Diapositives sans animation disponibles sur Chamilo
(pour téléphone)

Projet : disponible dès maintenant

- À faire par binôme
- À rendre le 7 décembre 2025 sur teide
- Vous ne pourrez pas tout faire dès maintenant :
  - partie lexicale faisable tout de suite
  - partie grammaticale (cours 9 et 10)

Théorie des Langages

Cours 8 : Ambiguïté et algorithmes d'analyse

Lionel Rieg

Grenoble INP - Ensimag, 1re année

Ambiguïté

Définition (Grammaire ambiguë)

Une grammaire est dite ambiguë s'il existe un mot qui peut être engendré par deux dérivations canoniques différentes.

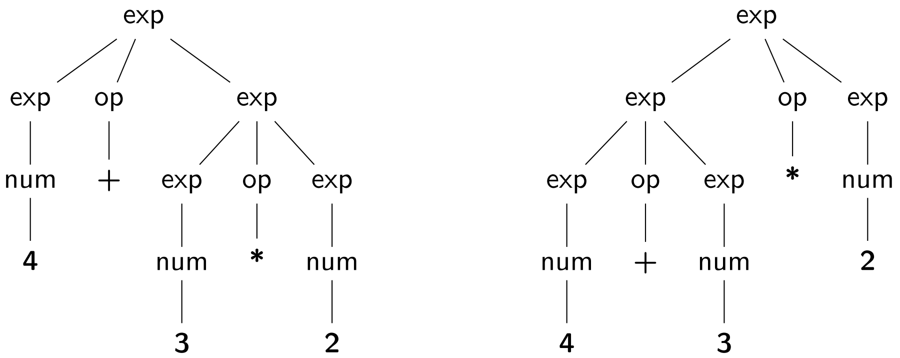
RMQ : Pour les grammaires HC, cela revient à avoir plusieurs arbres de dérivation différents pour un même mot.

Exemple de grammaire ambiguë

Exemple (Expressions arithmétiques)

$$
\begin{array}{lcl}
\text{exp} & \to & \text{num} \mid \text{exp op exp} \mid (\text{exp}) \\
\text{op} & \to & + \mid - \mid * \mid / \\
\text{num} & \to & \mathbf{0} \mid \ldots \mid \mathbf{9}
\end{array}
$$

Deux arbres d'analyse différents pour $4 + 3 * 2$:



Ambiguïté

Définition (Grammaire ambiguë)

Une grammaire est dite ambiguë s'il existe un mot qui peut être engendré par deux dérivations canoniques différentes.

RMQ : Pour les grammaires HC, cela revient à avoir plusieurs arbres de dérivation différents pour un même mot.

Pourquoi cherche-t-on à éviter l'ambiguïté ?
Car cela donne plusieurs sens à un même mot.
(plusieurs programmes différents pour un même code source !)

Définition (langage ambigü)

Un langage est dit (intrinsèquement) ambigü si toute grammaire qui l'engendre est ambiguë.

Exemple : $\{a^n b^n c^m \cup a^m b^n c^n \mid n, m \in \mathbb{N}\}$ (à cause de $a^n b^n c^n$)

Autre exemple

Exemple (Mots bien parenthésés)

$$S \rightarrow SS \mid \mathbf{aSb} \mid \varepsilon$$

Exercice : Montrer que cette grammaire est ambiguë.

Voici une grammaire équivalente non-ambiguë :

$$S \rightarrow \mathbf{aSbS} \mid \varepsilon$$

Questions :

- Comment être sûr que cette grammaire est non-ambigüe ?
- Comment savoir si ces deux grammaires sont équivalentes ?
  $\rightsquigarrow$ montrer chaque inclusion par induction structurelle
  (cf. correction d'une grammaire)

Conditions suffisantes de non-ambiguïté

Idée :

1. On ne peut pas générer un même mot avec deux règles différentes.
   $\Rightarrow$ éviter les règles qui commutent
2. Pour chaque mot, l'instantiation d'une règle est unique.

Théorème (Non-ambiguïté)

Une grammaire HC qui verifie les conditions suivantes est non-ambiguë.

1. *Pour tout couple de règle* $A \rightarrow \alpha$ *et* $A \rightarrow \beta$ *avec* $\alpha \neq \beta$*,*
   $\mathcal{L}(\alpha) \cap \mathcal{L}(\beta) = \emptyset$*.*
2. *Pour toute règle* $A \rightarrow X_1 \ldots X_n$ *avec* $X_i \in V$ *et tout* $w \in V_T^*$ *tel que* $X_1 \ldots X_n \Longrightarrow^* w$*,*
   il existe une unique découpage $w = w_1 \ldots w_n$ *tel que* $X_i \Longrightarrow^* w_i$*.*

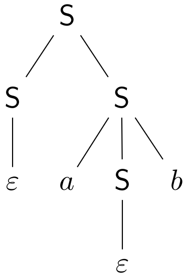
C'est une condition suffisante mais elle n'est pas nécessaire.
Il n'existe pas d'algorithme pour décider l'ambiguïté.

