Contents

1 Projet compilo

1.1 Définitions

GPL Grammaire Petit Langage

Scanner analyse lexicale

Analyseur autres analyses (syntaxique et semantique)

1.2 Schémas

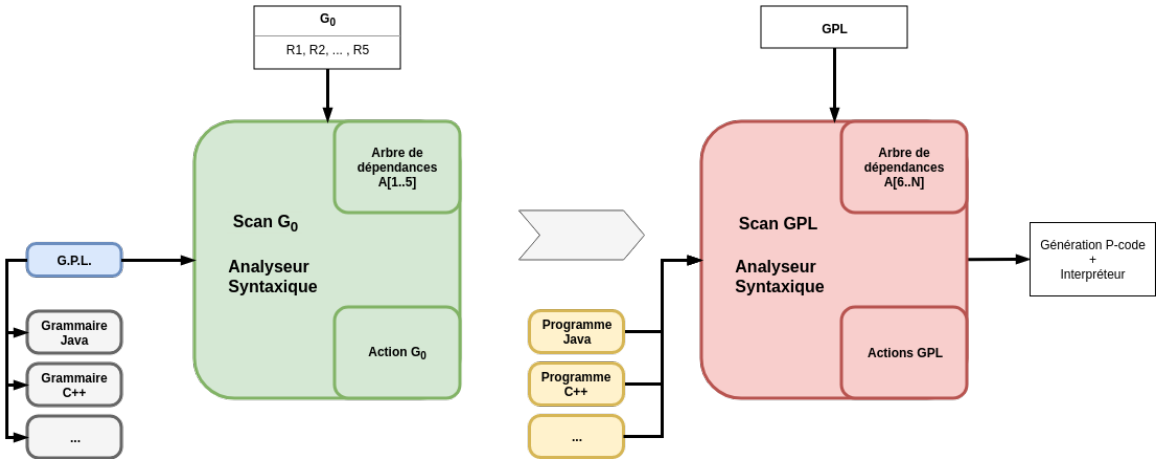
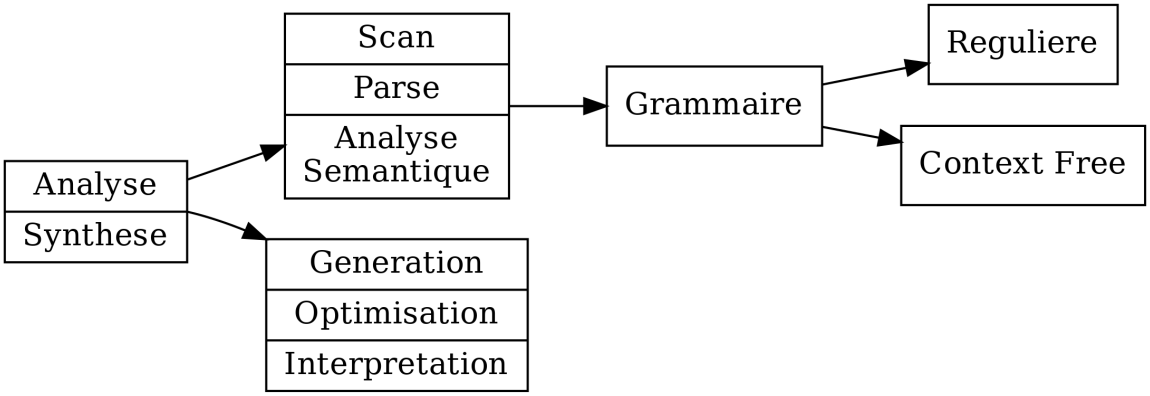
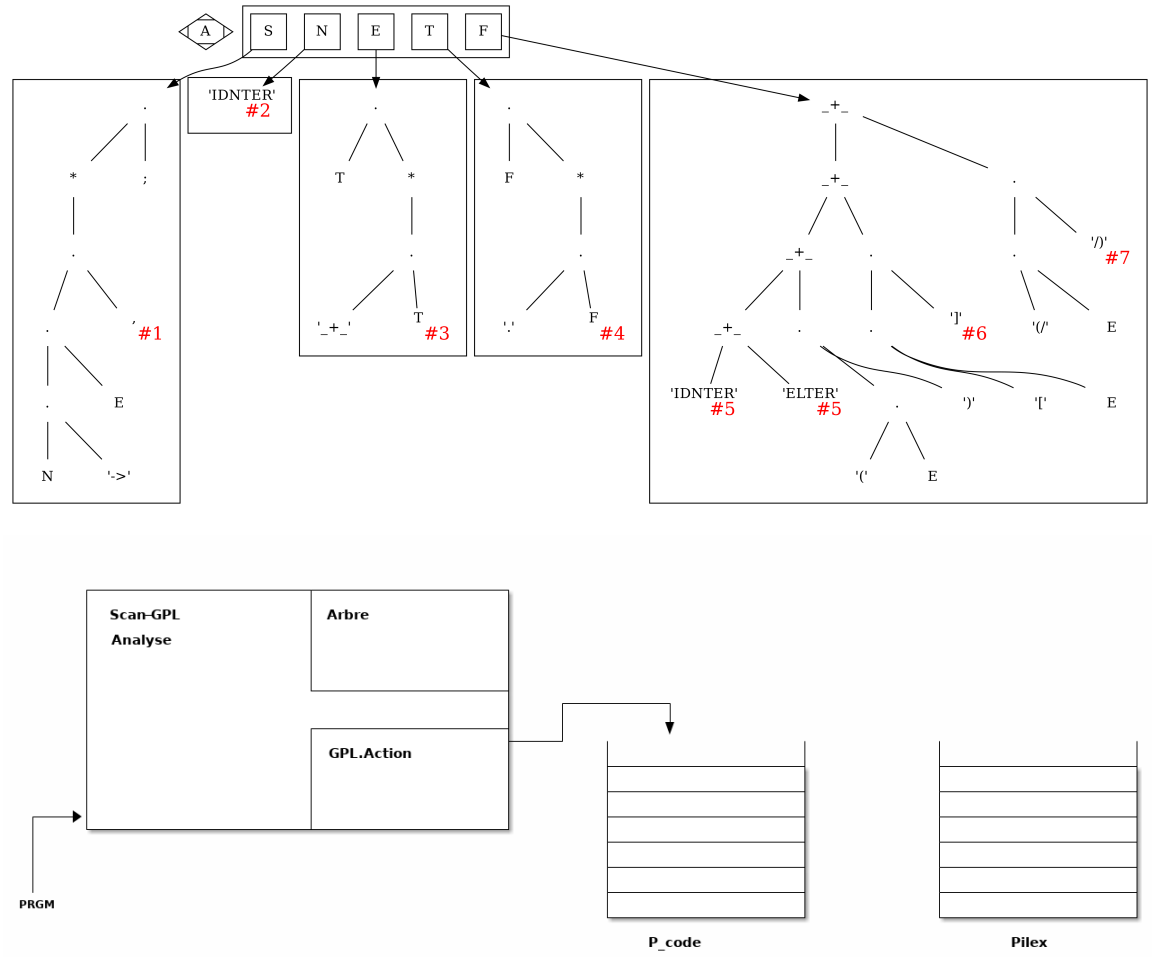


