skyguide

ENSG Géomatique

03.1 – Analyse syntaxique

Présenté par Yann Caron
skyguide

ENSG Géomatique

Plan du cours

LL(0)

LL(1)

LL(k)

Grammaires ambiguës

Définition

- Analyse LL
- Left to right, Left most derivation
- Analyse le mot d'entrée pour déterminer la production (Attention ! Limitaton !)
- L'arbre syntaxique est construit depuis la racine puis en descendant

LL1

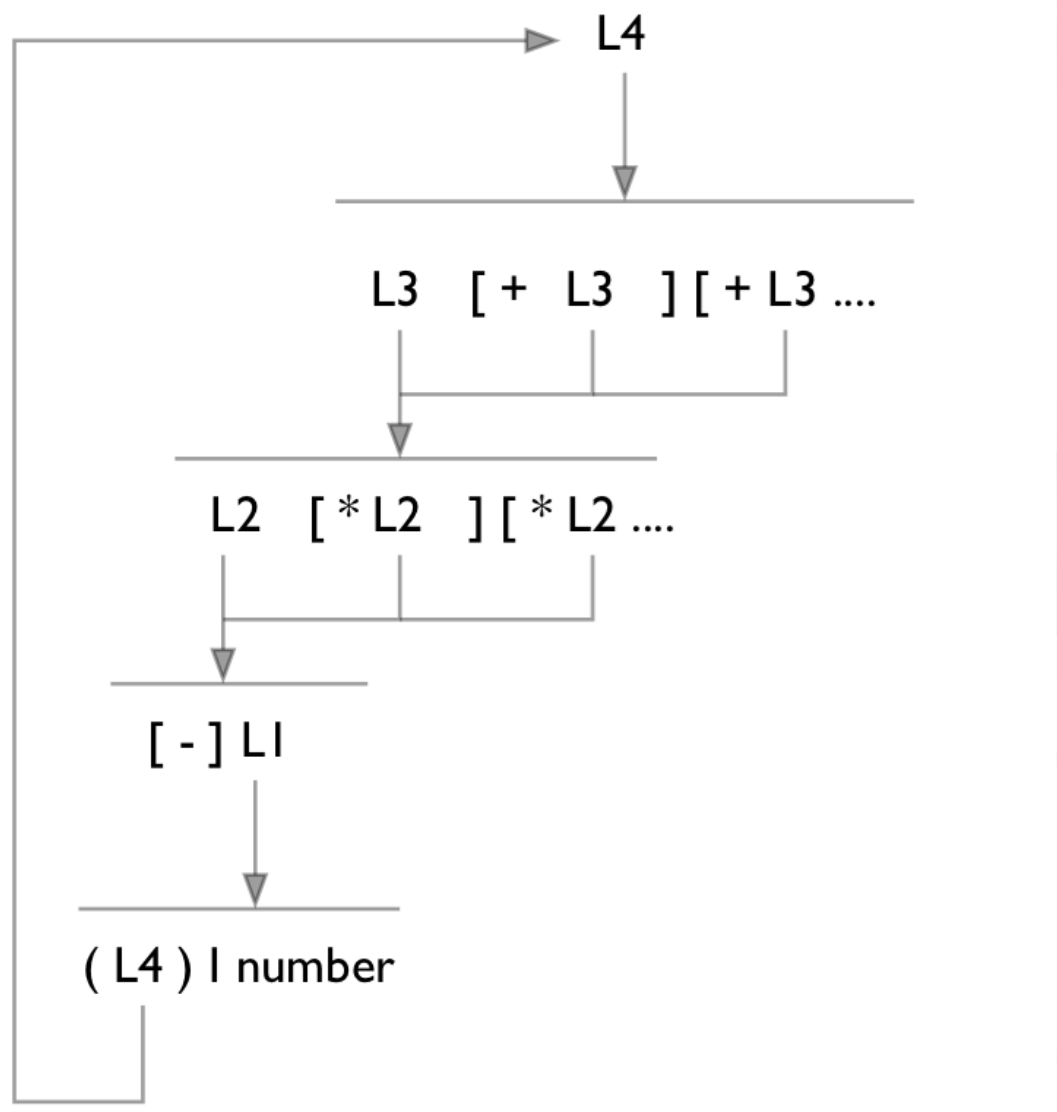
EBNF – Arithmétiques

- But : analyser une expression arithmétique
- Grammaire intuitive :

Observe la précédence
Des opérateurs

```
expr ::= expr ('+' | '-') expr |
         expr ('*' | '/') expr |
         '(' expr ')' |
         int
```

EBNF – Précédence



EBNF – Arithmétiques - LL1

```
expr  ::= factor expr';
expr' ::= '+' factor expr' |
          '-' factor expr' | ξ;

factor  ::= term factor';
factor' ::= '*' term factor' |
            '/' term factor' | ξ;

term ::= '(' expr ')'| int;
```

Boucle grammaticale

Optionnel

LL1 - Algorithme

Démo : recursive-descent-parser.al

E → F E'

