

LF – Théorie des langages formels

Sylvain Brandel

2022 – 2023

sylvain.brandel@univ-lyon1.fr



CM 10

ANALYSE SYNTAXIQUE

Source : Xavier Urbain

Automates finis

- Transducteurs rationnels
 - Automates finis avec transitions étiquetées par une sorte

→ génération de **tokens**
- Erreurs jusqu'ici **LEXICALES**

Grammaires

- Type 0
 - Grammaires générales Pas de restrictions
- Type 1
 - Grammaires contextuelles $uAv \rightarrow uvw$
 - Grammaires croissantes Si $w \rightarrow w' \in R$ alors $|w| \leq |w'|$
- Type 2
 - Grammaires hors contexte Si $w \rightarrow w' \in R$ alors $w \in V$
- Type 3
 - Grammaires régulières

Grammaires

- Grammaires croissantes

$$S \rightarrow ABCS$$

$$S \rightarrow T_c$$

$$CA \rightarrow AC$$

$$BA \rightarrow AB$$

$$CB \rightarrow BC$$

$$CT_c \rightarrow T_c c$$

$$CT_c \rightarrow T_b c$$

$$BT_b \rightarrow T_b b$$

$$BT_b \rightarrow T_a b$$

$$AT_a \rightarrow T_a a$$

$$T_a \rightarrow \varepsilon$$

$$S \rightarrow aSTc$$

$$S \rightarrow aTc$$

$$cT \rightarrow Tc$$

$$T \rightarrow b$$

$$S \rightarrow aSBc$$

$$S \rightarrow abc$$

$$cB \rightarrow Bc$$

$$bB \rightarrow bb$$

(contextuelle)

$$S \rightarrow aSBC$$

$$S \rightarrow aBC$$

$$CB \rightarrow HB$$

$$HB \rightarrow HC$$

$$HC \rightarrow BC$$

$$aB \rightarrow ab$$

$$bB \rightarrow bb$$

$$bC \rightarrow bc$$

$$cC \rightarrow cc$$

- Grammaire croissante \rightarrow grammaire contextuelle

Grammaires

$S \rightarrow ABCS$	$S \Rightarrow ABCS$
$S \rightarrow T_c$	$\Rightarrow ABCABC S$
$CA \rightarrow AC$	$\Rightarrow ABCABCT_c$
$BA \rightarrow AB$	$\Rightarrow ABACBCT_c$
$CB \rightarrow BC$	$\Rightarrow ABABCCT_c$
$CT_c \rightarrow T_c c$	$\Rightarrow AABBCCT_c$
$CT_c \rightarrow T_b c$	$\Rightarrow AABBC T_c c$
$BT_b \rightarrow T_b b$	$\Rightarrow AABBT_b cc$
$BT_b \rightarrow T_a b$	$\Rightarrow AABT_b bcc$
$AT_a \rightarrow T_a a$	$\Rightarrow AAT_a bbcc$
$T_a \rightarrow \varepsilon$	$\Rightarrow AT_a abbcc$
	$\Rightarrow T_a aabbcc$
	$\Rightarrow aabbcc$

Dérivations

- Expressions arithmétiques sur $\{+, \times, (,), \text{id}, \text{cte}\}$
- $E \rightarrow \text{id} \mid \text{cte} \mid E + E \mid E \times E \mid (E)$

tokens

$$x + 2.5 \times 4 + (y + z)$$

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow \text{id} + E \\ &\Rightarrow \text{id} + E \times E \\ &\Rightarrow \text{id} + \text{cte} \times E \\ &\Rightarrow \text{id} + \text{cte} \times E + E \\ &\Rightarrow \text{id} + \text{cte} \times \text{cte} + E \\ &\Rightarrow \text{id} + \text{cte} \times \text{cte} + (E) \\ &\Rightarrow \text{id} + \text{cte} \times \text{cte} + (E + E) \\ &\Rightarrow \text{id} + \text{cte} \times \text{cte} + (\text{id} + E) \\ &\Rightarrow \text{id} + \text{cte} \times \text{cte} + (\text{id} + \text{id}) \end{aligned}$$

- Dérivation : **gauche**
- Parenthésage implicite : $(x + (2.5 \times (4 + (y + z))))$

