(2) Non déterministe

923

Analyse lexicale et syntaxique. Applications.

I Compilation

pas compilation ^

[S] p13 14 La compilation traduit un programme compréhensible par l'homme en un programme compréhensible par l'ordinateur.

Ex1: . code html en affichage d'une page web
. fichier latex en fichier PDF ...

autres exemples (cam ?)

Elle se compose d'une phase d'analyse qui génère un arbre syntaxique et d'une phase de génération qui le transforme en code machine.

On s'intéresse ici à la phase d'analyse qui comprend une analyse lexicale (qui produit des unités lexicales) suivie d'une analyse syntaxique (qui produit un arbre syntaxique). cf. Annexe 1

II Analyse lexicale

1) Définitions préliminaires

[D] p103 Déf2 Une unité lexicale est un couple constitué d'un nom et d'une valeur attribut optionnelle.

(D) p103 Déf3 Un motif est une description de la forme que les lexèmes d'une unité lexicale peuvent prendre.

(D) p103 Déf4 Un lexème est une séquence de caractères du programme reconnu par le motif d'une unité lexicale.

Rq5: le nom d'une unité lexicale correspond à une classe de lexèmes, tous décrits par un même motif.
Les motifs sont représentés par ~~des expressions rationnelles~~.

(C) p39 Déf6 La classe $\mathcal{E}$ des expressions rationnelles est la plus petite famille d'expressions telle que : $-\ \emptyset \in \mathcal{E}$; $-$ pour toute lettre a, $a \in \mathcal{E}$
et $-\ \forall \epsilon, \epsilon' \in \mathcal{E}$, $\epsilon + \epsilon' \in \mathcal{E}$, $\epsilon . \epsilon' \in \mathcal{E}$, $\epsilon^* \in \mathcal{E}$.

(en peut donner la grammaire BNF)

Ex7

lexème	motif	unité lexicale
"a-23"	$(a|\cdots z)(a|\cdots z|0\cdots 9|-)^*$	(identifiant, a-23)
"3.25"	$(0|\cdots 9)^+ . (0|\cdots 9)^*$	(nombre réel, 3.25)

2) Reconnaissance des lexèmes

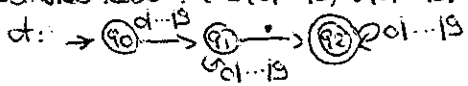
(C) p85 Déf8 Un automate fini $\mathcal{A}$ est un quintuplet (Q, A, δ, I, F) où Q est un ensemble fini d'états, A est un alphabet, $I \subseteq Q$ (resp. $F \subseteq Q$) est l'ensemble des états initiaux (resp. finaux) et $\delta \subseteq Q \times A \times Q$ est l'ensemble des transitions de l'automate.

(C) p110 Th9 (Kleene) Un langage L est décrit par une expression rationnelle ssi il est reconnaissable par un automate fini.

[S] p23 Prop10 Tout automate fini est équivalent à un automate fini déterministe minimal.

Ex11 : Nombres réels : $E = (0|\cdots 9)^+ . (0|\cdots 9)^*$

$\mathcal{A}$:



3) Construction de l'automate pour l'analyse lexicale

(A) p173 (MW) p236 On considère un langage de programmation avec un nombre fini de motifs $\Pi_1, \ldots \Pi_k$. Pour chaque motif Π_i, on considère l'automate fini $\mathcal{A}_i$ associé. On construit $\mathcal{A}$ l'automate minimal (déterministe) équivalent à $\bigcup \mathcal{A}_i$. Chaque état final F de $\mathcal{A}$ contient au moins un état final f_i d'un $\mathcal{A}_i$ (pour le motif Π_i).

On définit alors un ordre de priorité sur les motifs et on notera : type(F) := nom de l'unité lexicale correspondant au motif de plus haute priorité dont un état final est dans F.

4) Algorithme glouton

(MW) p237 (S) p23 Entrée programme $P = p_1 \cdots p_n$
Sortie liste des unités lexicales correspondants à P.

On lit un caractère du texte avec l'automate $\mathcal{A}$.
Si on est dans un état final, on vient de découvrir un lexème. On itère jusqu'à ce qu'un caractère courant p_j bloque l'automate.
Alors . si on est dans un état final F, on a reconnu le lexème $a_1 \ldots a_j$ qui correspond au nom "type(F)"
. sinon, on revient au lexème découvert par le dernier état final rencontré. (S'il n'y en a pas: "ERREUR").
On continue jusqu'à ce qu'on ait parcouru tout P.

