Langages, grammaires

Fabienne Carrier, Laurent Mounier, Catherine Parent

Université Grenoble Alpes

1er octobre 2019

Plan

Introduction

Solution à un problème informatique exprimée par un "algorithme"

Pour être exécuté, un algorithme doit être exprimé dans un **langage de programmation**, puis traduit en un **programme (exécutable)**

Questions :

- Quels types de problèmes sont exprimables ? Limites ? $\longrightarrow$ calculabilité
- Caractéristiques d'un langage ?
  Puissance/confort d'expression vs complexité des algorithmes de reconnaissance/traduction
- Comment vérifie-t-on que le langage est utilisé correctement ?
  $\longrightarrow$ syntaxe, sémantique
- Comment passe-t-on d'un langage à un autre ? $\longrightarrow$ traduction

Ici, nous allons nous intéresser

- à la description des langages
- aux outils et formalismes pour décrire et reconnaître un langage

Définition d'un langage

Alphabet V

"phrases" sur V : chaîne de longueur finie (éventuellement vide) d'éléments de V

Exemple : V = $\{a, e, i\}$ phrases = $\{a.e, a.e.i, i.e, i, \ldots, \varepsilon\}$

Ensemble de toutes les phrases noté : V^*

Langage : sous-ensemble de V^*

Cas particulier : aucune phrase = langage vide noté $\emptyset$

Comment représenter un langage ?

- Enumérer les phrases
  représentation en extension d'un ensemble
- Donner une méthode pour produire/générer les phrases
  définition d'une grammaire
- Donner une procédure qui réponde "oui" si un mot appartient au langage et "non" sinon
  reconnaisseur, analyseur

En pratique : définition formelle d'un langage

$\neq$ **niveaux de description :**

- l'alphabet (ou vocabulaire) V
- l'ensemble L des lexèmes

  1 lexème = une séquence (finie) sur V ("mots" du langage)
- l'ensemble P des phrases/programmes

  1 phrase = une séquence (finie) sur L

P est ensuite restreint en un ensemble P' des phrases "qui ont un sens"
(sémantique statique/dynamique du langage)

En général :

- L peut être décrit par un langage **régulier**
- P peut être décrit par un langage **hors-contexte**
- P' peut être décrit par un système de preuves

Définition d'un langage régulier

Soit V un **alphabet**
L'ensemble des **langages réguliers** sur V peut-être défini récursivement par les règles suivantes :

- le langage vide $\emptyset$ est un langage régulier
- le langage $\{\varepsilon\}$ est un langage régulier
- pour tout x de V, $\{x\}$ est un langage régulier
- si $R1$ et $R2$ sont des langages réguliers alors : $R1 \cup R2$, $R1.R2$ et $R1^*$ sont des langages réguliers.

Langages réguliers : exemples

Les langages suivants sont réguliers sur $V = \{x, y, z\}$

- $\emptyset$
- $\{\varepsilon\}$
- $\{x\}$
- $\{y\}$
- $\{z\}$
- $\{x\} \cup \{y\}$
- $\{x.y\} \cup \{z\}$
- $\{x.y\}^* \cup \{z\} = \{\varepsilon, x.y, x.y.x.y, \ldots, z\}$

Le langage $\{a^n.b^n \mid n > 0\}$ défini sur $\{a, b\}$ n'est pas régulier

$\Rightarrow$ La plupart des langages de programmation sont non réguliers ...

Expressions régulières

Un langage régulier peut être décrit par une **expression régulière**
Une expression régulière est un terme construit sur l'alphabet $V \cup \{+, ., *, \underline{\varepsilon}, \underline{\emptyset}\}$

- $\underline{\emptyset}$ décrit le langage vide $\emptyset$
- $\underline{\varepsilon}$ décrit le langage $\{\varepsilon\}$
- pour tout x appartenant V, x décrit le langage $\{x\}$
- si $r1$ et $r2$ sont des expressions régulières sur V décrivant respectivement les langages $R1$ et $R2$ alors :
  - $r1 + r2$ décrit le langage $R1 \cup R2$
  - $r1.r2$ décrit le langage $R1.R2$
  - $r1^*$ décrit le langage $R1^*$