Dérivations

- Expressions arithmétiques sur $\{+, \times, (,), \text{id}, \text{cte}\}$
- $E \rightarrow \text{id} \mid \text{cte} \mid E + E \mid E \times E \mid (E)$

$$x + 2.5 \times 4 + (y + z)$$

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow E + (E) \\ &\Rightarrow E + (E + E) \\ &\Rightarrow E + (E + \text{id}) \\ &\Rightarrow E + (\text{id} + \text{id}) \\ &\Rightarrow E \times E + (\text{id} + \text{id}) \\ &\Rightarrow E \times \text{cte} + (\text{id} + \text{id}) \\ &\Rightarrow E + E \times \text{cte} + (\text{id} + \text{id}) \\ &\Rightarrow E + \text{cte} \times \text{cte} + (\text{id} + \text{id}) \\ &\Rightarrow \text{id} + \text{cte} \times \text{cte} + (\text{id} + \text{id}) \end{aligned}$$

- Dérivation **droite**
- Parenthésage implicite : $((x + 2.5) \times 4) + (y + z)$

Dérivations

- $((((x + 2.5) \times 4) + (y + z)) \neq (x + (2.5 \times (4 + (y + z))))$
- Dérivation droite avec $((x + (2.5 \times 4)) + (y + z))$?
- G **ambiguë** si $\exists w \in L(G)$ tq plusieurs dérivations droite pour w.
- Lever l'ambiguïté
 - Pas toujours faisable
 - Cas faciles :
 - 1) Nouveau non terminal par **niveau de priorité**
 - 2) Récursif **gauche** si associatif **gauche** (resp. droite)

Dérivations

- Expressions arithmétiques sur $\{+, \times, (,), \text{id}, \text{cte}\}$
- $E \rightarrow \text{id} \mid \text{cte} \mid E + E \mid E \times E \mid (E)$
- $+ < \times$ + moins prioritaire que \times
- $+$ et \times : associatifs à gauche $x+y+z = (x+y)+z$

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow \text{id} \mid \text{cte} \mid (E)$$

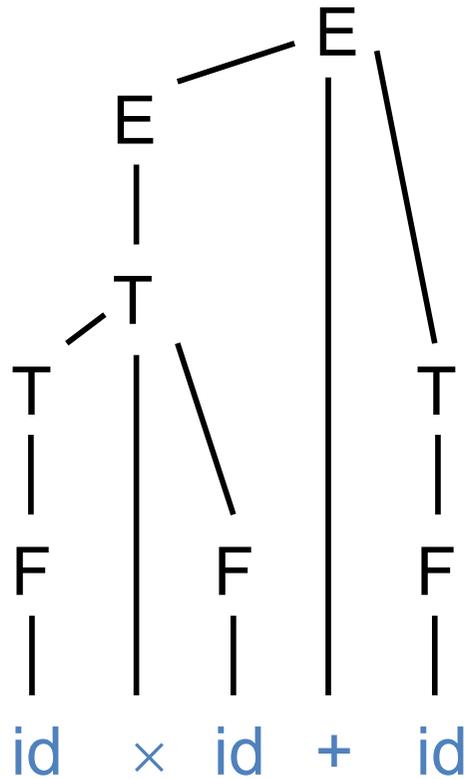
- Unique derivation gauche ou droite
- Un peu plus long et complexe mais univoque

Arbre syntaxique

- Plusieurs dérivations pour un même résultat (permutations, etc.)
 - Représentation invariante
 - Représentation unique lorsque G non ambiguë
- Soit $G = (V, \Sigma, R, S)$, arbres de syntaxe de G :
 - Nœuds internes étiquetés par V
 - Feuilles étiquetées par Σ
 - Si nœud interne N a k fils a_1, \dots, a_k alors $N \rightarrow a_1 \dots a_k \in R$
- Arbre de dérivation : $\Lambda = S$ feuilles $\in \Sigma$

Arbre syntaxique

- Exemple



- Arbre de la dérivation $E \Rightarrow^* \text{id} \times \text{id} + \text{id}$

Arbre syntaxique

- Un arbre de dérivation = plusieurs dérivations
 - Stratégies de parcours (parent avant fils)
- Pour arbre de dérivation : mot des feuilles $\in L(G)$
- Réciproque : produire un arbre, récurrence sur longueur de dérivation
 - Nulle : arbre = feuille a
 - $N \Rightarrow^n w_1 M w_2 \Rightarrow w_1 a_1 \dots a_k w_2$:
ajout de k fils au nœud M de l'arbre de $N \Rightarrow^n w_1 M w_2$
- G ambiguë si $\exists w \in L(G)$ avec deux arbres de dérivation distincts
- Construction de l'arbre vers le haut ou vers le bas ?

