Analyseur syntaxique

Rapport de projet VISI_201

Porteries Tristan

5 mai 2018

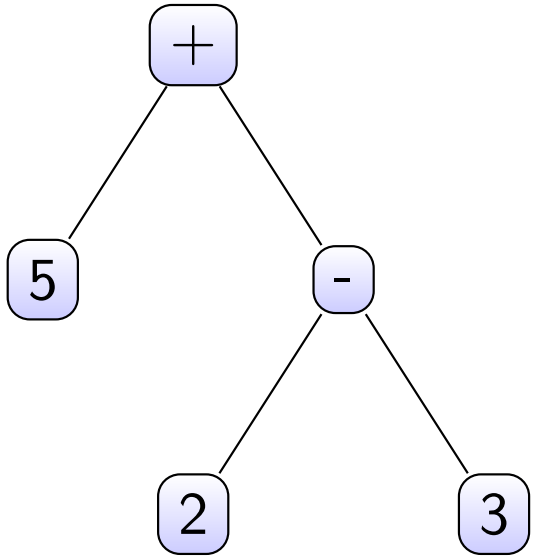
Sommaire

Introduction

Objectifs d'un analyseur syntaxique :

- déterminer l'existence d'une chaîne selon une grammaire ;
- reporter des erreurs ;
- construire un arbre abstrait de syntaxe.

5 + 2 - 3



Sommaire

$$G = (V, A, S, P)$$

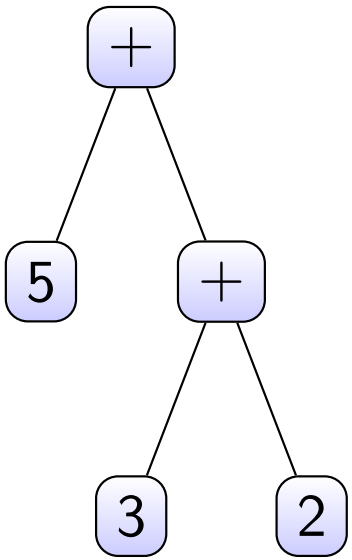
- V : ensemble des non-terminaux.
- A : ensemble des terminaux.
- S : non-terminal axiome, $S \in V$.
- P : ensemble des productions, $P \subset V \times (A \cup V)^*$.

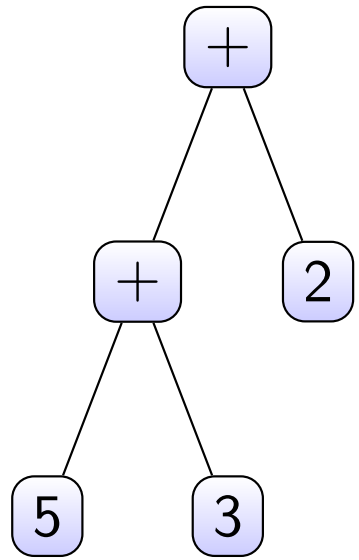
Les grammaires pouvant produire des arbres de dérivations différents pour la même chaîne sont ambiguës.

$$E \rightarrow E + E$$

$$E \rightarrow id$$

5 + 3 + 2





Toléré pour les opérateurs commutatifs.

$$S \rightarrow aSb$$

$$S \rightarrow \epsilon$$

Une dérivation d'un non-terminal selon une production mène à une proto-phrase ou phrase.

