

**Projet TRE 2016/2017**

**Rapport**

*Abderrazak ZIDANE*

---

# Introduction

## Qu'est ce que l'algorithme Earley?

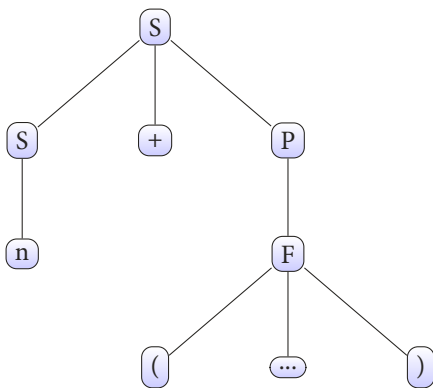
L'algorithme Earley, est un parmi les nombreux algorithmes d'analyse syntaxique, est comme tout ces algorithmes, Earley a besoin d'une grammaire

```
S = S + P
| P
P = P * F
| F
F = ( S )
| n
```

... pour transformer une chaîne de mot ...

n	+	(	n	*	n	+	n	)
---	---	---	---	---	---	---	---	---

... en un joli arbre syntaxique (AST)



Jusqu'à présent rien de spéciale comparé au autre algorithme qu'on conné.

## Pourquoi devrions-nous nous soucier?

Le plus grand avantage d'Earley, est sans doute son accessibilité. La plus part des autres algorithme offre une restriction sur le type de grammaires, utilisé une grammaire récursif gauche et en rentrera dans une boucle infini, utilisé un autre type et l'algorithme ne marchera plus. Biensur il y a des counternement qu'on peut faire, mais souvent sa complique d'avantage l'algorithme et rendra le travaille plus complexe.

Pour dire simple Earley marche avec tout.

D'une autre part, pour avoir cette généralité, nous devons sacrifié la vitesse. On ne pourra donc pas rivaliser avec des algorithme comme Flex/Bison en terme de rapidité brute. Ce n'ai si grave, puisque

- Earley a une complexité cubique  $O(3)$ , dans les pires des cas, ces meme cas qui ne pourons etre trété pas d'autre algorithme

- La plus part des grammaire simple aurons une complexité linéaire
- Même la pire grammaire non-ambiguë pourra être analysée en  $O(n)$

## Vocabulaire

Du point de vue de l'algorithme Earley, la grammaire est constituée de règles. Voici un exemple de règle

$$S = S + P$$

$S$  est un symbole non terminale, et tout ce qui ne commence pas avec une lettre majuscule est considéré comme étant un symbole terminal (le symbole  $+$  dans notre exemple).

Dans le jargon d'Earley, il y a la notion d'items, Voici un item :

$$A = B \cdot * C \quad (4)$$

C'est juste une règle de grammaire avec des informations en plus, qui représente une reconnaissance partielle.

- Le point représente la position courante qui indique jusqu'où on a parvenu à parser
- Le chiffre 4 représente la position initiale sur l'entrée qu'on veut parser

De plus Earley introduit la notion d'ensemble d'items, chaque ensemble est caractérisé par le fait que les items qui y sont associés, ont la même position courante

Et tous ces ensembles là, sont souvent nommés table Earley

## Prédiction, Lecture et Complétion

Pour construire la table Earley, on a besoin de définir trois opérations élémentaires qui s'appliquent sur un item pour produire un autre item :

- Prédiction : le symbole à droite du point est un nom terminal, on ajoute les règles de ce symbole au même ensemble
- Lecture : le symbole à droite du point est un terminal, on regarde si ce symbole coïncide avec la position courante, si oui, on ajoute cet item à l'ensemble suivant.
- Complétion : il n'y a rien à droite du point, et dans ce cas il y a reconnaissance partielle, on regarde l'item parent, et on l'ajoute à cet ensemble

## Exemple de construction de table d'Earley

Reprenons cette grammaire :

$$\begin{aligned} S &= S + P \\ P &= P * F \\ F &= ( S ) \\ n & \end{aligned}$$

on veut reconnaître l'entrée

n	+	(	n	*	n	)
---	---	---	---	---	---	---

À l'étape 0, le calcul démarre avec l'ensemble  $E(0)$  et les règles de l'axiome ' $S$ '

$E(0)$
$S = \bullet S + P \quad (0)$
$S = \bullet P \quad (0)$

la prédiction du premier item de  $E(0)$  nous donnera les mêmes 2 items de  $E(0)$ , et donc pas besoin de faire quoi que ce soit, donc une grammaire récursive gauche ne posera pas de problème à notre algorithme.