Analyse descendante

- Exemple : grammaire de Dick

$$(1) S \rightarrow (S)S$$

$$(2) S \rightarrow \varepsilon$$

- $((()())()) \in L(G)$?

- Construction de l'arbre :

- Lecture à partir de la gauche Left
- Construction dérivation gauche Left
- Règle déterminée par 1 caractère à produire 1

→ LL(1)

Analyse descendante

- Efficace
- Simple

- Pas toutes LL(1)
- Si récursif à gauche → échec

- Par exemple $E \rightarrow E + T \mid T$
 $T \rightarrow T \times F \mid F$
 $F \rightarrow \text{id} \mid \text{cte} \mid (E)$

pas faisable

Analyse ascendante

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow \text{id} \mid \text{cte} \mid (E)$$

- Réursive gauche ...
- $E \rightarrow T$ ou $E \rightarrow E + T$?
 - ➔ Analyse ascendante
- Construction de l'arbre
 - Lecture à partir de gauche
 - Dérivation droite à l'envers
 - ➔ LR
- En particulier construction d'une forêt

Analyse ascendante

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow \text{id} \mid \text{cte} \mid (E)$$

- Racine de forêt : suite des racines des arbres la constituant

- Opérations :

- 1. Lecture

Shift

- 2. Enracinement

Reduce

Sur juxtaposition $f f'$ construction de fN si $N \rightarrow \Lambda(f')$

$\text{id} \times \text{id} + \text{id} ?$

Syntaxe abstraite

- Type et signification : depuis l'arbre de syntaxe
- **Utilisateur** : ce qu'on écrit
- **Concrète** : presque comme on écrit Impropre à bonne compréhension

→ niveau intermédiaire : syntaxe **abstraite** dépolluée

- On veut :
 - Objectif 1 : sans ambiguïté
 - Objectif 2 : sans scories
 - Objectif 3 : structure **et** valeurs

Syntaxe abstraite

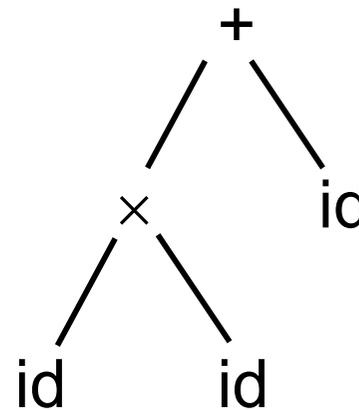
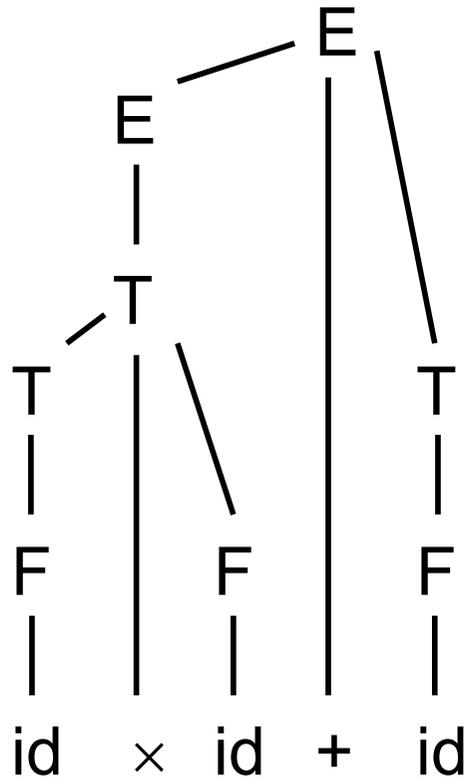
- T ensemble d'étiquettes abstraites
- $\Sigma = T \cup \{ (;); , \}$
- $G = (V, \Sigma, R, S)$ abstraite si
 1. Règles : $N \rightarrow t$
ou $N \rightarrow t(N_1, \dots, N_k)$ pour $N_i \in V, t \in T$
 2. Occurrence t unique dans G
- Mot généré : expression abstraite

$$T = \{p, m, cte\} \quad E \rightarrow p(E,E) \mid m(E,E) \mid cte$$

$$T = \{+, \times, cte\} \quad E \rightarrow +(E,E) \mid \times(E,E) \mid cte$$

Syntaxe abstraite

- Comparaison AS / ASA



- Objectif 2 OK

Syntaxe abstraite

- Pour valeurs : étiquette = langage rationnel

$$E \rightarrow +(E,E) \mid \times (E,E) \mid \text{cte}(\text{Nat})$$
$$\text{Nat} \in (0 \mid 1 \mid \dots \mid 9)^+$$