Figure 1.1: Projet Compilo





1.3 Processus divers

1.3.1 Scan G_0

Scanne les

- élements terminaux
- élements 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), $n \geq 0$
— $x \iff \cdot$

1.4.2 Règle 1

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

— concatenation $\iff \cdot$
— pour differencier les terminaux et les non terminaux, on met les terminaux entre guillemets

1.4.3 Règle 2

$$N \rightarrow' IDNTER',$$

1.4.4 Règle 3

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

1.4.5 Règle 4

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

1.4.6 Règle 5

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

1.5 Structure de données

Syntaxe maison...

```
Type Atomtype = (Terminal, Non-Terminal);
    Operation = (Conc, Union, Star, UN, Atom);
PTR = \uparrow{} 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 \uparrow{}.left := P1;
  P \uparrow{}.right := P2;
  P \uparrow{}.class := conc;
  GenConc := P;
fin

Fonction GenUnion(P1, P2 : PTR) : PTR;
  var P : PTR;
  d[U+FFFD]but
    New(P, union);
    P \uparrow{}.left := P1;
    P \uparrow{}.right := P2;
    P \uparrow{}.class := union;
    GenUnion := P;
  fin

Fonction GenStar(P1 : PTR) : PTR; //0 ou n fois
  var P:PTR;
  d[U+FFFD]but
    New(P, star);
    P \uparrow{}.stare := P1;
    P \uparrow{}.class := star;
    GenStar := P;
  fin

Fonction GenUn(P1 : PTR) : PTR; //0 ou une fois
  var P:PTR;
  d[U+FFFD]but
    New(P, un);
    P \uparrow{}.une := P1;
    P \uparrow{}.class := un;
    GenUn := P;
  fin

Fonction GenAtom(COD, Act : int, AType : Atomtype) : PTR
  var P:PTR;
  d[U+FFFD]but
    New(P, atom);
    P \uparrow{}.COD := COD;
    P \uparrow{}.Act := Act;
    P \uparrow{}.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 s[U+FFFD]r des r[U+FFFD]sultats :

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.

```
Proc[U+FFFD]dure Action G0(Act : int);
  var T1, T2 : PTR;
  d[U+FFFD]but
    case Act of
    1: D[U+FFFD]piler(T1);
       D[U+FFFD]piler(T2);
       A[T2[U+FFFD].cod + 5] := T1; ##Arbres GPL commencent [U+FFFD] 6
    2: Empiler(GenAtom(Recherche(DicoT), Action, CAType)) ##donne la partie gauche d'une r[U+FFFD]gle
      ##Recherche() stocke le token si non stock[U+FFFD] dans dico
    3: D[U+FFFD]piler(T1);
```

```
     D[U+FFFD]piler(T2);
     Empiler(GenUnion(T2,T1))
  4: D[U+FFFD]piler(T1);
     D[U+FFFD]piler(T2);
     Empiler(GenConc(T2,T1))
  5: if CAType = Terminal then
      Empiler(GenAtom(Recherche(DicoT), Action, Terminal))
     else
      Empiler(GenAtom(Recherche(DicoNT), Action, Terminal))
  6: D[U+FFFD]piler(T1);
     Empiler(GenStar(T1));
  7: D[U+FFFD]piler(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'ambiguité. 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 Premier(N)

— Si $N \to A \ldots$ alors $Premier(N) = Premier(A)$
— Si $N \to c \ldots$ alors $Premier(N) = \{c\}$
— Si $N \to A.B \ldots \wedge A \Rightarrow \epsilon$ alors $Premier(N) = Premier(B)$

Avec "⇒" signifiant "se derivant en".

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

2.2 Suivants

— Si $A \to \ldots Nc \ldots$ alors $Suiv(N) = \{c\}$
— Si $A \to \ldots NB \ldots$ alors $Suiv(N) = Prem(B)$
— Si $A \to N \ldots$ alors $Suiv(N) = Suiv(A)$

2.3 Grammaire LL(1)

— si $A \to \alpha_1/\alpha_2/\ldots/\alpha_n$ 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 possede qu'une derivation, la règle 1 ne s'applique pas. Si une règle ne possede pas de suiv, la règle 2 ne s'applique pas.

3 Génération automatique de la table SR

3.1 Opérateurs $\doteq$, $\gtrdot$, et $\lessdot$

— $X \doteq Y$ si

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

— $X \lessdot Y$ si

$$A \to \dots X.Q \dots \in \mathcal{P}$$
$$\text{et } Q \overset{*}{\Rightarrow} Y$$

— $X \gtrdot Y$ si

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

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

— (ligne $\doteq$ colonne) et (ligne $\lessdot$ colonne) se traduisent en (ligne Shift colonne)
— (ligne $\gtrdot$ colonne) se traduit en (ligne Reduce colonne)

4 Types des grammaires

0 type c

1 type context sensitive CS $\gamma \to \beta$ avec $\|\gamma\| \leq \|\beta\|$

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

3 type reguliere $\begin{cases} A \to aB \\ A \to a \end{cases}$ ou $\begin{cases} A \to Ba \\ A \to a \end{cases}$

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

l'intersection de deux languages de type x n'est pas forcement de type x.