Exemple d'application

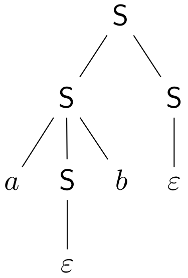
Exemple (Mots bien parenthésés) : $\mathsf{S} \rightarrow \mathsf{SS} \mid \mathbf{aSb} \mid \varepsilon$

- Condition 1 non satisfaite : $\mathcal{L}(SS)$ contient $\mathcal{L}(S)$
  $\mathsf{S} \Longrightarrow \mathbf{aSb} \Longrightarrow \mathbf{ab}$
  $\mathsf{S} \Longrightarrow \mathsf{SS} \Longrightarrow \mathsf{S} \Longrightarrow \mathbf{aSb} \Longrightarrow \mathbf{ab}$
- Condition 2 non satisfaite :
  pour $S \rightarrow SS$ et $SS \Longrightarrow^* ab$, on peut choisir :

$w_1 = \varepsilon$ et $w_2 = ab$



$w_1 = ab$ et $w_2 = \varepsilon$



Exemple d'application

Exemple (Mots bien parenthésés) : S $\rightarrow$ $\mathbf{aSbS}$ | ε

- Condition 1 satisfaite : $\mathcal{L}(\varepsilon) = \{\varepsilon\} \notin \mathcal{L}(aSbS)$
- Condition 2 satisfaite :
  - pour S $\rightarrow$ ε : aucun découpage à faire (car $n = 0$)
  - pour S $\rightarrow$ $\mathbf{aSbS}$ et $\mathbf{aSbS} \Longrightarrow^* w$:
    D'après le théorème de décomposition, w s'écrit aw_1bw_2.
    Montrons que w_1 est unique.
    Idée : Une parenthèse ouvrante a une unique fermante correspondante.
    Il suffit de montrer que tout mot de la forme $\mathbf{aSb}$ n'a pas de préfixe strict de cette forme. En effet, s'il y avait deux découpages possibles aw_1bw_2 de w, l'un des w_1 serait préfixe de l'autre.
    Soit f la fonction $x \mapsto |x|_a - |x|_b$. On peut montrer que :
    - $\forall x \in \mathcal{L}(S), f(x) = 0$
    - $\forall x \in \mathcal{L}(S), \forall u$ préfixe de w, $f(u) \geqslant 0$.

    Tout mot x de la forme $\mathbf{aSb}$ verifie $f(x) = 0$.
    Un préfixe strict de aw_1b serait ou bien ε (qui n'est pas de la forme $\mathbf{aSb}$) ou bien s'écrit av avec v un préfixe de w_1. Alors $f(av) = 1 + f(v) \geqslant 1$ (car $w_1 \in \mathcal{L}(S)$) donc n'est pas de la forme $\mathbf{aSb}$.

Rendre une grammaire non ambiguë

Comment rendre une grammaire non ambiguë ?
Transformer les règles.

Quelques possibilités :

- Commutation de règles (viole la condition 1)
  $\Rightarrow$ ordonner les règles avec un nouveau non-terminal :
  Exemple : $\mathsf{S} \rightarrow \mathbf{a}\mathsf{S} \mid \mathsf{S}\mathbf{b} \mid \varepsilon$
  devient $\mathsf{S} \rightarrow \mathbf{a}\mathsf{S} \mid \mathsf{T} \qquad \mathsf{T} \rightarrow \mathsf{T}\mathbf{b} \mid \varepsilon$
- Plusieurs découpages (viole la condition 2)
  Exemple : $4 + 3 * 2$ signifie $(4 + 3) * 2$ ou $4 + (3 * 2)$?
  Exemple : $1 - 2 - 3$ signifie $(1 - 2) - 3$ ou $1 - (2 - 3)$?
  1. ordonner les règles
     $\Rightarrow$ donner des niveaux de priorités aux opérateurs
  2. associativité
     $\Rightarrow$ autoriser la répétition que d'un côté

Exemple des expressions arithmétiques

Exemple (Expressions arithmétiques avant)

$$\begin{array}{lll} \mathsf{exp} & \rightarrow & \mathsf{num} \mid \mathsf{exp} + \mathsf{exp} \mid \mathsf{exp} - \mathsf{exp} \mid \mathsf{exp} * \mathsf{exp} \mid \mathsf{exp} \,/\, \mathsf{exp} \mid (\mathsf{exp}) \\ \mathsf{num} & \rightarrow & \mathbf{0} \mid \ldots \mid \mathbf{9} \end{array}$$

1. Niveaux de priorité
   - Niveau 0 (le plus prioritaire) : num, ( )
   - Niveau 1 : $*, /$
   - Niveau 2 (le moins prioritaire) : $+, -$
2. Associativité

   $\Rightarrow$ garder le même niveau de priorité d'un seul côté

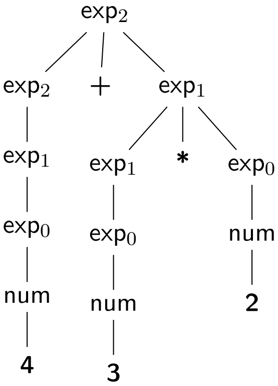
Exemple (Expressions arithmétiques après)

$$\begin{array}{lll} \mathsf{exp}_0 & \rightarrow & \mathsf{num} \mid (\,\mathsf{exp}_2\,) \\ \mathsf{exp}_1 & \rightarrow & \mathsf{exp}_1 * \mathsf{exp}_0 \mid \mathsf{exp}_1 \,/\, \mathsf{exp}_0 \mid \mathsf{exp}_0 \\ \mathsf{exp}_2 & \rightarrow & \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \end{array}$$