La prédiction du deuxième item de  $E(0)$  générera deux nouveaux items :

E(0)
S = •S + P (0)
S = •P (0)
P = •P * F (0)
P = •F (0)

Le prédiction du 3ème item de E(0) ne sert a rien. La prédiction du 4ème item de E(0) générera deux nouveaux items supplémentaire :

E(0)
S = •S + P (0)
S = •P (0)
P = •P * F (0)
P = •F (0)
F = •( S ) (0)
F = •n (0)

La Lecture du 5ème item de E(0) échoue puisque le symbole ne correspond pas a l'entrée.

La lecture du 6ème item se fait avec succès, est génère un nouveau item dans l'ensemble suivant E(1)

E(1)
F = n• (0)

On a traité tout les items de E(0), attaquons nous a l'ensemble E(1)

La Complétion du premier item de E(1), nous fait ajouter le 4ème item de E(0) a E(1) :

E(1)
F = n• (0)
P = F• (0)

la Complétion du deuxième item de E(1), nous fait ajouter le deuxième et troisième item de E(0) dans E(1)

E(1)
F = n• (0)
P = F• (0)
S = P• (0)
P = P• * F (0)

...

Au finale notre table Earley ressemblera a :

E(0)	E(1)	E(2)	E(3)	E(4)
S = •S + P (0) S = •P (0) P = •P * F (0) P = •F (0) F = •( S ) (0) F = •n (0)	F = n• (0) P = F• (0) S = P• (0) P = P• * F (0) S = S• + P (0)	S = S + •P (0) P = •P * F (2) P = •F (2) F = •( S ) (2) F = •n (2)	F = ( •S ) (2) S = •S + P (3) S = •P (3) P = •P * F (3) P = •F (3) F = •( S ) (3) F = •n (3)	F = n• (3) P = F• (3) S = P• (3) P = P• * F (3) S = S• + P (3) F = ( S• ) (2)
E(5)	E(6)	E(7)		
P = P * •F (3) F = •( S ) (5) F = •n (5)	F = n• (5) P = P * F• (3) S = P• (3) P = P• * F (3) F = ( S• ) (2) S = S• + P (3)	F = ( S )• (2) P = F• (2) S = S + P• (0)		

## Que va t'on faire?

Nous allons créer un programme qui va avoir en entrée une grammaire, et aura en sortie un analyseur syntaxique suivant l'algorithme Earley.

Par la suite en va modifier cet algorithme pour notre besoin (TO DO :)

---

# Développement de l'outils

Le programme sera écrit en C++, mais sera facilement traduit dans d'autre langage si nécessaire.

## Petite Pré-analyse avant de commencer

Notre outils aura en entrée une grammaire et en sortie, on aura un analyseur syntaxique. Plusieurs questions se sont posées durant le développement de cet outil. Voici un récapitulatif des décisions prises :

- Nous appellerons notre programme `earley`
- La première entrée de notre programme, sera un fichier, qui contiendra la description de la grammaire en format Yacc
- La deuxième entrée de notre programme sera un deuxième fichier, qui contiendra la chaîne à analyser.
- La sortie sera un troisième fichier contenant l'AST

Le programme sera exécuter en ligne de commande suivant la syntaxe suivante :

`earley <file1> <file2> <file3>`

`file1` : le fichier de grammaire `file2` : le fichier contenant la chaîne à analyser `file3` : le fichier contenant l'AST

## Explication du code