Parser LL(k)

Voici la page wikipedia

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

Exemple

- Z → YZ
- Z → “d”
- Y → “c”
- Y → ε
- X → Y
- X → “a”

Ensemble NULL

L’ensemble *NULL* est l’ensembles des elements non terminaux qui peuvent donner un mot vide.

$NULL = \{Y, X\}$

Ensemble First

X → Y → First(X) ⊃ First(Y)

X → “a” → First(X) ⊃ $\{a\}$

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

Quand on calcul l’ensemble *First* il faut avoir un oeil sur l’ensemble *NULL*.

Ensemble Follow

- Z → YZ
  - → Follow(X) ⊃ First(YZ) = FIRST(Y) ∪ FIRST(Z)
  - → Follow(Y) ⊃ First(Z)

- Z → “d” → ∅
- Y → “c” → ∅
- Y → ε → ∅
- X → Y → Follow(Y) ⊃ Follow(X)
- X → “a” → ∅

Donc cela nous donne:

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

Table du parser

1. Z → YZ
2. Z → “d”
3. Y → “c”
4. Y → ε
5. X → Y
6. X → “a”

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

On va parse !

- E → E + T/F
- T $rightarrow$ R * F | F
- F → n

```
void parse_E()
{
  parse_T();
  while (get_next_token() == TOK_PLUS)
  {
    eat(TOK_PLUS);
    parse_+();
  }
}
```

```
void parse_T()
{
  parse_F();
  wihle (get_next_token() == TOK_STAR)
  {
    eat(TOK_STAR);
    parse_F();
  }
}
```

```
void parse_F()
{
  eat_number();
}
```

Creer un AST pendant le parsing

```
NODE parse_E()
{
  NODE t = parse_T();
  while (get_next_token() == TOK_PLUS)
  {
    eat(TOK_PLUS);
    NODE tp = parse_+();
    t = new_node(t, '+', tp);
  }
  return t;
}
```

```
NODE parse_T()
{
  NODE f = parse_F();
  wihle (get_next_token() == TOK_STAR)
  {
    eat(TOK_STAR);
    NODE fp = parse_F();
    f = new_node(f, '*', fp);
  }
  return f;
}
```

```
void parse_F()
{
  eat_number();
}
```

Parse LR

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

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

Le principe de ce parser est de **shift** (prendre des elements) et de les stocker dans une pile. Lorsque la tete de pile reconnais un element produit par une regle alors on reduit la pile par l’element qui produit la regle.