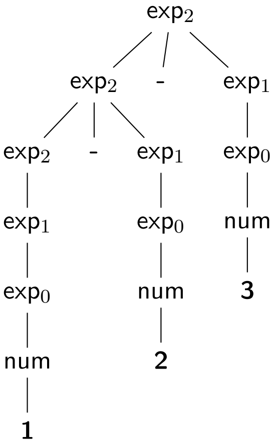
Exemple des expressions arithmétiques

$$
\begin{array}{lcl}
\mathsf{exp}_0 & \rightarrow & \mathsf{num} \;\mid\; (\,\mathsf{exp}_2\,) \\
\mathsf{exp}_1 & \rightarrow & \mathsf{exp}_1 * \mathsf{exp}_0 \;\mid\; \mathsf{exp}_1 \,/\, \mathsf{exp}_0 \;\mid\; \mathsf{exp}_0 \\
\mathsf{exp}_2 & \rightarrow & \mathsf{exp}_2 + \mathsf{exp}_1 \;\mid\; \mathsf{exp}_2 - \mathsf{exp}_1 \;\mid\; \mathsf{exp}_1
\end{array}
$$

$4 + 3 * 2 = 4 + (3 * 2)$



$1 - 2 - 3 = (1 - 2) - 3$



À vous !

Exercice (grammaire des expressions conditionnelles)

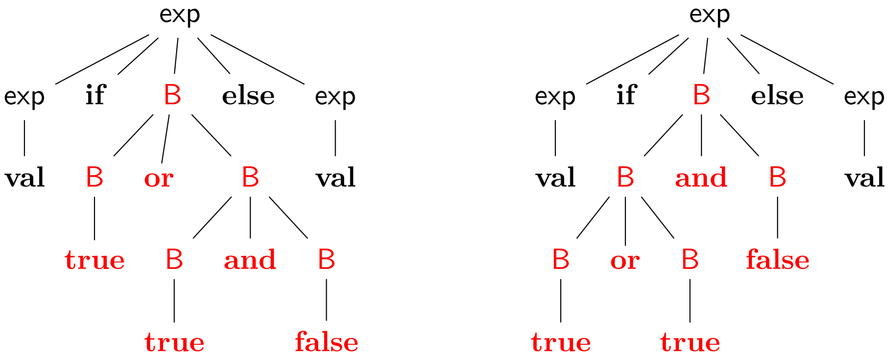
Montrer que la grammaire suivante (extraite de Python) est ambiguë puis la rendre non-ambiguë.

$$\begin{array}{rcl} \text{exp} & \rightarrow & \textbf{val} \mid \text{exp } \textbf{if} \text{ B } \textbf{else} \text{ exp} \\ \text{B} & \rightarrow & \textbf{true} \mid \textbf{false} \mid \textbf{not} \text{ B} \mid \text{B } \textbf{or} \text{ B} \mid \text{B } \textbf{and} \text{ B} \mid (\text{ B }) \end{array}$$

avec $V_T = \{\textbf{and}, \textbf{else}, \textbf{false}, \textbf{if}, \textbf{not}, \textbf{or}, \textbf{true}, \textbf{val}\}$ et $V_N = \{\text{exp}, \text{B}\}$

Priorités :			
	Niveau 2	**or**	associatif à gauche
	Niveau 1	**and**	associatif à gauche
	Niveau 0	**not**	non associatif
		_ **if** _ **else** _	associatif à droite

Correction



$$
\begin{aligned}
\mathsf{exp}_0 &\rightarrow \textbf{val} \\
\mathsf{exp}_1 &\rightarrow \mathsf{exp}_0 \ \textbf{if}\ \mathsf{B}\ \textbf{else}\ \mathsf{exp}_1 \ \mid\ \mathsf{exp}_0
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{B}_0 &\rightarrow \textbf{true} \ \mid\ \textbf{false} \ \mid\ \textbf{not}\ \mathsf{B}_0 \ \mid\ (\ \mathsf{B}_2\ ) \\
\mathsf{B}_1 &\rightarrow \mathsf{B}_1\ \textbf{and}\ \mathsf{B}_0 \ \mid\ \mathsf{B}_0 \\
\mathsf{B}_2 &\rightarrow \mathsf{B}_2\ \textbf{or}\ \mathsf{B}_1 \ \mid\ \mathsf{B}_1
\end{aligned}
$$

Algorithmes d'analyse

Différents algorithmes d'analyse

Question : $w \in \mathcal{L}(G)$?

- Analyse descendante : passer de S à w
  - Algorithme général : semi-décision
    Générer successivement des mots jusqu'à atteindre le mot voulu
  - Pour les grammaires sous-contexte : décision
    Ne générer que les mots de longueur $\leqslant |w|$
    RMQ : Comme les règles sont de la forme $\alpha A \beta \ \rightarrow \ \alpha w \beta$,
    les terminaux ne disparaissent pas.
    $\rightsquigarrow$ optimiser en vérifiant que les terminaux correspondent
  - Pour les grammaires hors-contexte : pas besoin de contexte
    $\rightsquigarrow$ on verifie juste que les terminaux correspondent, puis on les efface
    $\Rightarrow$ On se limite aux dérivations canoniques (gauche ou droite)
    $\Rightarrow$ algorithmes LL ou CYK
- Analyse montante : passer de w à S
  - Même principe pour les algorithmes généraux
  - Mêmes idées d'optimisation
  - Problème potentiel avec $u \ \rightarrow \ \varepsilon$
  - Pour les grammaires hors-contexte : algorithmes LR

Analyse LL

Analyse LL = Left to right reading, Leftmost derivation
Grammaires non ambiguës $\Rightarrow$ une seule dérivation possible par mot
$\leadsto$ La difficulté est de trouver la règle à utiliser à chaque étape.
Pour cela, on regarde les prochains terminaux.