Expressions régulières : exemples

Priorités croissantes : $*$ puis $.$ puis $+$
Souvent on omet l'opérateur $.$

Expression régulière	Eléments du langage
$a+b.c$	$\{a, bc\}$
$(ab)^*$	$\{\varepsilon, ab, abab, ababab, \ldots\}$
$ab+c*$	$\{\varepsilon, ab, c, cc, ccc, \ldots\}$
$xy^*y+zx+\varepsilon$	$\{\varepsilon, zx, xy, xyy, xyyy, xyyyy, \ldots\}$
aa^*bb^*	$\{ab, aaaaaaab, abbbbb, aaabbbbb, \ldots\}$

aa^*bb^* : des a suivis de b est un langage régulier
$a^nb^n, n > 0$: des a suivis de b mais en même nombre n'est pas un langage régulier

Problème de la reconnaissance pour un langage régulier

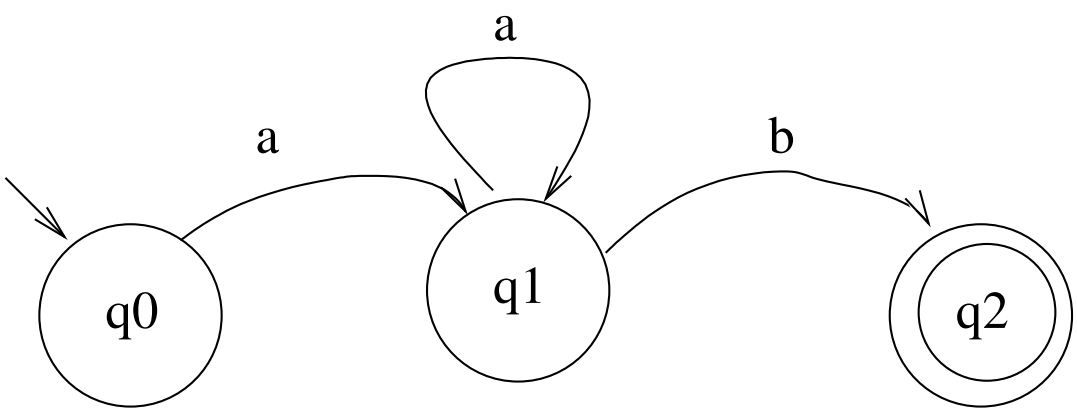
Soit R un langage régulier sur un vocabulaire V

Problème : reconnaître si un mot de V^* appartient à R

Algorithme efficace reposant sur la caractérisation de R à l'aide d'un automate d'état fini déterministe

Automates d'état fini : exemple

$V = \{a, b\}$



Automate reconnaissant le langage aa^*b

Etats : $q0, q1, q2$ Etat initial : $q0$ Etat final, accepteur : $q2$
Transition : flèches, passage d'un état à un autre

Automates d'état fini : définition

Quintuplet $A = (Q, V, \delta, q0, F)$

- Q est un ensemble fini d'états
- V est un ensemble fini de symboles (le vocabulaire d'entrée)
- $\delta \subseteq Q \times V \cup \{\varepsilon\} \times Q$ est la relation de transition
- $q_0 \in Q$ est l'état initial
- $F \subseteq Q$ est l'ensemble des états terminaux

Exemple :
$Q = \{q0, q1, q2\}, F = \{q2\}, \delta = \{(q0, a, q1), (q1, a, q1), q1, b, q2)\}$

Automates d'état fini : langage accepté

Soit $w = x_0.x_1.x_2.\ldots.x_n \in V^*$

w est accepté (reconnu) par l'automate A ssi
il existe $q_0.q_1.q_2.\ldots.q_n.q_{n+1} \in Q^*$ une séquence d'états
telle que

- $\forall 0 \leq i \leq n,\ (q_i, x_i, q_{i+1}) \in \delta$
- $q_{n+1} \in F$

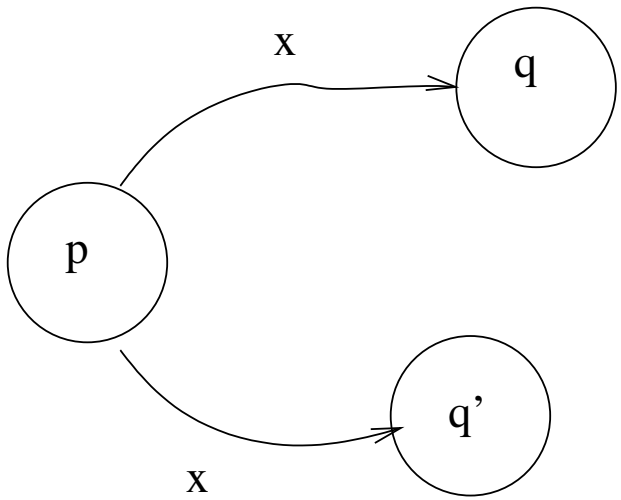
Automates d'état fini : déterminisme

Un automate d'état fini A est dit **déterministe** si et seulement si la relation de transition est une fonction :

$$\forall p, q, q' \in Q \ \forall x \in V$$

$$\text{si } (p, x, q) \in \delta \wedge (p, x, q') \in \delta \text{ alors } q = q'$$

Problème de choix ?



