

# Compilation M1

Félix Jamet, Mica Ménard

Avril 2018

## Contents

<b>1</b>	<b>Projet compil</b>	<b>2</b>
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 4 . . . . .	4
1.4.5	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 $G_0$ . . . . .	7
<b>2</b>	<b>Grammaires LL(k)</b>	<b>8</b>
2.1	First(N) . . . . .	8
2.2	Follow(N) . . . . .	8
2.3	Grammaire LL(1) . . . . .	8
<b>3</b>	<b>Tables S.R.</b>	<b>9</b>
3.1	Algorithme Table Analyse L.R. . . . .	9
3.2	Génération automatique de la table SR . . . . .	9
3.2.1	Opérateurs $\doteq$ , $>$ , et $<$ . . . . .	9
3.3	Exemple de génération de table S.R. . . . .	10
<b>4</b>	<b>Types des grammaires</b>	<b>10</b>

# 1 Projet compilo

## 1.1 Définitions

**GPL** Grammaire Petit Langage

**Scanner** analyseur lexical, découpe du texte en unités syntaxiquement corrects (tokens)

**Parseur** analyse syntaxique, s'assure que les tokens soient syntaxiquement corrects

## 1.2 Schémas

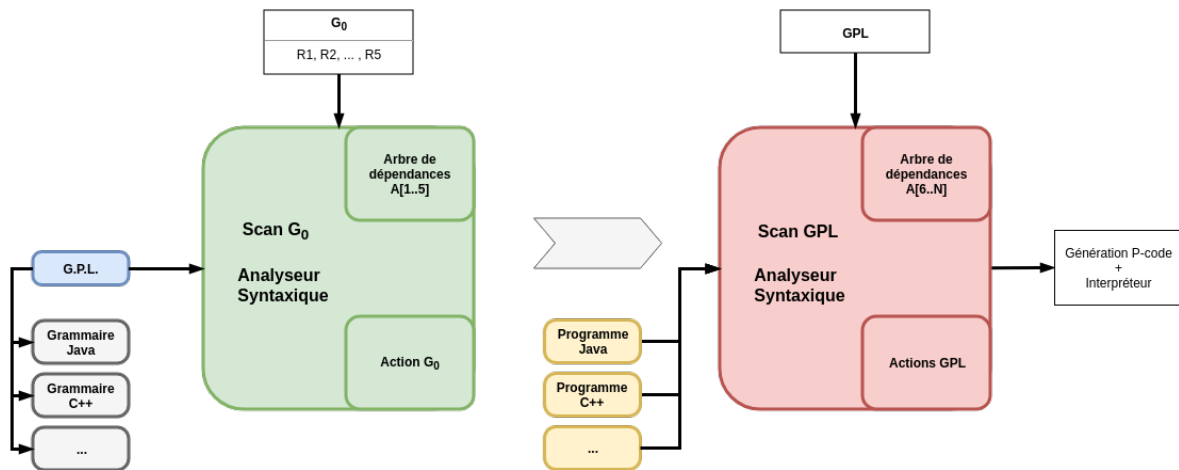


Figure 1: Projet Compilo

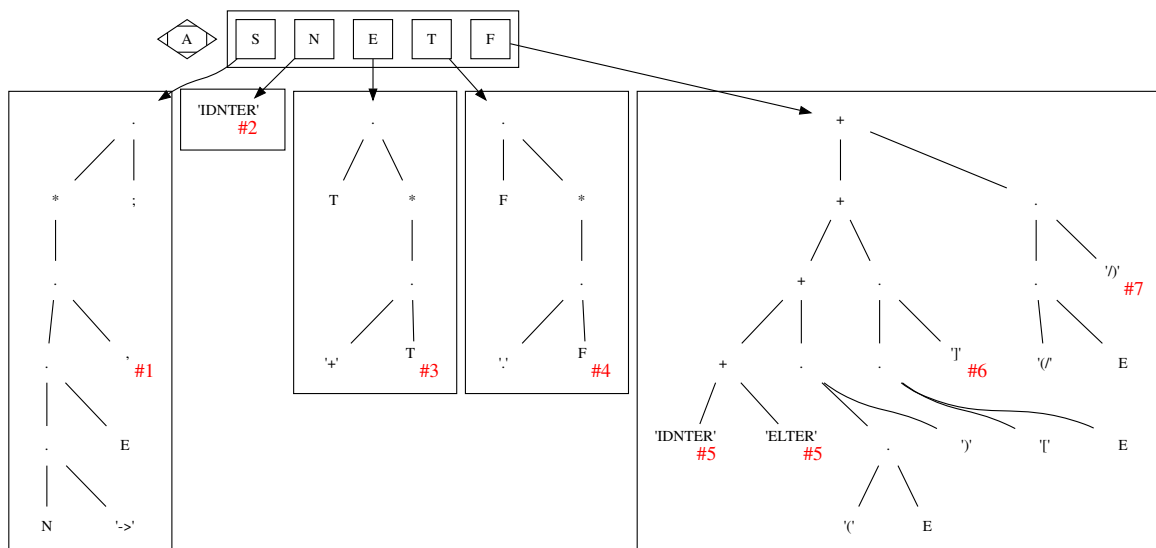


Figure 2: Arbres de dépendances  $G_0$

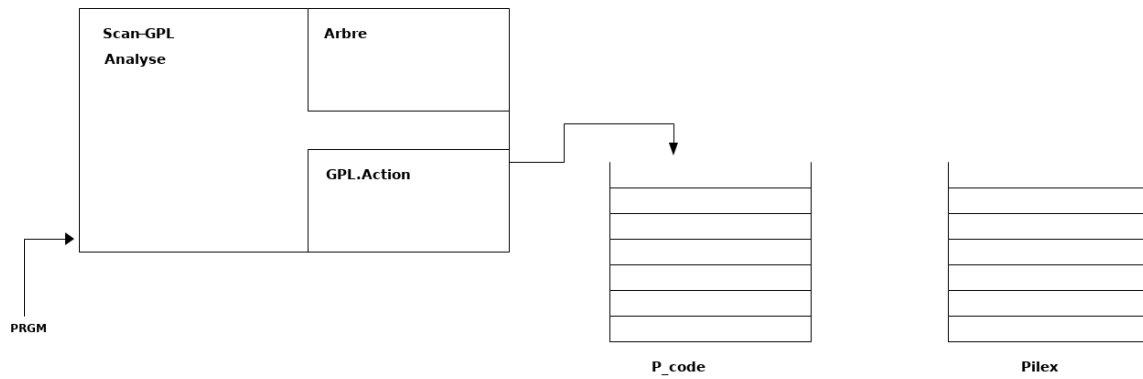


Figure 3: Actions de la grammaire GPL

### 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.

- $::= \iff \rightarrow$
- $[X] \iff X.X.X...X$  (n fois),  $n \geq 0$
- $(/X/)$   $\iff X$  ou Vide
- $/ \iff +$
- *concat*  $\iff \cdot$
- ' $X$ ' 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. ### Règle 3

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

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

### 1.4.4 Règle 4

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