Exemples

- $\{a^n b^n \mid n \in \mathbb{N}\}$: S $\rightarrow$ $\mathbf{aSb}$ | ε
  Si le prochain terminal est a, on utilise S $\rightarrow$ $\mathbf{aSb}$, sinon S $\rightarrow$ ε.
  Exécution : Mot : aabb.
  Dérivation : S

Analyse LL

Analyse LL = Left to right reading, Leftmost derivation

Grammaires non ambiguës $\Rightarrow$ une seule dérivation possible par mot

$\leadsto$ La difficulté est de trouver la règle à utiliser à chaque étape.

Pour cela, on regarde les prochains terminaux.

Exemples

- $\{a^n b^n \mid n \in \mathbb{N}\}$: $\quad \mathsf{S} \rightarrow \mathbf{aSb} \mid \varepsilon$

  Si le prochain terminal est a, on utilise $\mathsf{S} \rightarrow \mathbf{aSb}$, sinon $\mathsf{S} \rightarrow \varepsilon$.

  Exécution : Mot : $\cancel{a}$abb.

  Dérivation : $\mathsf{S} \Longrightarrow \mathbf{aSb}$

Analyse LL

Analyse LL = Left to right reading, Leftmost derivation

Grammaires non ambiguës ⇒ une seule dérivation possible par mot

$\rightsquigarrow$ La difficulté est de trouver la règle à utiliser à chaque étape.

Pour cela, on regarde les prochains terminaux.

Exemples

- $\{a^n b^n \mid n \in \mathbb{N}\}$: $\mathsf{S} \to \mathbf{aSb} \mid \varepsilon$

  Si le prochain terminal est a, on utilise $\mathsf{S} \to \mathbf{aSb}$, sinon $\mathsf{S} \to \varepsilon$.

  Exécution : Mot : $\cancel{a}\cancel{a}$bb.

  Dérivation : $\mathsf{S} \Longrightarrow \mathbf{aSb} \Longrightarrow \mathbf{aaSbb}$

Analyse LL

Analyse LL = Left to right reading, Leftmost derivation

Grammaires non ambiguës ⇒ une seule dérivation possible par mot

$\rightsquigarrow$ La difficulté est de trouver la règle à utiliser à chaque étape.

Pour cela, on regarde les prochains terminaux.

Exemples

- $\{a^n b^n \mid n \in \mathbb{N}\}$: $\mathsf{S} \rightarrow \mathbf{aSb} \mid \varepsilon$

  Si le prochain terminal est a, on utilise $\mathsf{S} \rightarrow \mathbf{aSb}$, sinon $\mathsf{S} \rightarrow \varepsilon$.

  Exécution : Mot : $\cancel{a}\cancel{a}\cancel{b}b$.

  Dérivation : $\mathsf{S} \Longrightarrow \mathbf{aSb} \Longrightarrow \mathbf{aaSbb} \Longrightarrow \mathbf{aabb}$

Analyse LL

Analyse LL = Left to right reading, Leftmost derivation

Grammaires non ambiguës $\Rightarrow$ une seule dérivation possible par mot

$\leadsto$ La difficulté est de trouver la règle à utiliser à chaque étape.

Pour cela, on regarde les prochains terminaux.

Exemples

- $\{a^n b^n \mid n \in \mathbb{N}\}$: $\quad \mathsf{S} \rightarrow \mathbf{aSb} \mid \varepsilon$
  Si le prochain terminal est a, on utilise $\mathsf{S} \rightarrow \mathbf{aSb}$, sinon $\mathsf{S} \rightarrow \varepsilon$.
  Exécution : Mot : $\cancel{a}\cancel{a}\cancel{b}\cancel{b}$.
  Dérivation : $\mathsf{S} \Longrightarrow \mathbf{aSb} \Longrightarrow \mathbf{aaSbb} \Longrightarrow \mathbf{aabb}$
  $\Rightarrow$ On a seulement besoin de connaître 1 terminal : grammaire LL(1)
- $\{a^n b^n \mid n \geqslant 1\}$: $\quad \mathsf{S} \rightarrow \mathbf{aSb} \mid \mathbf{ab}$
  Si le prochain terminal est a, on ne sait pas choisir ...
  mais c'est bon en regardant 2 terminaux !
  si $\mathbf{aa}$, $\mathsf{S} \rightarrow \mathbf{aSb}$ si $\mathbf{ab}$, $\mathsf{S} \rightarrow \mathbf{ab}$ $\Rightarrow$ grammaire LL(2)

Grammaires LL(k) et LL(*) s'il n'y a pas de borne sur k

Exemple non LL(k)

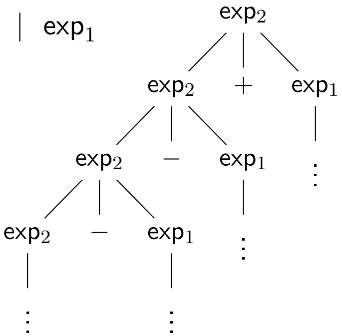
Proposition

Si une grammaire possède un non-terminal X *récursif à gauche, c.-à-d. tel que* $\mathsf{X} \Longrightarrow^{+} \mathsf{X}\alpha$*, elle ne peut pas être LL(k).*

Exemple (Expressions arithmétiques) :

$$\mathsf{exp}_2 \;\rightarrow\; \mathsf{exp}_2 + \mathsf{exp}_1 \;\mid\; \mathsf{exp}_2 - \mathsf{exp}_1 \;\mid\; \mathsf{exp}_1$$

Pour analyser $1 - 2 - 3 + 4$



On verra plus tard comment choisir la bonne règle et comment transformer certaines grammaires pour les rendre $LL(k)$.

