Projet compilo

Définitions

GPL Grammaire Petit Langage
Scanner analyse lexicale
Analyseur autres analyses (syntaxique et semantique)

Schémas

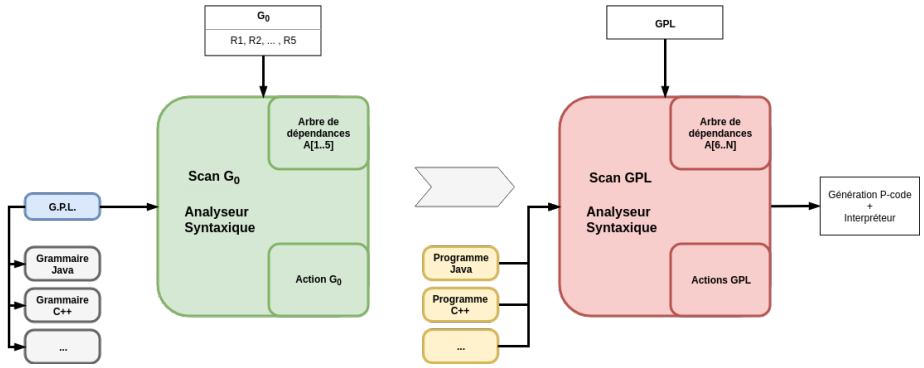


Figure 1: Projet Compilo

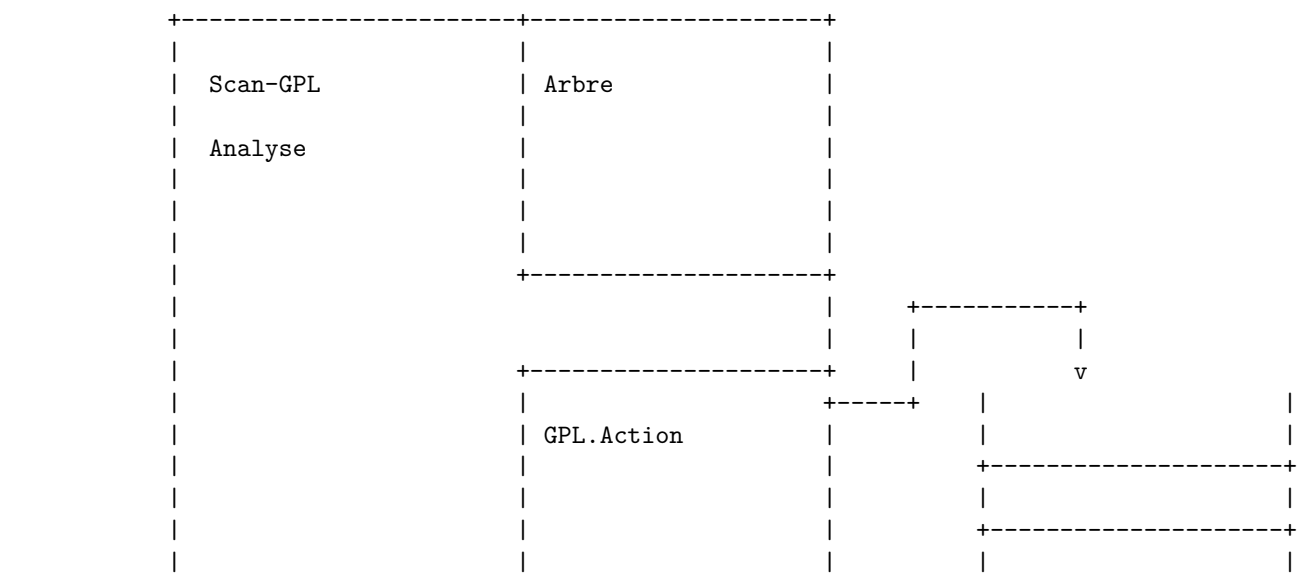
```
digraph {
   rankdir=LR
   node[shape=record]
   ansy [label="<a>Analyse|<s>Synthese"]
   an [label="<sc>Scan|<pa>Parse|<asem>Analyse\nSemantique"]
   gr [label="Grammaire"]
   re [label="Reguliere"]
   cf [label="Context Free"]
   sy [label="Generation|<op>Optimisation|Interpretation"]

   ansy:an -> an -> gr -> re,cf
   ansy:sy -> sy
}
```