`aabb`

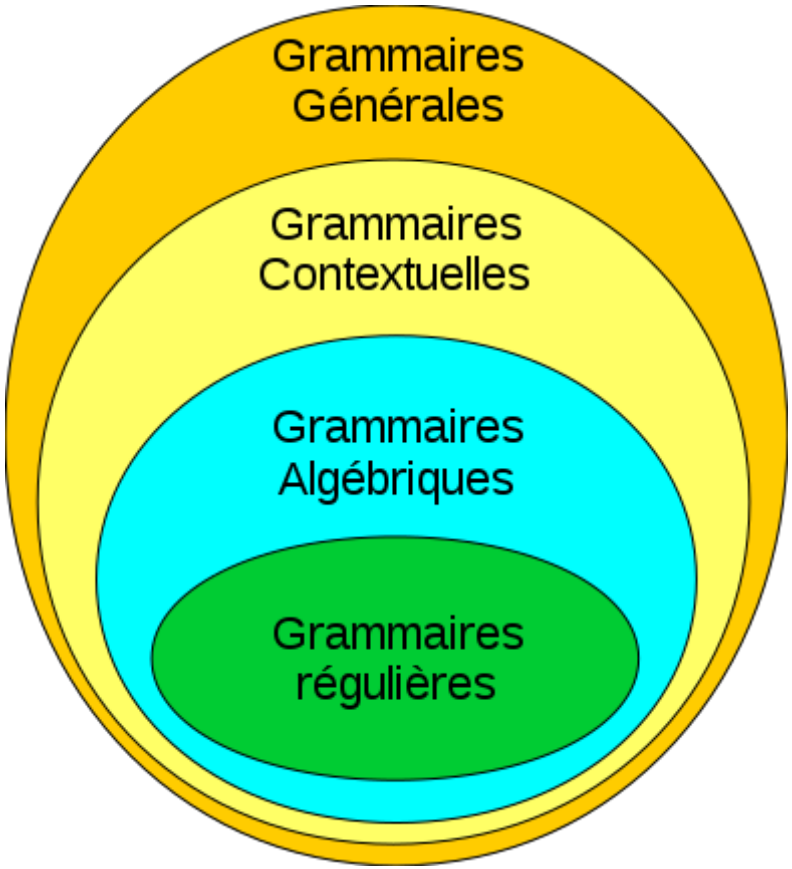
$$S \rightarrow aSb$$

$$S \rightarrow aaSbb$$

$$S \rightarrow aabb$$

$$\boxed{S \overset{*}{\rightarrow} aabb}$$

Sommaire



Grammaires contextuelles	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Grammaires algébriques	$A \rightarrow \gamma$
Grammaire régulières	$A \rightarrow aB, A \rightarrow a$

Sommaire

Analyse de la chaine d'entrée pour reconnaitre des lexèmes.
Utilisation d'expression pour décrire des grammaires linéaire.
Soit x et y appartenant au langage :

Opération	Notation	Exemple
Concaténation	xy	$\{ab\}$
Union	$x\|y$	$\{a, b\}$
Étoile Kleene	$(x\|y)*$	$\{\epsilon, a, b, ab, ba, aa, bb, ...\}$

Limitation : $a^n b^n$

Théorème de Kleene : l'ensemble des langages rationnels sur un alphabet A est exactement l'ensemble des langages sur A reconnaissables par automate fini.

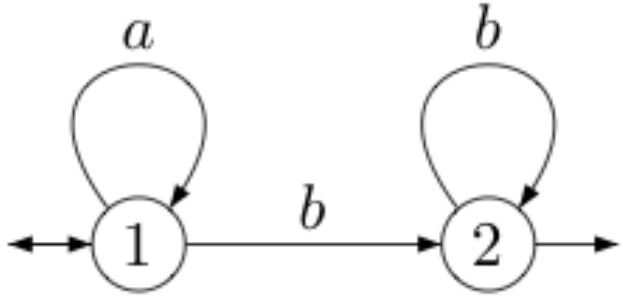


FIGURE – Automate reconnaissant le langage a^*b^*

Sommaire

Description de règles d'analyse (productions) :

$$A \rightarrow aAb$$

```
<A> ::= a <A> b
```

Dérivation de l'axiome à partir de la gauche, si les dérivations mènent à la phrase d'entrée : accepter la phrase.
Si $S \xrightarrow{*} \omega$ accepter ω

$$S \rightarrow aAb$$
$$S \rightarrow \epsilon$$

$\omega = aabb$

1. S
2. aAb
3. $aaAbb$
4. $aabb$

Contrainte de récursivité gauche des productions.

$$A \rightarrow A\alpha$$

$$A \rightarrow \beta$$

Suppression de la récursivité gauche :

$$A \rightarrow \beta A'$$

$$A' \rightarrow \alpha A'$$

$$A' \rightarrow \epsilon$$

Pile	Entrée	Opération
S	aabb	
aSb	aabb	dériver $S \rightarrow aSb$
Sb	abb	valider a
$aSbb$	abb	dériver $S \rightarrow aSb$
Sbb	bb	valider a
bb	bb	dériver $S \rightarrow \epsilon$
b	b	valider b
		valider b

Comment choisir la production à dériver ?
Utilisation d'un symbole de prévision : analyseur LL(1).
Construction d'une table associant non-terminal, lexème à une production.
Limitation : seul les grammaires où toutes les productions d'un non-terminal n'ont pas les mêmes terminaux préfixes.

$$A \rightarrow a\gamma$$

$$A \rightarrow b\gamma$$

Lire l'entrée depuis la gauche, reconnaître des production et réduire (décalage / réduction).