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\text{exp}_2 \rightarrow \text{exp}_2 + \text{exp}_1 \mid \text{exp}_2 - \text{exp}_1 \mid \text{exp}_1 \qquad \text{exp}_1 \rightarrow \mathbf{nb}$
On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

ε

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$

On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \rightsquigarrow_S \mathbf{nb}$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\text{exp}_2 \rightarrow \text{exp}_2 + \text{exp}_1 \mid \text{exp}_2 - \text{exp}_1 \mid \text{exp}_1 \qquad \text{exp}_1 \rightarrow \mathbf{nb}$

On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \leadsto_S \mathbf{nb} \leadsto_R \text{exp}_1 \leadsto_R \text{exp}_2$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$

On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \rightsquigarrow_S \mathbf{nb} \rightsquigarrow_R \mathsf{exp}_1 \rightsquigarrow_R \mathsf{exp}_2 \rightsquigarrow_S \mathsf{exp}_2+ \rightsquigarrow_S \mathsf{exp}_2+\mathbf{nb}$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \;\rightarrow\; \mathsf{exp}_2 + \mathsf{exp}_1 \;\mid\; \mathsf{exp}_2 - \mathsf{exp}_1 \;\mid\; \mathsf{exp}_1 \qquad \mathsf{exp}_1 \;\rightarrow\; \mathbf{nb}$
On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \leadsto_S \mathbf{nb} \leadsto_R \mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2+ \leadsto_S \mathsf{exp}_2+\mathbf{nb} \leadsto_R \mathsf{exp}_2+\mathsf{exp}_1$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$
On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \rightsquigarrow_S \mathbf{nb} \rightsquigarrow_R \mathsf{exp}_1 \rightsquigarrow_R \mathsf{exp}_2 \rightsquigarrow_S \mathsf{exp}_2+ \rightsquigarrow_S \mathsf{exp}_2+\mathbf{nb} \rightsquigarrow_R \mathsf{exp}_2+\mathsf{exp}_1$
$\rightsquigarrow_R \mathsf{exp}_2$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation

Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome

$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle $\mathsf{X} \rightarrow \alpha$ en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$

On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \leadsto_S \mathbf{nb} \leadsto_R \mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2+ \leadsto_S \mathsf{exp}_2+\mathbf{nb} \leadsto_R \mathsf{exp}_2+\mathsf{exp}_1$
$\leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2- \leadsto_S \mathsf{exp}_2-\mathbf{nb}$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$
On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \rightsquigarrow_S \mathbf{nb} \rightsquigarrow_R \mathsf{exp}_1 \rightsquigarrow_R \mathsf{exp}_2 \rightsquigarrow_S \mathsf{exp}_2+ \rightsquigarrow_S \mathsf{exp}_2+\mathbf{nb} \rightsquigarrow_R \mathsf{exp}_2+\mathsf{exp}_1$
$\rightsquigarrow_R \mathsf{exp}_2 \rightsquigarrow_S \mathsf{exp}_2- \rightsquigarrow_S \mathsf{exp}_2-\mathbf{nb} \rightsquigarrow_R \mathsf{exp}_2-\mathsf{exp}_1$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation

Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome

$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$

On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \leadsto_S \mathbf{nb} \leadsto_R \mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2+ \leadsto_S \mathsf{exp}_2+\mathbf{nb} \leadsto_R \mathsf{exp}_2+\mathsf{exp}_1$
$\leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2- \leadsto_S \mathsf{exp}_2-\mathbf{nb} \leadsto_R \mathsf{exp}_2-\mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto$ fin

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation

Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome

$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle $X \rightarrow \alpha$ en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$

On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \leadsto_S \mathbf{nb} \leadsto_R \mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2+ \leadsto_S \mathsf{exp}_2+\mathbf{nb} \leadsto_R \mathsf{exp}_2+\mathsf{exp}_1$
$\leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2- \leadsto_S \mathsf{exp}_2-\mathbf{nb} \leadsto_R \mathsf{exp}_2-\mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto$ fin

Dérivation correspondante :

exp_2

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$
On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \leadsto_S \mathbf{nb} \leadsto_R \mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2+ \leadsto_S \mathsf{exp}_2+\mathbf{nb} \leadsto_R \mathsf{exp}_2+\mathsf{exp}_1$
$\leadsto_R \mathsf{exp}_2 \leadsto_S \mathsf{exp}_2- \leadsto_S \mathsf{exp}_2-\mathbf{nb} \leadsto_R \mathsf{exp}_2-\mathsf{exp}_1 \leadsto_R \mathsf{exp}_2 \leadsto$ fin

Dérivation correspondante :
$\mathsf{exp}_2 \Longrightarrow \mathsf{exp}_2-\mathsf{exp}_1$

Analyse LR

Analyse LR = Left to right reading, Rightmost derivation
Grammaire non ambiguë, analyse ascendante : on passe de w à l'axiome
$\Rightarrow$ la dérivation droite se retrouve en lisant à l'envers les règles appliquées

Algorithme : appliquer les deux actions suivantes

- Shift : lire le terminal suivant de l'entrée
- Reduce : appliquer la règle X $\rightarrow$ α en remplaçant α par X

Exemple (Expressions arithmétiques)

$\text{exp}_2 \rightarrow \text{exp}_2 + \text{exp}_1 \mid \text{exp}_2 - \text{exp}_1 \mid \text{exp}_1 \qquad \text{exp}_1 \rightarrow \mathbf{nb}$
On cherche à analyser $\mathbf{nb}+\mathbf{nb}-\mathbf{nb}$.