Automates d'état fini : propriétés

Pour tout automate d'état fini reconnaissant un langage L, on peut construire un automate d'état fini déterminisme reconnaissant L

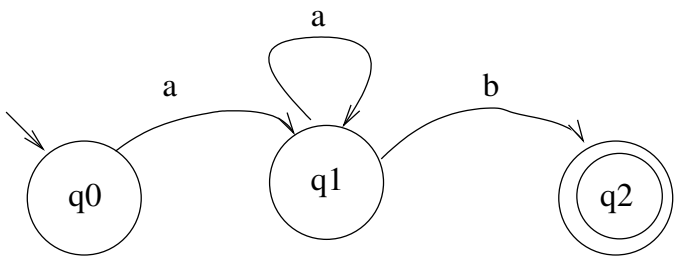
A partir de toute expression régulière, on peut construire un automate d'état fini déterminisme acceptant le même langage

Langages réguliers : description

Expression régulière : $a\ a^*\ b$

Grammaire :
$V_T = \{a, b\}\ \ V_N = \{S, X\}$
$P = \{S \to a\,X,\ X \to b,\ X \to a\,X\}$

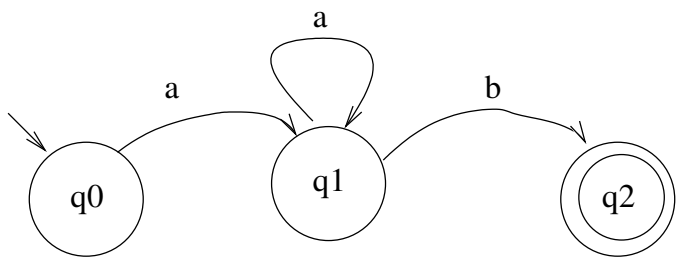
Automate d'état fini :



Construire une dérivation consiste à parcourir un chemin jusqu'à l'état final dans l'automate : $q0 \to a\,q1 \to a\,a\,q1 \to a\,a\,b$

Automate d'état fini : programmation

```
etat: entier; symb: caractere
lire(c) : lecture du caractere suivant dans c
```




```
etat <- 0; lire (symb)
tantque (symb != fin) et (etat != 2)
    selon etat
    etat=0: si symb='a' alors etat<-1 sinon erreur
    etat=1: si  symb='a' alors etat<-1
            sinon si symb='b' alors etat<-2
            sinon erreur
    lire (symb)
```

Analyse lexicale

Reconnaissance d'un ensemble de mots

- debut
- fin
- $=$
- $==$
- lettre : $[''A'' - ''Z'']|[''a'' - ''z'']$
- identificateur : lettre $([''0'' - ''9'']| < lettre >)^*$
- constante entière : $[''1'' - ''9''][[''0'' - ''9'']]^*$

Comment ajouter 0 à la définition des constantes entières ?

Grammaire

Outil permettant de décrire un langage : génération de phrases

Peut aussi servir à produire un reconnaisseur "efficace" (si bonnes propriétés)

Formellement $G = (V_T, V_N, S, P)$

V_T : vocabulaire terminal (alphabet)
V_N : vocabulaire non terminal
$S \in V_N$: axiome
$P \subset (V_T \cup V_N)^+ \times (V_T \cup V_N)^*$: règles de production

Un élément de P est noté : $\alpha \to \beta, \alpha \in (V_T \cup V_N)^+, \beta \in (V_T \cup V_N)^*$

Grammaire : exemple

$$G1 = (V_T, V_N, S, P)$$

$$V_T = \{a, b\}$$

$$V_N = \{A, B, S\}$$

$$P = \{S \to A\,B, A \to a, A \to a\,A, B \to \varepsilon, B \to b\,B\}$$

Dérivation (1/2)