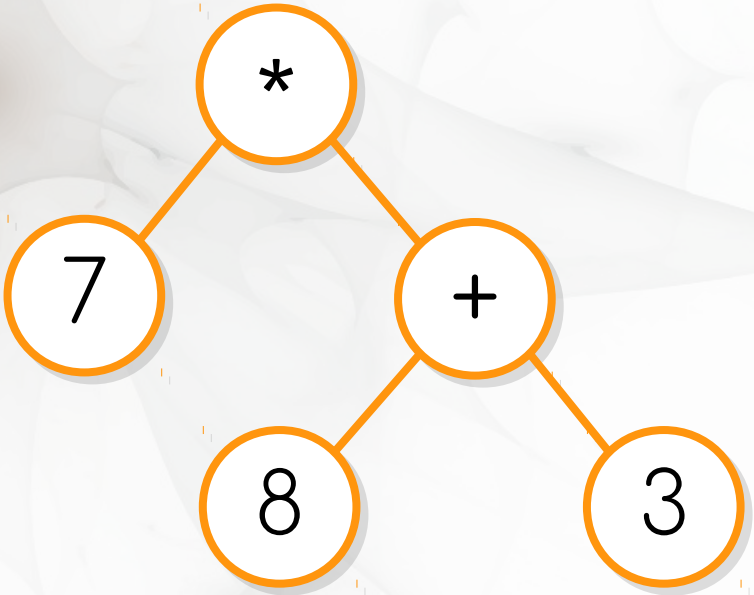
E' → '+' F E' | '-' F E' | ξ

F → T F'

F' → '*' T F' | '/' T F' | ξ

T → (E) | int

7 * (8 + 3)



Implémentation - Typique

- Top-down parser – mutuellement récursif
- Recursive Descent Parser
  - Tester qu'il reste bien un lexem
  - Tester le symbole
  - Appeler les fonctions si nécessaire

Limitations

- Pour produire, il faut un premier symbole fixe (déterminant)
- Convient pour le Basic, le Pascal ou l'analyse lexicale
  - Pourquoi les identifiants doivent commencer par une lettre (C like, Python, Pascal, Basic ...)
  - `String a87 = « myString »` est valide
  - `String 87A = « myString »` ne l'est pas !

Limitations - solution

- Pour augmenter le nombre de grammaire reconnu on peu utiliser un "look ahead"
- Avant la production on regarde les éléments suivant
- L'analyseur LLk
- k détermine le nombre d'éléments à regarder
- LL1 est un cas particulier du LLk
- LL(*) lorsque k est illimité (pré-analyse sans production)

LLk

Analyseur LLk

- Détermine la grammaire en analysant plusieurs lexèmes à l'avance
- k est un entier fini qui détermine le nombre de lexèmes maximum
- LL1 en est un cas particulier de degré 1
- Cas de JavaCC
- La grammaire est simplifiée

EBNF – Arithmétiques - LLk

- Le degré 2 suffit pour l'arithmétique
- Plus proche de la grammaire intuitive :

```
expr ::= factor [('+' | '-') expr]
factor ::= term [('*' | '/') factor]
term ::= '(' expr ')' | int
```

LLk - Algorithme

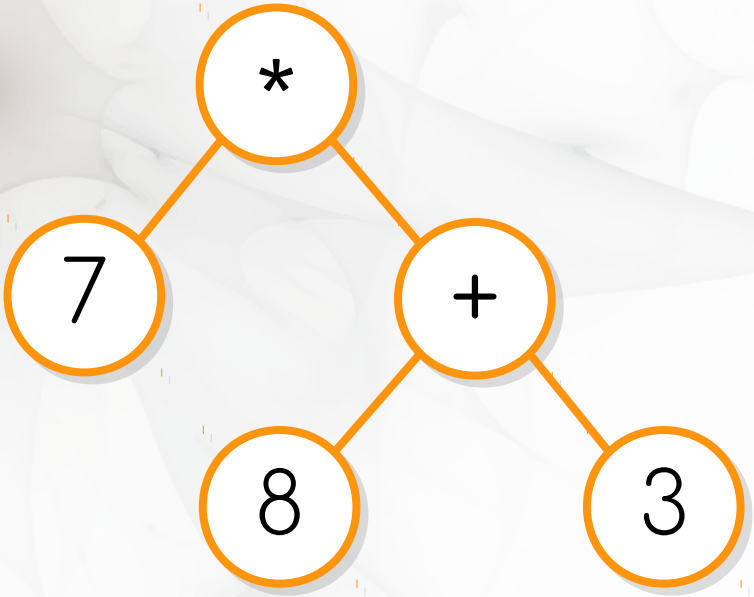
Démo : recursive-descent-parser2.al

E → F [(‘+’ | ‘-’) E]

F → T [(‘*’ | ‘/’) F]

T → (E) | int

7 * (8 + 3)



LL(*)

Analyseur LL(*)

- Détermine la grammaire en analysant tous les lexèmes à l'avance
- LLk en est un cas particulier de degré finit
- Principe : lorsqu'il faut déterminer un choix, une analyse prévisionnel totale est lancée
- Analyse sans production

EBNF – Arithmétiques - LL(*)

- Lookahead infini
- Plus proche de la grammaire intuitive :

```
expr ::= factor [('+' | '-') expr]
factor ::= term [('*' | '/') factor]
term ::= '(' expr ')' | int
```

Complexité

- La complexité de l'algorithme LL(*) est exponentielle : $O(c^n)$

- Pour réduite on peut utiliser un cache appelé Memorizer
- On mémorise les choix effectués durant la phase de prévision
- Un tableau associatif suffit

- Packrat

Grammaires ambiguës

Grammaires ambiguës

- Définition : une grammaire ambiguë c'est lorsque qu'une grammaire peut engendré deux arbres différents
- Exemple : if - if – else (à quel if correspond le else)
- Solution : le Parsing Expression Grammar
  - Dans le cas d'une grammaire ambiguë, le PEG va choisir la première qui correspond
  - En réalité proche du programme et non de la théorie

Merci de votre attention