

Parser LL(k)

Voici la page wikipedia

- Le premier **L** est pour “left to right”.
- Le deuxième est pour “leftmost derivation”.
- Le **k** indique le nombre de fois que l’on va regarder en avance (le follow).

Exemple

- $Z \rightarrow YZ$
- $Z \rightarrow \text{“d”}$
- $Y \rightarrow \text{“c”}$
- $Y \rightarrow \epsilon$
- $X \rightarrow Y$
- $X \rightarrow \text{“a”}$

Ensemble NULL

L'ensemble *NULL* est l'ensemble des éléments non terminaux qui peuvent donner un mot vide.

$$NULL = \{Y, X\}$$

Ensemble First

$$X \rightarrow Y \rightarrow \text{First}(X) \supset \text{First}(Y)$$

$$X \rightarrow \text{“a”} \rightarrow \text{First}(X) \supset \{a\}$$

- $\text{First}(X) = \{a, c\}$
- $\text{First}(Y) = \{c, \}$
- $\text{First}(Z) = \{d, a, c\}$

Quand on calcule l'ensemble *First* il faut avoir un œil sur l'ensemble *NULL*.

Ensemble Follow

- $Z \rightarrow YZ$
 - $\rightarrow \text{Follow}(X) \supset \text{First}(YZ) = \text{FIRST}(Y) \cup \text{FIRST}(Z)$
 - $\rightarrow \text{Follow}(Y) \supset \text{First}(Z)$

- $Z \rightarrow "d" \rightarrow \emptyset$
- $Y \rightarrow "c" \rightarrow \emptyset$
- $Y \rightarrow \varepsilon \rightarrow \emptyset$
- $X \rightarrow Y \rightarrow \text{Follow}(Y) \supset \text{Follow}(X)$
- $X \rightarrow "a" \rightarrow \emptyset$

Donc cela nous donne:

- $\text{Follow}(X) = \{a, c, d\}$
- $\text{Follow}(Y) = \{a, c, d\}$
- $\text{Follow}(Z) = \emptyset$

Table du parser

1. $Z \rightarrow YZ$
2. $Z \rightarrow "d"$
3. $Y \rightarrow "c"$
4. $Y \rightarrow \varepsilon$
5. $X \rightarrow Y$
6. $X \rightarrow "a"$

	a	c	d
X	6/5	5/5	/5
Y	/4	3/4	/4
Z	1/	1/	12/

On va parse !

- $E \rightarrow E + T / F$
- $T \rightarrow R^* F \mid F$
- $F \rightarrow n$

```
1 void parse_E()
2 {
3     parse_T();
4     while (get_next_token() == TOK_PLUS)
5     {
6         eat(TOK_PLUS);
7         parse_+();
8     }
9 }
```

```
1 void parse_T()
2 {
3     parse_F();
4     while (get_next_token() == TOK_STAR)
5     {
6         eat(TOK_STAR);
7         parse_F();
8     }
9 }
```

```
1 void parse_F()
2 {
3     eat_number();
4 }
```

Creer un AST pendant le parsing

```
1  NODE parse_E()
2  {
3      NODE t = parse_T();
4      while (get_next_token() == TOK_PLUS)
5      {
6          eat(TOK_PLUS);
7          NODE tp = parse_+();
8          t = new_node(t, '+', tp);
9      }
10     return t;
11 }
```

```
1  NODE parse_T()
2  {
3      NODE f = parse_F();
4      while (get_next_token() == TOK_STAR)
5      {
6          eat(TOK_STAR);
7          NODE fp = parse_F();
8          f = new_node(f, '*', fp);
9      }
10     return f;
11 }
```

```
1  void parse_F()
2  {
3      eat_number();
4  }
```

Parse LR

- Le premier **L** est pour “left to right”.
- Le deuxième est pour “rightmost derivation”.

Le **LR** est plus expressif que le **LL**.

Le principe de ce parser est de **shift** (prendre des éléments) et de les stocker dans une pile. Lorsque la tête de pile reconnaît un élément produit par une règle alors on réduit la pile par l'élément qui produit la règle.