$\varepsilon \leadsto_S \mathbf{nb} \leadsto_R \text{exp}_1 \leadsto_R \text{exp}_2 \leadsto_S \text{exp}_2+ \leadsto_S \text{exp}_2+\mathbf{nb} \leadsto_R \text{exp}_2+\text{exp}_1$
$\leadsto_R \text{exp}_2 \leadsto_S \text{exp}_2- \leadsto_S \text{exp}_2-\mathbf{nb} \leadsto_R \text{exp}_2-\text{exp}_1 \leadsto_R \text{exp}_2 \leadsto$ fin

Dérivation correspondante :

$$\text{exp}_2 \Longrightarrow \text{exp}_2-\text{exp}_1 \Longrightarrow \text{exp}_2-\mathbf{nb} \Longrightarrow \text{exp}_2+\text{exp}_1-\mathbf{nb}$$
$$\Longrightarrow \text{exp}_2+\mathbf{nb}-\mathbf{nb} \Longrightarrow \text{exp}_1+\mathbf{nb}-\mathbf{nb} \Longrightarrow \mathbf{nb}+\mathbf{nb}-\mathbf{nb}$$

Conflits shift/reduce : quelle action/règle choisir ?

Réalisation et lien avec les automates à pile

Grammaires HC $\iff$ automates à pile

Comment retrouver des piles dans les algorithmes précédents ?

Pile $\approx$ la partie en cours d'analyse

- Analyse LL :

  pile = partie « droite » de l'arbre, qui n'est pas encore complète

  - si haut de pile $\in V_T$:

    si le même que dans l'entrée, on dépile + avance dans l'entrée
  - si haut de pile $\in V_N$:

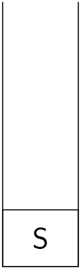
    appliquer X $\rightarrow$ w = dépiler X et empiler les w_i

Analyse LL et pile

Exemple : $S \rightarrow \mathbf{aSb} \mid \varepsilon$

Actions :

- Si le prochain terminal est **a**, utiliser $S \rightarrow \mathbf{aSb}$
- Si le prochain terminal est **b**, utiliser $S \rightarrow \varepsilon$

Mot à analyser	Pile	Arbre	Dérivation
a a b b		S	S

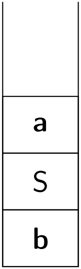
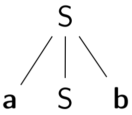
Pile = partie de l'arbre correspondant à l'entrée pas encore consommée

Analyse LL et pile

Exemple : $\mathsf{S} \rightarrow \mathbf{aSb} \mid \varepsilon$

Actions :

- Si le prochain terminal est **a**, utiliser $\mathsf{S} \rightarrow \mathbf{aSb}$
- Si le prochain terminal est **b**, utiliser $\mathsf{S} \rightarrow \varepsilon$

Mot à analyser	Pile	Arbre	Dérivation
a a b b			$\mathsf{S} \Longrightarrow \mathbf{aSb}$

Pile = partie de l'arbre correspondant à l'entrée pas encore consommée

Analyse LL et pile

Exemple : $S \rightarrow \mathbf{aSb} \mid \varepsilon$

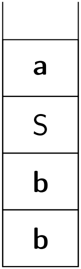
Actions :

- Si le prochain terminal est **a**, utiliser $S \rightarrow \mathbf{aSb}$
- Si le prochain terminal est **b**, utiliser $S \rightarrow \varepsilon$

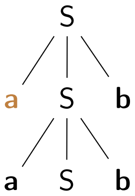
Mot à analyser

a a b b

Pile



Arbre



Dérivation

$$S \Longrightarrow \mathbf{aSb} \Longrightarrow \mathbf{aaSbb}$$

Pile = partie de l'arbre correspondant à l'entrée pas encore consommée

Analyse LL et pile

Exemple : $S \rightarrow \mathbf{aSb} \mid \varepsilon$

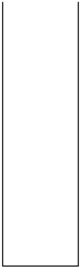
Actions :

- Si le prochain terminal est **a**, utiliser $S \rightarrow \mathbf{aSb}$
- Si le prochain terminal est **b**, utiliser $S \rightarrow \varepsilon$

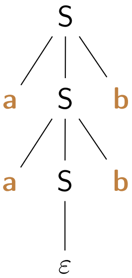
Mot à analyser

a a b b

Pile



Arbre



Dérivation

$$\begin{aligned} S &\Longrightarrow \mathbf{aSb} \\ &\Longrightarrow \mathbf{aaSbb} \\ &\Longrightarrow \mathbf{aabb} \end{aligned}$$

Pile = partie de l'arbre correspondant à l'entrée pas encore consommée

Réalisation et lien avec les automates à pile

Grammaires HC $\iff$ automates à pile

Comment retrouver des piles dans les algorithmes précédents ?

