Analyse syntaxique - Expl :

$$P = \begin{cases} S \longrightarrow E\$ \\ E \longrightarrow T\Pi \\ T \longrightarrow FN \\ \Pi \longrightarrow \varepsilon \mid +T\Pi \\ N \longrightarrow \varepsilon \mid *FN \\ F \longrightarrow (E) \mid a \end{cases}$$

On a déjà img G est LL(1)

On peut donc construire un analyseur syntaxique comme un réseau de procédures mutuellement récursives

On écrit 1 procédure par non-terminal
Le main() correspond à l'axiome ajouté (ici S)

```
main ( ) { // S → ES
    t = scanner( ); // récupère le prochain token
                    (Les symboles terminaux de G)

    analyse_E( );
    si (t == $) { succès }
    sinon { erreur }
}

analyse_E( ) { // E → TΠ
    analyse T( );
    analyse Π( );
}
```

```
analyse_T( ){ // T → FN
    analyseF( );
    analyseN( );
}

analyse_Π( ){ // Π → ε | +TΠ
    si ( t ∈ first(+TΠ)){ (ie t=='+')
        t = scanner( ) // on vient de trouver le + donc passage au suivant
        analyse_T( );
        analyse_Π( );
    sinon{
        si (t ∈ follows(Π)){ // on ne fait rien : passage à la suite ci-dessus de la récursion
        sinon { ERREUR }
}

analyse_N( ){ // N → ε | *FN
    si t ∈ first(*FN){
        t = scanner( );
        analyse_F( );
        analyse_N( );
    sinon
        si (t ∈ follow(N)){ //rien }
        sinon { ERREUR }
}
```

```
analyse_F(){ // F → (E) | a
    si ( t ∈ suivant((E)) ){
        t = scanner(); // trouvé (
        analyse_E()
        si ( t == ')' ){
            t = scanner(); // trouvé )
        }sinon{
            ERREUR
        }
    }sinon{
        si t ∈ suivant(a){
            t = scanner();
        }sinon{
            ERREUR
        }
    }
}
```

Si on veut substituer
{(→ [
}) →] ...

write([)

write(])

Rq : Si on a une règle $X \to Ya \mid b$

```
analyse_X(){
    si (t ∈ suivant(Ya)) { // pas de t = scanner() ici,
                           // car t n'est pas encore trouvé
                           // Mais on sait qu'il faut passer
                           // par cette règle.
        analyse_Y()
        si (t == 'a') {
            t = scanner() // t trouvé
        }sinon { ERREUR }
    }sinon{
        si (t ∈ suivant(b)){
            t = scanner() // on a trouvé b.
        }sinon{
            ERREUR
        }
    }
```

← c'est analyseY() qui remonte t