II Analyse syntaxique

L'analyse syntaxique se fait sur les noms des unités lexicales. Les règles de syntaxe sont (en général) décrites par une grammaire algébrique.

1) Grammaire et arbre de dérivation

(C) p33 **Déf 12** Une grammaire algébrique G est un triplet (V_N, V_T, P) où V_N et V_T sont des alphabets disjoints (V_N = non-terminaux ; V_T = terminaux) et P est un ensemble (fini) de règles : $P \subseteq V_N \times (V_N \cup V_T)^*$

Rq On note S l'axiome de G.

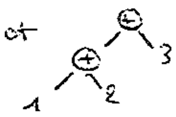
[S] p39 **Ex 13** G : $S \to (S+S)$; $S \to c$

(C) p115 **Déf 14** Un automate à pile est un sept-uplet $(Q, A, \Gamma, q_0, \gamma_0, \delta, F)$ où Q est un ensemble fini d'états, A est l'alphabet du langage, Γ est l'alphabet de pile, $q_0 \in Q$ est l'état initial, $F \subseteq Q$ est l'ensemble des états finaux et $\delta \subset (Q \times A \cup \{\varepsilon\} \times \Gamma) \times Q \times \Gamma^*$ est l'ensemble des transitions de l'automate.

(C) p119 **Prop 15** Un langage L est généré par une grammaire algébrique ssi il est reconnu par un automate à pile.

(S) p37 **Déf 16** Soit G une grammaire algébrique, une dérivation correspond à une application successive de règles de G. On lui associe un arbre de dérivation.

Ex 17 (cf. Ex 13) "((1+2)+3)" est reconnu par G et on a l'arbre de dérivation



→ But de l'analyse syntaxique

Construire l'arbre de dérivation correspondant au programme source.

Rq 18 : Une méthode générique (mais coûteuse !) serait par essais / erreurs

2) CYK : un algorithme générique || DÉV

(P3) p518 (C) p199 Décide si un mot w appartient au langage engendré par une grammaire propre G (ie ne contient pas de règles de la forme $X \to \varepsilon$ et $X \to X'$ avec $X, X' \in V_N$). Il est polynomial en la taille de la grammaire et cubique en la taille du mot.

Les méthodes présentées ci-après ne sont pas génériques et parfois plus coûteuses en la taille de la grammaire mais linéaires en la taille du texte.

3) Analyse descendante dans le cas LL(1)

On construit l'arbre à partir de la racine. Soit $G = (V_N, V_T, P)$ une grammaire algébrique.

Déf 19 Soient $\alpha, \beta \in (V_N \cup V_T)^*$: (A) p226 227

$Premier(\alpha) = \{x \in V_T \mid \exists \beta' \in (V_N \cup V_T)^*, \alpha \to^* x\beta'\}$

$Suivant(\beta) = \{x \in V_T \mid S \to^* \alpha'\beta\gamma', x \in Premier(\gamma'), \alpha', \gamma' \in (V_T \cup V_N)^*\}$

Déf 20 G est LL(1) si $\forall N \in V_N$, en notant $N \to \alpha_1, \dots N \to \alpha_n$ les règles dont la partie gauche est N, on a : (S) p74 (p66)

- les $Premier(\alpha_i)_{1 \le i \le n}$ sont deux à deux disjoints
- si $N \to^* \varepsilon$, alors $Suivant(N)$ est disjoint de chaque $Premier(\alpha_i)$

Calcul pratique :

On cherche un point fixe en appliquant les règles suivantes : (D) p203 Ich

pour $Premier(\alpha)$ avec $\alpha \in (V_T \cup V_N)^*$: - si $\alpha \in V_T$, $Premier(\alpha) = \{\alpha\}$

- si $\alpha \in V_N$ et $\alpha \to \alpha_1 \dots \alpha_k \in P$ avec $k \ge 1$,
  - si $\forall 1 \le i \le k$, $\varepsilon \in Premier(\alpha_i)$ alors $Premier(\alpha) \leftarrow Premier(\alpha) \cup \{\varepsilon\}$
  - si $\exists 1 \le i \le k$, $a \in Premier(\alpha_i)$ et $\varepsilon \in \bigcap Premier(\alpha_j)$ alors $a \in Premier(\alpha)$
- si $\alpha \to \varepsilon \in P$ alors $Premier(\alpha) \leftarrow Premier(\alpha) \cup \{\varepsilon\}$

pour $Suivant(\beta)$ avec $\beta \in V_N$: - $Suivant(S) \leftarrow Suivant(S) \cup \{\#\}$

où # = marqueur de fin.

