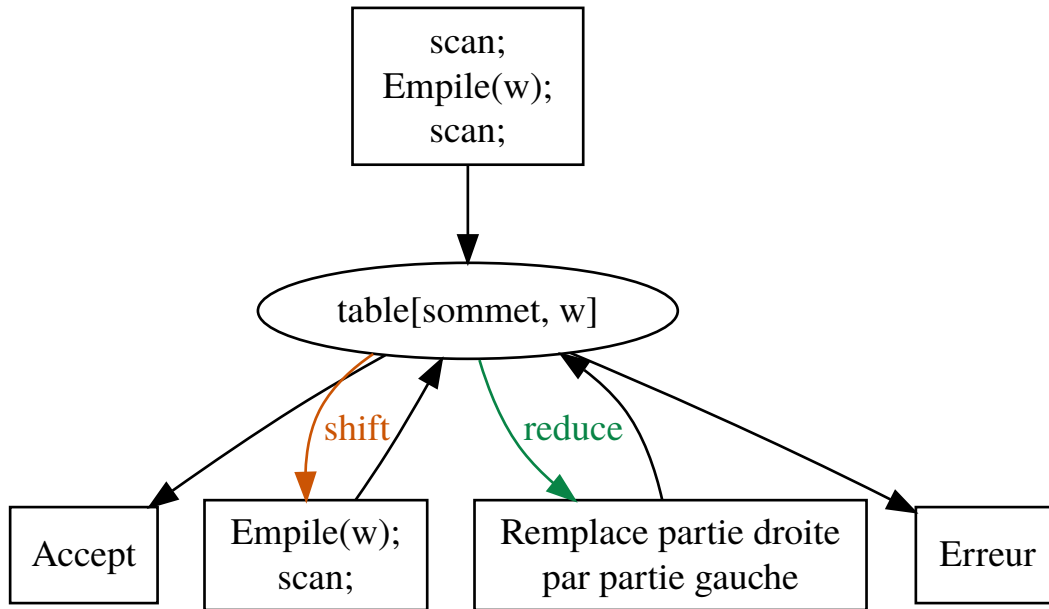


Figure 4: Algorithme Table Analyse L.R.

- $X \doteq Y$ si

$$A \rightarrow \dots X.Y \dots \in \mathcal{P}$$

- $X < Y$ si

$$A \rightarrow \dots X.Q \dots \in \mathcal{P}$$

$$\text{et } Q \overset{*}{\Rightarrow} Y$$

3.2.1.2 Reduce

- $X > Y$ si

$$A \doteq Y$$

$$\text{et } A \overset{*}{\Rightarrow} X$$

On peut remplir le tableau SR à partir des relations \doteq , $>$ et $<$:

- (ligne \doteq colonne) et (ligne $<$ colonne) se traduisent en (ligne Shift colonne)
- (ligne $>$ colonne) se traduit en (ligne Reduce colonne)

3.3 Exemple de génération de table S.R.

4 Types des grammaires

0 type c

1 type context sensitive CS $\gamma \rightarrow \beta$ avec $\gamma \leq \beta$

2 type context free CF $A \rightarrow B$ avec $A \in V_N, B \in V^+$

3 type régulière

$$\begin{cases} A \rightarrow aB \\ A \rightarrow a \end{cases} \quad \text{ou} \quad \begin{cases} A \rightarrow Ba \\ A \rightarrow a \end{cases}$$

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

l'intersection de deux langages de type x n'est pas forcément de type x.