- Objectif 3 OK

Syntaxe abstraite

- Actions dans une grammaire

$E \rightarrow E + T$	$\{ +(E_1, T_1) \}$
$E \rightarrow T$	$\{ T_1 \}$
$T \rightarrow T \times F$	$\{ \times(T_1, F_1) \}$
$T \rightarrow F$	$\{ F_1 \}$
$F \rightarrow \text{Nat}$	$\{ \text{cte}(\text{val}(\text{Nat}_1)) \}$
$F \rightarrow (E)$	$\{ E_1 \}$

- Erreurs ici **SYNTAXIQUES**

Grammaire du C

<p>unité-de-traduction : déclaration-externe unité-de-traduction déclaration-externe</p> <p>déclaration-externe : définition-de-fonction déclaration</p> <p>définition-de-fonction : spécificateurs-de-déclaration</p> <p>déclaration : spécificateurs-de-déclaration</p> <p>liste-de-déclarations : déclaration liste-de-déclarations déclar</p> <p>spécificateurs-de-déclaration : spécificateur-de-classe-de- spécificateur-de-type spécif qualificatif-de-type spécif</p> <p>spécificateur-de-classe-de-stockage : auto register st</p> <p>spécificateur-de-type : un parmi void char short unioned spécificateur- non-type def</p> <p>qualificatif-de-type : un parmi const volatile</p> <p>spécificateur-de-struct-ou-union : struct ou union identificateur struct-ou-union identificateur</p> <p>struct-ou-union : un parmi struct union</p> <p>liste-de-déclarations-de-struct : déclaration-de-struct liste-de-déclarations-de-struct</p> <p>liste-de-déclarations-init : déclarateur-init liste-de-déclarateurs-init ,</p> <p>déclarateur-init : déclarateur déclarateur = initialiseur</p>	<p>déclaration-de-paramètres : spécificateurs-de-déclaration déclarateur spécificateurs-de-déclaration déclarateur abstraite</p> <p>liste-d'identificateurs : identificateur liste-d'identificateurs , ident</p> <p>initialisateur : expression-d'affectation { liste-d'initialisateurs } { liste-d'initialisateurs , }</p> <p>liste-d'initialisateurs : initialisateur liste-d'initialisateurs , initial</p> <p>nom-de-type : type de qualificatif de spécif</p> <p>déclarateur-abstrait : pointeur pointeur de déclarateur-abstrait</p> <p>déclarateur-abstrait-absoolu : déclarateur-abstrait déclarateur-abstrait absolu déclarateur-abstrait absolu</p> <p>nom-type-def : identificateur</p> <p>instruction : instruction-étiquetée instruction-expression instruction-composée instruction-de-sélection instruction-d'affectation instruction-de-sauvegarde</p> <p>instruction-étiquetée : identificateur : instruction case expression constante default : instruction</p> <p>instruction-expression : expression</p> <p>instruction-composée : { liste-de-déclarations } { }</p> <p>liste-d'instructions : instruction liste-d'instructions instruction</p>	<p>déclaration-de-struct : liste-de-qualificatifs-de-spécificateurs liste-de-déclarateurs-de-struct :</p> <p>liste-de-qualificatifs-de-spécificateurs :</p> <p>instruction-de-sélection : if { expression } instruction if { expression } instruction else instruction switch (expression) instruction</p> <p>expression-égalité : expression relationnelle expression relationnelle expression d'égalité == expression relationnelle expression d'égalité != expression relationnelle</p> <p>expression relationnelle : expression-de-décalage expression relationnelle expression relationnelle expression relationnelle</p> <p>expression-de-décalage : expression-additive expression de décalage expression de décalage</p> <p>expression-additive : expression-multiplicative expression additive + expression additive - expression additive * expression multiplicative /</p> <p>expression-multiplicative : expression-unary expression-multiplicative expression-multiplicative expression-multiplicative</p> <p>expression-unary : expression-unary (nom-de-type) expression</p> <p>opérateur-unary : un parmi & * + - !</p> <p>expression postfixée : expression-primaire expression postfixée (expression postfixée) expression postfixée [expression postfixée] expression postfixée { expression postfixée } expression postfixée [expression postfixée] expression postfixée [expression postfixée]</p>	<p>expression-primaire : identificateur constante chaîne-de-caractères (expression)</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------