- si $X \to \alpha\beta\gamma \in P$ avec $\beta \in V_N$ alors $Suivant(\beta) \leftarrow Suivant(\beta) \cup (Premier(\gamma) \setminus \{\varepsilon\})$
- si $X \to \alpha\beta \in P$ avec $\beta \in V_N$ alors $Suivant(\beta) \leftarrow Suivant(\beta) \cup Suivant(X)$ (ou $X \to \alpha\beta\gamma \in P$ avec $Premier(\gamma) \ni \varepsilon$)

(S) p75 **Ex 21** : G | $S \to +SS$; $S \to c$ est LL(1) | G | $S \to SS+$; $S \to c$ n'est pas LL(1)

Rq 22 : Dans le cas LL(1), on regarde un caractère en avant. On construit une table prédictive qui dit quelle règle appliquer en fonction du caractère courant.

[A] p245 **Calcul d'une table prédictive[1]** : colonnes = V_N ; lignes = V_T

$\forall (N \to \alpha) \in P$ - si $a \in Premier(\alpha) \setminus \{\varepsilon\}$: on ajoute $(N \to \alpha)$ à la case $T[N,a]$

- si $\varepsilon \in Premier(\alpha)$ et $b \in Suivant(N)$: on ajoute $(N \to \alpha)$ à la case $T[N,b]$.

[A] p245 Ex 23 . cf annexe 2

4) Analyse ascendante

On construit l'arbre à partir des feuilles

a) Analyse LR(0)

Soit $G = (V_N, V_T, P)$ une grammaire algébrique,

[S] p91 **Déf 24** Un item de G est une règle de la forme $N \to \alpha \uparrow \beta$ où $N \to \alpha\beta \in P$; $\alpha, \beta \in (V_T \cup V_N)^*$.
On note I_G l'ensemble des items de G.

[S] p92 **Déf 25** On appelle règle de réduction (resp. de lecture) un item de la forme $N \to \alpha \uparrow$ (resp. $N \to \alpha \uparrow \beta$ avec $\beta \neq \varepsilon$)

[S] p92 **Déf 26** la clôture de $I \subseteq I_G$ est la plus petite partie $\overline{I} \subseteq I_G$ contenant I et telle que : si $N \to \alpha \uparrow \gamma\beta \in \overline{I}$ et $\gamma \to \gamma' \in P$ alors $\gamma \to \uparrow\gamma' \in \overline{I}$. On note $Q_G = \{\overline{I}, I \subseteq I_G\}$.

[S] p94 **Déf 27** L'automate LR(0) est l'automate d'états Q_G, d'état initial $\{S \to \uparrow\alpha / S \to \alpha \in P\}$, de fonction de transition δ où $\delta : Q_G \times (V_N \cup V_T) \to Q_G$
$(E, \alpha) \longmapsto \{X \to \gamma\alpha \uparrow \beta / X \to \gamma \uparrow \alpha\beta \in E\}$

[S] p101 **Déf 28** Une grammaire G est dite LR(0) si son automate LR(0) n'a pas de conflit (ie tout état qui contient une règle de réduction ne contient que celle-ci).

[S] p116 **Ex 29** $G \begin{cases} S \to Sa \\ S \to a \end{cases}$ est LR(0) cf annexe 3

$G \begin{cases} S \to E \\ E \to E+T \mid T \\ T \to (E) \mid a \end{cases}$ n'est pas LR(0)

b) Analyse LR(1)

[S] p121 **Déf 30** Un item LR(1) de G est un item (LR(0)) et un symbole terminal $a \in V_T \cup \{\#\}$ ($\# =$ marqueur de fin).
On le note $N \to \alpha \uparrow \beta \,/\, a$.

Déf 31 Soit $\beta \in (V_T \cup V_N)^*$, Soit $\tau \subseteq T \cup \{\#\}$

$$Premier^{LR(1)}(\beta, \tau) = \begin{cases} Premier^{LL(1)}(\beta) & \text{si } \beta \not\to^* \varepsilon \\ (Premier^{LL(1)}(\beta) \setminus \{\varepsilon\}) \cup \tau & \text{si } \beta \to^* \varepsilon \end{cases}$$

[S] p122

Déf 32 la clôture de $I \subseteq I_G^{LR(1)}$ est $\overline{I}^{LR(1)}$: la plus petite partie de $I_G^{LR(1)}$ contenant I et telle que :
. Si $N \to \alpha \uparrow \gamma\beta \,/\, \tau \in \overline{I}^{LR(1)}$ et $\gamma \to \gamma' \in P$ alors $\gamma \to \uparrow\gamma' \,/\, Premier^{LR(1)}(\beta\tau) \in \overline{I}^{LR(1)}$

[S] p123

Déf 33 L'automate LR(1) est construit de manière similaire à celle de l'automate LR(0) à partir de l'état initial $\{S \to \uparrow\alpha \,/\, \# \ ; S \to \alpha \in P\}$.