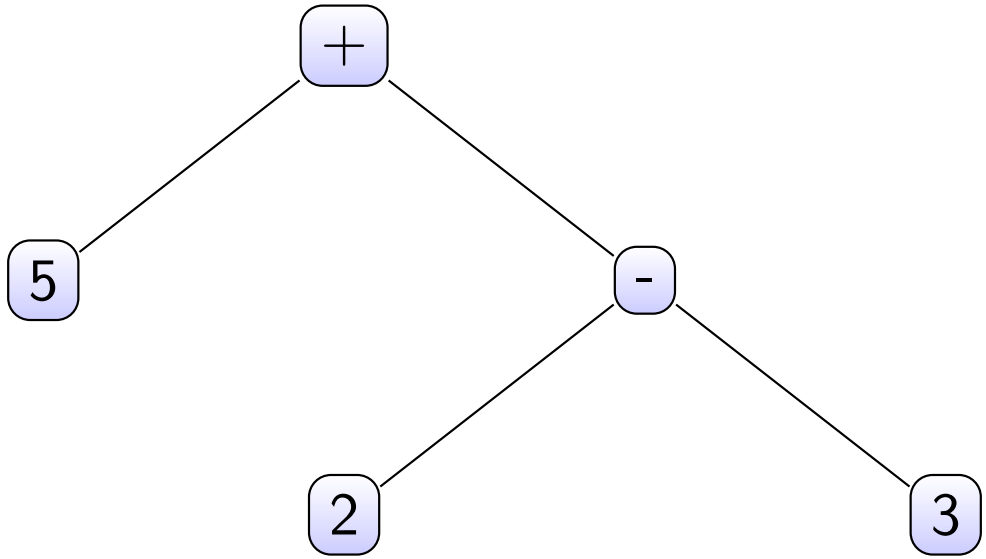
Pile	Entrée	Opération
	$aabb$	
a	abb	décaler
aa	bb	décaler
aab	b	décaler
aA	b	réduire $A \rightarrow ab$
aAb		décaler
A		réduire $A \rightarrow aAb$

Avantage : pas de récursivité.

$$A \rightarrow ab$$

$$A \rightarrow aAb$$

Sommaire

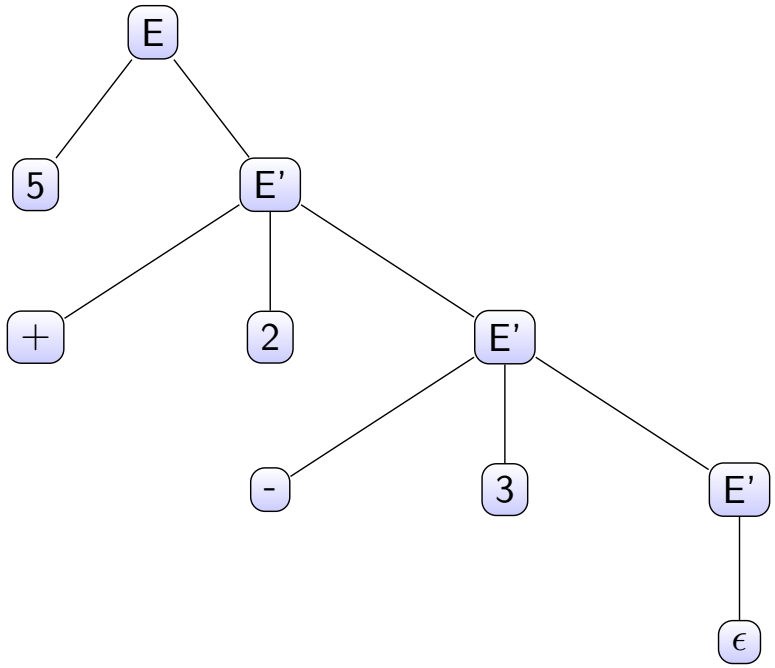


Les feuilles sont des opérandes, les autres nœuds des opérateurs.

$$E \rightarrow id\ E'$$

$$E' \rightarrow op\ id\ E'$$

$$E' \rightarrow \epsilon$$



Arbre des dérivations successives, les feuilles sont des terminaux, les autres nœuds des non-terminaux, la racine est l'axiome.

Associer à chaque production des règles pour construire les nœuds de l'arbre abstrait en fonction des nœuds de l'arbre de dérivation.