Pour générer des mots nous utilisons la notion de dérivation

Soient $\alpha \to \beta \in P$, $\delta, \gamma \in (V_T \cup V_N)^*$

On note $\delta\alpha\gamma \Rightarrow_G \delta\beta\gamma$ la dérivation de $\delta\beta\gamma$ à partir de $\delta\alpha\gamma$ pour la grammaire G

$\delta\beta\gamma$ est dite chaîne dérivée en 1 pas (ou directement) de $\delta\alpha\gamma$

Dérivation (2/2)

Soient $\gamma_1, \gamma_2, \ldots, \gamma_n \in (V_T \cup V_N)^*$

avec $\gamma_1 \Rightarrow_G \gamma_2 \Rightarrow_G \Rightarrow_G \ldots \Rightarrow_G \gamma_n$

On écrit $\gamma_1 \Rightarrow_G^* \gamma_n$

γ_n est dérivée à partir de γ_1 par application d'un certain nombre de dérivations directes

Langage défini par une grammaire

$L(G) = \{w \in V_T^* \text{ tels que } S \Rightarrow w\}$

ensemble des chaînes formées de symboles de V_T qui sont dérivables à partir de l'axiome

exemple : $P = \{S \to A\,B, A \to a, A \to a\,A, B \to \varepsilon, B \to b\,B\}$

$S \Rightarrow_{G1} A\,B \Rightarrow_{G1} a\,B \Rightarrow_{G1} a$
donc $a \in L(G1)$

$S \Rightarrow_{G1} A\,B \Rightarrow_{G1} a\,A\,B \Rightarrow_{G1} a\,A\,b\,B \Rightarrow_{G1} a\,a\,b\,B \Rightarrow_{G1} a\,a\,b$
donc $aab \in L(G1)$

Types de grammaires

Classification de Chomsky

linguiste américain (1928) $\rightarrow$ grammaires génératives

- type 0 : aucune contrainte sur les règles
- type 1 : grammaires sous-contexte
  les règles sont de la forme $\alpha\, A\, \beta \rightarrow \alpha\, \gamma\, \beta$ avec
  $A \in V_N, \alpha, \beta, \gamma \in (V_T \cup V_N)^*, \gamma \neq \varepsilon$
- type 2 : grammaires hors-contexte
  les règles sont de la forme $X \rightarrow \beta$ avec
  $X \in V_N, \beta \in (V_T \cup V_N)^*$
- type 3 : grammaires régulières, linéaires (à droite)
  les règles sont de la forme $A \rightarrow a\, B$ ou $A \rightarrow x$ ou $A \rightarrow \varepsilon$
  avec
  $A, B \in V_N, a, x \in V_T$

Grammaire hors-contexte

$G = \{V_T, V_N, S, P\}$

$P \subseteq V_N \times (V_T \cup V_N)^*$

Langage engendré par la grammaire hors-contexte G
$L(G) = \{w \in V_T^* \text{ tels que } S \Rightarrow w\}$

Soit un langage L
L est un langage hors-contexte s'il existe une grammaire hors-contexte qui le reconnait

Grammaire hors-contexte : exemple

Langage des expressions artithmétiques

$V_T = \{*, +, -, \text{idf}, \text{cte}, (,)\}$

mots du langage = lexemes

$\{\text{lexemes}\}$ est un langage régulier

idf désigne un identificateur : `foo`, `toto`, `x`

cte désigne une constante entière : `2`, `123`

$V_N = \{E, T, F\}$ E est l'axiome

Règles de production

$E \to E + T \mid E - T \mid T$

$T \to T * F \mid F$

$F \to \text{cte} \mid \text{idf} \mid (E)$

Langage hors-contexte : reconnaissance

$G = (V_T, V_N, S, P)$

soit $w \in V_T^*$

pour savoir si w appartient au langage $L(G)$ il faut trouver une dérivation à partir de l'axiome qui engendre ce mot

Soient $\gamma_1, \gamma_2, \ldots, \gamma_n \in (V_T \cup V_N)^*$

avec $S \Rightarrow_G \gamma_1 \Rightarrow_G \gamma_2 \Rightarrow_G \Rightarrow_G \ldots \Rightarrow_G \gamma_n \Rightarrow_G w$

c'est-à-dire $S \Rightarrow_G^* w$

donc $w \in L(G)$

Reconnaissance : exemple

$E \to E + T \mid E - T \mid T$
$T \to T * F \mid F$
$F \to \text{cte} \mid \text{idf} \mid (E)$