Pile $\approx$ la partie en cours d'analyse

- Analyse LL :
  pile = partie « droite » de l'arbre, qui n'est pas encore complète
  - si haut de pile $\in V_T$:
    si le même que dans l'entrée, on dépile + avance dans l'entrée
  - si haut de pile $\in V_N$:
    appliquer X $\rightarrow$ w = dépiler X et empiler les w_i
- Analyse LR :
  pile = partie « gauche » de l'arbre, racines des arbres déjà construits
  - action Shift :
    on empile le prochain terminal de l'entrée (nouvelle racine)
  - action Reduce :
    appliquer X $\rightarrow$ w = dépiler les w_i et empiler X
    (fusionner les arbres des w_i en un seul de racine X)

Analyse LR et pile

Exemple : $\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$

Actions :

Reduce	Si partie droite d'une règle, remplacer par sa partie gauche
Shift	Sinon empiler le prochain terminal de l'entrée

Mot à analyser	Pile	Arbres
nb + **nb** - **nb**	**nb**	**nb**

Pile = racines des arbres déjà construits

Analyse LR et pile

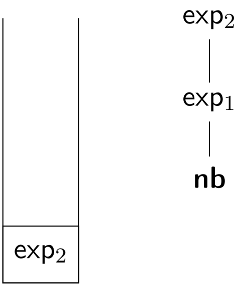
Exemple : $\text{exp}_2 \;\rightarrow\; \text{exp}_2 + \text{exp}_1 \;\mid\; \text{exp}_2 - \text{exp}_1 \;\mid\; \text{exp}_1 \qquad \text{exp}_1 \;\rightarrow\; \textbf{nb}$

Actions :

- **Reduce** Si partie droite d'une règle, remplacer par sa partie gauche
- **Shift** Sinon empiler le prochain terminal de l'entrée

Mot à analyser — Pile — Arbres

nb + **nb** - **nb**



Pile = racines des arbres déjà construits

Analyse LR et pile

Exemple : $\mathsf{exp}_2 \;\to\; \mathsf{exp}_2 + \mathsf{exp}_1 \;\mid\; \mathsf{exp}_2 - \mathsf{exp}_1 \;\mid\; \mathsf{exp}_1 \qquad \mathsf{exp}_1 \;\to\; \mathbf{nb}$

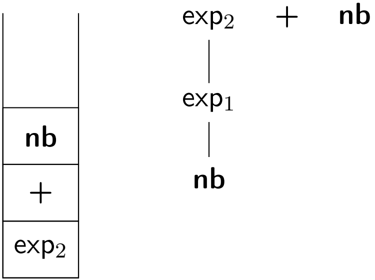
Actions :

Reduce Si partie droite d'une règle, remplacer par sa partie gauche

Shift Sinon empiler le prochain terminal de l'entrée

Mot à analyser | Pile | Arbres

nb + nb - nb



Pile = racines des arbres déjà construits

Analyse LR et pile

Exemple : $\mathsf{exp}_2 \;\rightarrow\; \mathsf{exp}_2 + \mathsf{exp}_1 \;\mid\; \mathsf{exp}_2 - \mathsf{exp}_1 \;\mid\; \mathsf{exp}_1 \qquad \mathsf{exp}_1 \;\rightarrow\; \mathbf{nb}$

Actions :

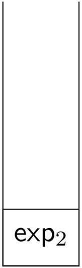
Reduce Si partie droite d'une règle, remplacer par sa partie gauche

Shift Sinon empiler le prochain terminal de l'entrée

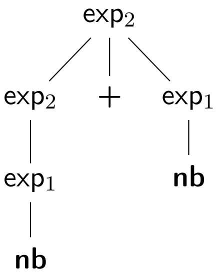
Mot à analyser

nb + nb - nb

Pile



Arbres



Pile = racines des arbres déjà construits

Analyse LR et pile

Exemple : $\mathsf{exp}_2 \rightarrow \mathsf{exp}_2 + \mathsf{exp}_1 \mid \mathsf{exp}_2 - \mathsf{exp}_1 \mid \mathsf{exp}_1 \qquad \mathsf{exp}_1 \rightarrow \mathbf{nb}$

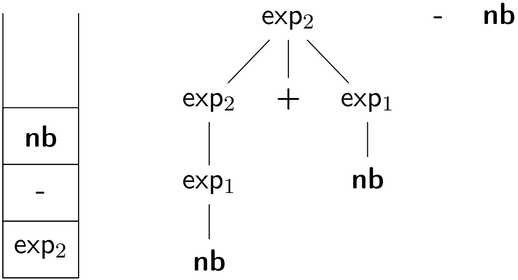
Actions :

Reduce Si partie droite d'une règle, remplacer par sa partie gauche

Shift Sinon empiler le prochain terminal de l'entrée

Mot à analyser — Pile — Arbres

nb + nb - nb



Pile = racines des arbres déjà construits

Analyse LR et pile

Exemple : $\text{exp}_2 \rightarrow \text{exp}_2 + \text{exp}_1 \mid \text{exp}_2 - \text{exp}_1 \mid \text{exp}_1 \qquad \text{exp}_1 \rightarrow \textbf{nb}$

Actions :

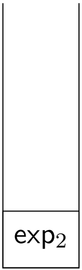
Reduce Si partie droite d'une règle, remplacer par sa partie gauche

Shift Sinon empiler le prochain terminal de l'entrée

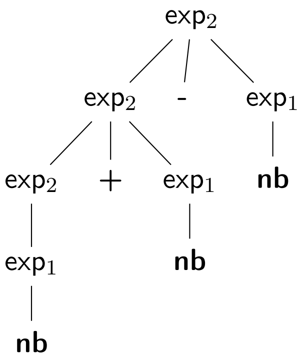
Mot à analyser

nb + nb - nb

Pile



Arbres



Pile = racines des arbres déjà construits

Réalisation et lien avec les automates à pile

Grammaires HC $\iff$ automates à pile

Comment retrouver des piles dans les algorithmes précédents ?