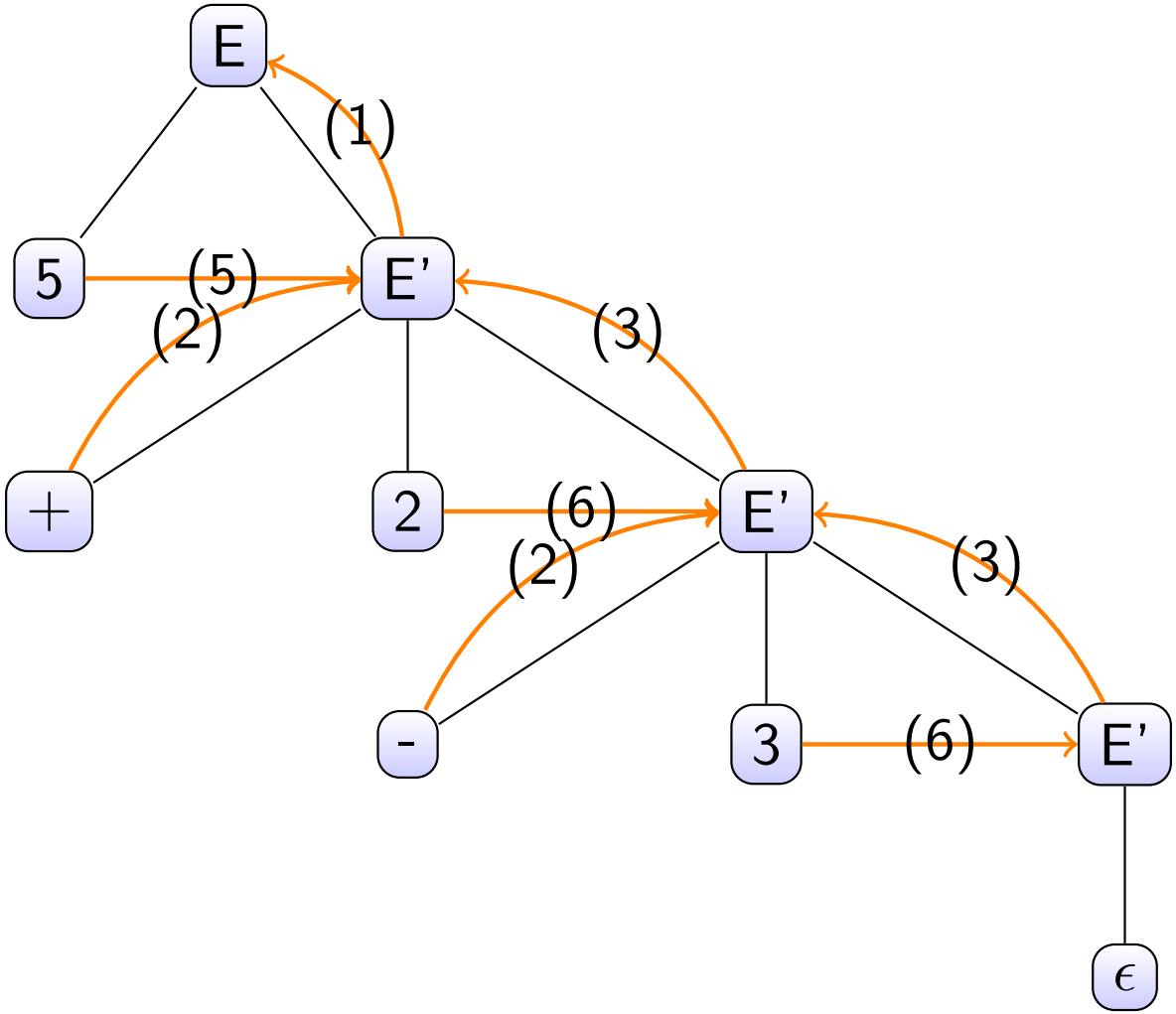
Attributs dépendant des nœuds fils d'un nœud de dérivation ou de lui même.

$$E \rightarrow id\ E' \Longrightarrow E.noeud = E'.noeud; \quad (1)$$
$$E'_1 \rightarrow id\ op\ E'_2 \Longrightarrow E'_1.noeud = noeud(op.val); \quad (2)$$
$$E'_1.noeud.ajouter(E'_2.noeud); \quad (3)$$
$$E' \rightarrow \epsilon \Longrightarrow E'.noeud = noeud(); \quad (4)$$

Attributs dépendant des nœuds parents ou des nœuds voisins.

$$E \rightarrow id\ E' \Longrightarrow E'.noeud.ajouter(noeud(id.val)) \quad (5)$$

$$E'_1 \rightarrow id\ op\ E'_2 \Longrightarrow E'_2.noeud.ajouter(noeud(id.val)) \quad (6)$$


E
(1)
5
(5)
E'
(2)
(3)
+
2
(6)
E'
(2)
(3)
-
3
(6)
E'
ϵ

Sommaire

Dépot git : `https://github.com/panzergame/analyseur_cmi`

`analyzer input_file bnf_file regex_file method`

- `input_file` : Fichier texte à analyser.
- `bnf_file` : Fichier de description des productions.
- `regex_file` : Fichier de description des règles lexicales et des séparateurs.
- `method` : La méthode d'analyse, `naive` pour LL récursif, `stack` pour LL avec pile et `slr` pour SLR.
- Résultat : un historique de l'analyse en console et un arbre de dérivation au format `.dot` : `derivation_tree.dot`.