Soit la phrase $a + b * 3$

Recherche d'une dérivation
$E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow T + T * F \Rightarrow F + T * F \Rightarrow a + T * F \Rightarrow$
$a + T * 3 \Rightarrow a + F * 3 \Rightarrow a + b * 3$

donc $a + b * 3$ appartient au langage

Reconnaissance : dérivations

$E \to E + T \mid E - T \mid T$
$T \to T * F \mid F$
$F \to \text{cte} \mid \text{idf} \mid (E)$

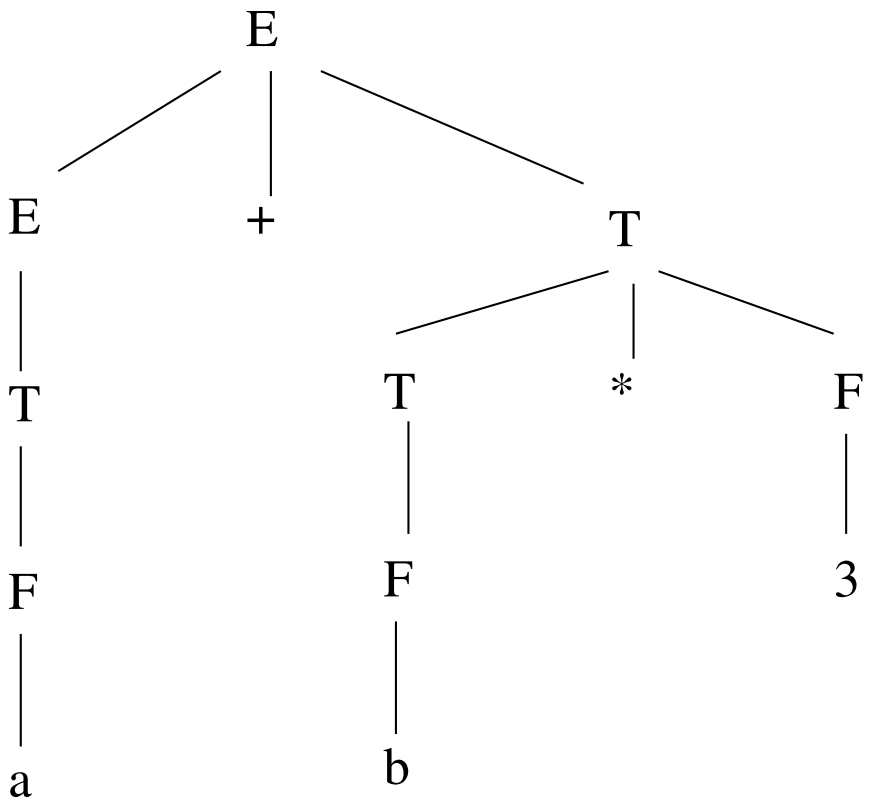
Il y a plusieurs dérivations

$E \Rightarrow E + T \Rightarrow E + T * F \Rightarrow T + T * F \Rightarrow F + T * F \Rightarrow a + T * F \Rightarrow$
$a + T * 3 \Rightarrow a + F * 3 \Rightarrow a + b * 3$

$E \Rightarrow E + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow a + T * F \Rightarrow a + F * F \Rightarrow$
$a + b * F \Rightarrow a + b * 3$

Reconnaissance : arbre de dérivation

Représentation des dérivations par un arbre : arbre syntaxique ou arbre de dérivation



Arbre de dérivation : définition

$G = (V_T, V_N, S, P)$ w un mot de $L(G)$

On appelle *arbre de dérivation* (ou *arbre syntaxique*) de w dans G tout arbre n-aire A dont les noeuds sont des éléments de $(V_T \cup V_N)$ et tel que :

- la racine de A est S (l'axiome de G)
- les feuilles de A sont des éléments de $V_T \cup \{\varepsilon\}$ et la séquence gauche-droite des feuilles de A est égale w
- les noeuds non feuilles de A sont des éléments de V_N et si un noeud non feuille X a pour fils $u_1, u_2, \ldots u_n$ dans A alors la règle $X \to u_1.u_2 \ldots u_n$ doit appartenir P