Pile $\approx$ la partie en cours d'analyse

- Analyse LL :
  pile = partie « droite » de l'arbre, qui n'est pas encore complète
  - si haut de pile $\in V_T$:
    si le même que dans l'entrée, on dépile + avance dans l'entrée
  - si haut de pile $\in V_N$:
    appliquer X $\rightarrow$ w = dépiler X et empiler les w_i
- Analyse LR :
  pile = partie « gauche » de l'arbre, racines des arbres déjà construits
  - action Shift :
    on empile le prochain terminal de l'entrée (nouvelle racine)
  - action Reduce :
    appliquer X $\rightarrow$ w = dépiler les w_i et empiler X
    (fusionner les arbres des w_i en un seul de racine X)

Algorithme CYK

Algorithme général d'analyse montante pour une grammaire HC

(plus général mais moins efficace que LL)

Prérequis : G en forme normale de Chomsky

Forme normale de Chomsky

Théorème (Forme normale de Chomsky)

Toute grammaire HC qui ne contient pas ε peut s'écrire sous la forme :

$$X \rightarrow Y\,Z \qquad X \rightarrow a \qquad \text{avec } X, Y, Z \in V_N, a \in V_T$$

Méthode :

- Supprimer les ε-règle (de la forme $\mathsf{X} \rightarrow \varepsilon$)
- Supprimer les 1-règle (de la forme $\mathsf{X} \rightarrow \mathsf{Y}$)
- Dans les règles à plusieurs symboles en partie droite, remplacer chaque terminal $\mathbf{a}$ par un nouveau non-terminal $\mathsf{C}_{\mathbf{a}}$ et ajouter la règle $\mathsf{C}_{\mathbf{a}} \rightarrow \mathbf{a}$
- Remplacer les règles $\mathsf{X} \rightarrow \mathsf{Y}_1 \ldots \mathsf{Y}_n$ par les règles $\mathsf{X} \rightarrow \mathsf{Y}_1\mathsf{Z}_1$, $\quad \mathsf{Z}_i \rightarrow \mathsf{Y}_{i+1}\mathsf{Z}_{i+1}$, $\quad \mathsf{Z}_{\mathsf{n}-2} \rightarrow \mathsf{Y}_{n-1}\mathsf{Y}_n$

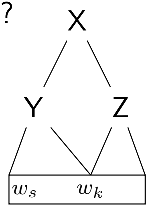
⚠ L'ordre a une influence sur la taille de la grammaire finale.

Algorithme CYK

Algorithme général d'analyse montante pour une grammaire HC
(plus général mais moins efficace que LL)

Prérequis : G en forme normale de Chomsky $(X \rightarrow Y\,Z \qquad X \rightarrow \mathbf{a})$

Idée : Pour un mot $w = w_1 \dots w_n$, avec $w_i \in V_T$,
pour chaque s, l, quels X donnent $X \Longrightarrow^* w_s \dots w_{s+l}$?

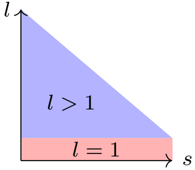


Par programmation dynamique sur l puis s

$l = 1$ regarder les règles $X \rightarrow \mathbf{a}$

$l > 1$ $X \Longrightarrow Y\,Z$ si $\exists k$ tel que $s \leqslant k \leqslant s + l - 1$,
$Y \Longrightarrow^* w_s \dots w_k$ et $Z \Longrightarrow^* w_{k+1} \dots w_{s+l}$

Tableau triangulaire :



Complexité : $\mathcal{O}(n^3|G|)$

- trois boucles imbriquées : l, s, k
- pour chaque couple (Y, Z),
  on parcourt les règles de G

Exemple d'analyse CYK

Phrase à analyser : Le chat mange la souris.

Grammaire :

$$
\begin{aligned}
\mathsf{P} &\rightarrow \mathsf{GN\,GV} \\
\mathsf{GN} &\rightarrow \mathsf{Det\,N} \\
\mathsf{N} &\rightarrow \mathbf{chat} \\
\mathsf{N} &\rightarrow \mathbf{souris} \\
\mathsf{Det} &\rightarrow \mathbf{la} \;\mid\; \mathbf{le} \\
\mathsf{GV} &\rightarrow \mathsf{V\,GN} \\
\mathsf{GV} &\rightarrow \mathbf{mange} \\
\mathsf{V} &\rightarrow \mathbf{mange}
\end{aligned}
$$

Tableau triangulaire :

$l = 5$	P				
$l = 4$					
$l = 3$	P		GV		
$l = 2$	GN			GN	
$l = 1$	Det	N	GV, V	Det	N
	le	**chat**	**mange**	**la**	**souris**