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

### 1.4.5 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', \varnothing, NonTerminal),
            GenAtom('-', 5, Terminal)
          ),
          GenAtom('E', \varnothing, 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', \varnothing, NonTerminal),
  GenStar(
    GenConc(
      GenAtom('+', ?, Terminal),
      GenAtom('T', \varnothing, Terminal)
    )
  )
)
```

4. T

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

5. F

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

```

```

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

```

## 1.7 Scan $G_0$

Fonction analyse...

## 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];

```

Exemples à venir...

## 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 a chaque fois, mais plutot 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 ( $A \rightarrow a$ )

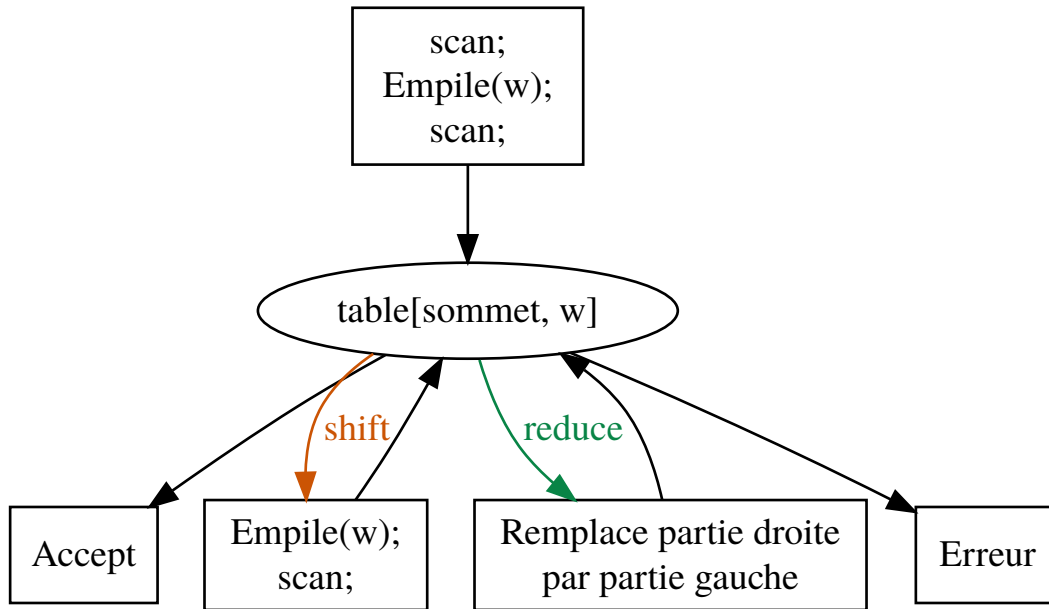


Figure 4: Algorithme Table Analyse L.R.

#### 3.2 Génération automatique de la table SR

##### 3.2.1 Opérateurs $\doteq$ , $\triangleright$ , et $\triangleleft$

- **Shift**
  - $X \doteq Y$  si

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

- $X \triangleleft Y$  si

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

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

- **Reduce**

–  $X \succ Y$  si

$$A \doteq Y$$

$$\text{et } A \xRightarrow{*} X$$

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

- (ligne  $\doteq$  colonne) et (ligne  $\prec$  colonne) se traduisent en (ligne Shift colonne)
- (ligne  $\succ$  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.