Arbre de dérivation : ambiguïté

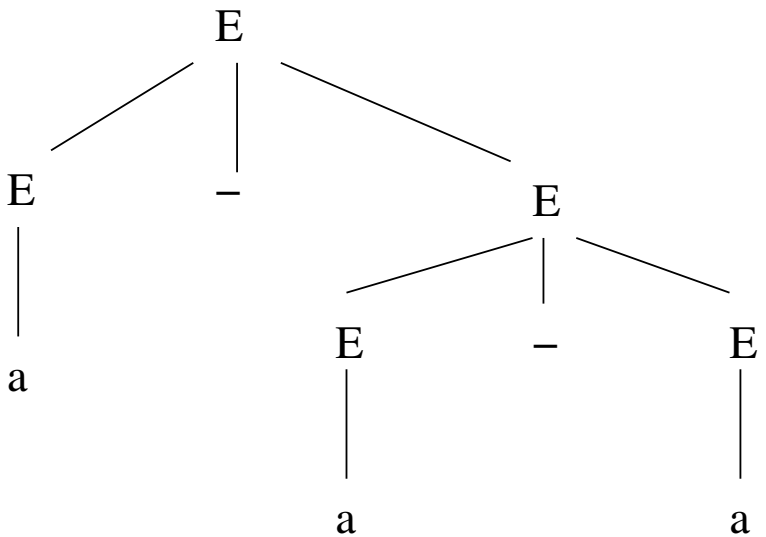
A tout mot de L(G) on peut associer un arbre de dérivation

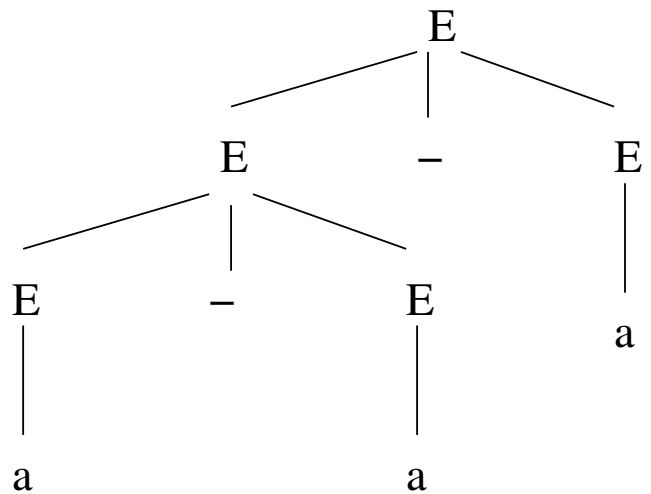
S'il en existe plusieurs alors la grammaire est dite ambigüe

Exemple : ambiguïté

Soit $G = (V_T = \{a, -\}, V_N = \{E\}, E, P = \{E \to E - E, E \to a\})$

La phrase $a - a - a$ peut être interprétée de deux façons différentes





Exemple : priorités, associativité (1/2)

Priorités

$$a + \underbrace{b * c} \qquad \text{ou} \qquad \underbrace{a + b} * c$$

Associativité

$$a + \underbrace{b + c} \qquad \text{ou} \qquad \underbrace{a + b} + c$$

Exemple : priorités, associativité (2/2)

La grammaire des expressions a été écrite de façon à respecter les priorités et associativités usuelles

priorités croissantes : $+$ et $-$, $*$, ()
associativité : $+, -, *$ sont associatifs à gauche

$$a + \underbrace{b * c} \qquad \underbrace{a + b} + c \qquad \underbrace{(a + b)} * c$$

Exercices (1/2)

$V_T = \{*, +, \text{idf}, \text{cte}, (,)\}$
$V_N = \{E, T, F\}$

Considérons les règles de production
$E \rightarrow E * F \mid F$
$F \rightarrow F + T \mid T$
$T \rightarrow \text{cte} \mid \text{idf} \mid (E)$

Dessiner l'arbre de dérivation pour la phrase : $a + b * 3$

Exercices (2/2)

$V_T = \{*, +, \text{idf}, \text{cte}, (,)\}$
$V_N = \{E, T, F\}$

Considérons les règles de production
$E \rightarrow T + E \mid T$
$F \rightarrow T * F \mid F$
$F \rightarrow \text{cte} \mid \text{idf} \mid (E)$

Dessiner l'arbre de dérivation pour la phrase : $a + b + 3$

Langage hors-contexte : reconnaissance

Pour reconnaître une phrase d'un langage hors-contexte on utilise un automate à pile

Intuitivement pour reconnaître $\{a^n\ b^n\}$ nécessité d'avoir une mémoire (pile) pour compter les a et "décompter" les b

Reconnaissance d'un langage hors-contexte : analyse syntaxique