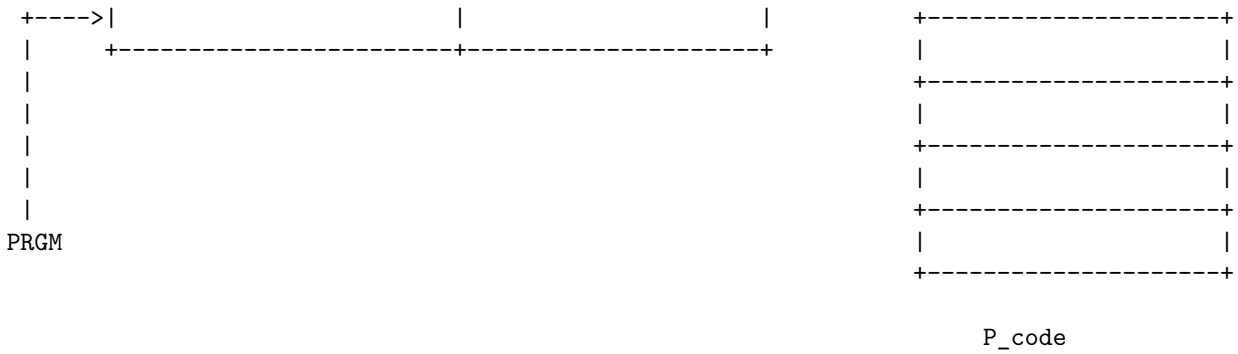
```
digraph G {
A [shape=Mdiamond];
subgraph clusterA {
S [shape=square];
N [shape=square];
E [shape=square];
```

```
T [shape=square];
F [shape=square];

}

subgraph clusterS {

    edge [dir=none]
    node [shape=none, style=none];
    ls1  [ label = "." ];

    ls21 [ label = "*" ];
    ls22 [ label = ";" ];
    ls31 [ label = "." ];
    ls41 [ label = "." ];
    ls42 [ label =< ,<BR />
        <FONT color="red" POINT-SIZE="18">#1</FONT>> ];
    ls51 [ label = "." ];
    ls52 [ label = "E" ];
    ls61 [ label = "N" ];
    ls62 [ label = "'->'" ];

    ls1  -> { ls21 ls22 };
    ls21 -> { ls31 };
    ls31 -> { ls41 ls42 };
    ls41 -> { ls51 ls52 };
    ls51 -> { ls61 ls62};
}

subgraph clusterN {
    node [shape=none, style=none];
    ln [ label=< 'IDNTER'<BR />
        <FONT color="red" POINT-SIZE="18">#2</FONT>>];
}

subgraph clusterE {
    edge [dir=none]
    node [shape=none, style=none];
    le1  [ label = "." ];
    le21 [ label = "T" ];
    le22 [ label = "*" ];
    le31 [ label = "." ];
    le41 [ label = "'_+_'" ];
    le42 [ label = < T <BR />
        <FONT color="red" POINT-SIZE="18">#3</FONT>>];
```

```
    le1  -> { le21 le22 };
    le22 -> { le31 };
    le31 -> { le41 le42 };
}

subgraph clusterT {
    edge [dir=none]
    node [shape=none, style=none];
    lt1  [ label = "." ];
    lt21 [ label = "F" ];
    lt22 [ label = "*" ];
    lt31 [ label = "." ];
    lt41 [ label = "'.'" ];
    lt42 [ label = < F <BR />
        <FONT color="red" POINT-SIZE="18">#4</FONT>>];

    lt1  -> { lt21 lt22 };
    lt22 -> { lt31 };
    lt31 -> { lt41 lt42 };
}

subgraph clusterf {
    edge [dir=none]
    node [shape=none, style=none];
    lf1  [ label = "_+_" ];
    lf21  [ label = "_+_" ];
    lf22  [ label = "." ];
    lf31  [ label = "_+_" ];
    lf32  [ label = "." ];
    lf33  [ label = "." ];
    lf34  [ label = < '/)' <BR />
        <FONT color="red" POINT-SIZE="18">#7</FONT>>];
    lf41  [ label = "_+_" ];
    lf42  [ label = "." ];
    lf44  [ label = "." ];
    lf43  [ label = < ']' <BR />
        <FONT color="red" POINT-SIZE="18">#6</FONT>>];
    lf45  [ label = "'(/'" ];
    lf51  [ label = < 'IDNTER' <BR />
        <FONT color="red" POINT-SIZE="18">#5</FONT>>];
    lf52  [ label = < 'ELTER' <BR />
        <FONT color="red" POINT-SIZE="18">#5</FONT>>];
    lf53  [ label = "." ];
    lf54  [ label = "')'" ];
    lf55  [ label = "'['" ];
    lf56  [ label = "E" ];
```

```
    lf61  [ label = "'(" ];
    lf62  [ label = "E" ];
    lf63  [ label = "E" ];

    lf1 -> { lf21 lf22 };
    lf21 -> {lf31 lf32};
    lf22 -> {lf33 lf34};
    lf31 -> {lf41 lf42};
    lf32 -> {lf43 lf44};
    lf33 -> {lf45 lf56};
    lf41 -> {lf51 lf52};
    lf42 -> {lf53 lf54};
    lf44 -> {lf62 lf55};
    lf53 -> {lf61 lf63};


}

    S -> ls1;
  N -> ln;
  E -> le1;
  T -> lt1;
  F -> lf1;

}
```





Processus divers

Scan G_0

Scanne les

- élements terminaux
- élements terminaux

Scan GPL

Scanne les

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

Action G_0

Construit l'arbre GPL

Construction de la grammaire G_0

Notation B.N.F.

- $::= \iff \rightarrow$
- $[X] \iff X.X.X...X$(n fois), $n \geq 0$
- $x \iff \cdot$

Règle 1

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

- concatenation $\Longleftrightarrow$ $\cdot$
- pour differencier les terminaux et les non terminaux, on met les terminaux entre guillemets

Règle 2

$$N \to' IDNTER',$$

Règle 3

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

Règle 4

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

Règle 5

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

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:
```

Construction des 5 Arbres

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;
  début
    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;
  début
    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;
  début
    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;
  début
```

```
  New(P, atom);
  P \uparrow{}.COD := COD;
  P \uparrow{}.Act := Act;
  P \uparrow{}.AType := AType;
  GenAtom := P;
fin
```

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)
            )
          )
```

Scan G_0

Fonction analyse...

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 règ
      ##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...

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.

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 "$\Rightarrow$" signifiant "se derivant en".

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

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 $Suiv(N) = Suiv(A)$

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.

Génération automatique de la table SR

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

- $X \doteq Y$ si

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

- $X \lessdot Y$ si

$$$$

REFAIRE EQUATION $$

- $X \gtrdot Y$ si

REFAIRE EQUATION

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)

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 reguliere

```
\begin{cases}
        A \rightarrow aB\\
        A \rightarrow a\\
\end{cases}

ou

\begin{cases}
        A \rightarrow Ba\\
        A \rightarrow 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.