[S] p124

Déf 34 Une grammaire est dite LR(1) si son automate LR(1) n'a pas de conflit.

[S] p126.

Ex 35 $\begin{cases} S \to a \\ S \to ab \end{cases}$ n'est pas LR(0) mais LR(1) cf annexe 4.

[S] p127

c) Analyse SLR(1)

Déf 36 L'automate SLR(1) est l'automate LR(0) où les états $N \to \alpha \uparrow \beta$ sont remplacés par $N \to \alpha \uparrow \beta \,/\, Suivant_{LL(1)}(N)$

[S] p144

Déf 37 Une grammaire G est dite SLR(1) si son automate SLR(1) n'a pas de conflit.

[S] p144

Prop 38 C'est une analyse de type LR(1)

[S] p144

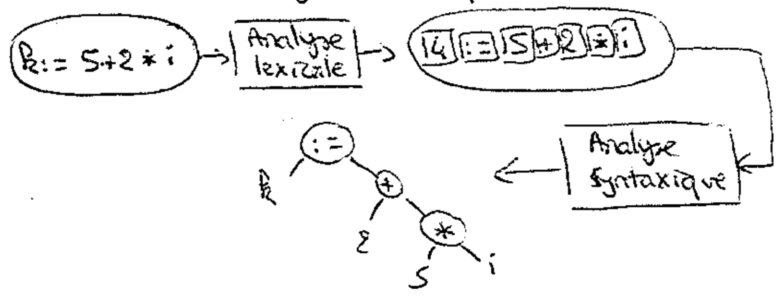
Ex 39 : cf ex 25 Construction des automates LR(0) et SLR(1) FDV

d) Inclusions

[S] p263

On a donc inclusions $LR(0) \subseteq SLR(1) \subseteq LR(1)$ et on en a même avec $LL(1)$, cf annexe 5.

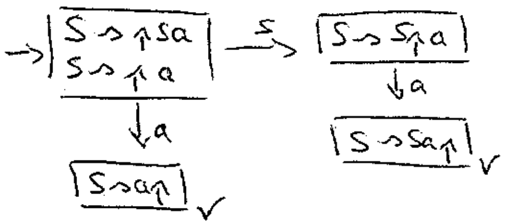
Annexe 1 : Phase d'analyse d'un compilateur [S] p13



Annexe 2 : Table prédictive

	a	+	*	(	)	#
E	E→TM			E→TM		
T	T→FN			T→FN		
M		M→+TM			M→ε	M→ε
N		N→ε	N→*FN		N→ε	N→ε
F	F→a			F→(E)		

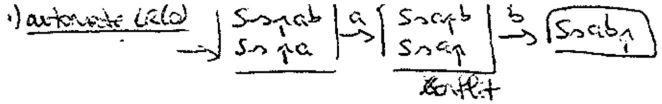
Annexe 3 : Automate LR(0)



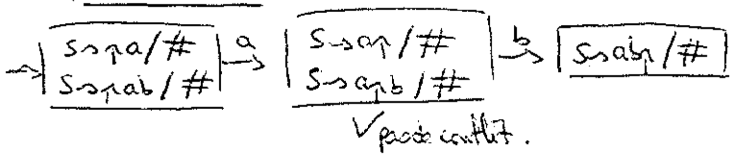
où ✓ = pas de conflit.

Annexe 4 | S→Sab
| S→a

1) automate LR(0)

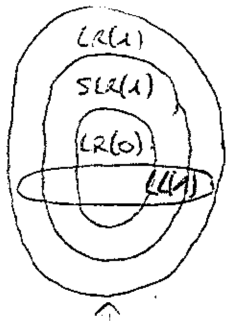


2) automate LR(1)



✓ pas de conflit.

Annexe 5 Inclusions



(Sipser) - Autebert

Comment on montre les inclusions ?

Références

[S] R. Legendre / F. Schwarzentruber, Compilation : analyse lexicale et syntaxique

[D] Dragon

[A] Aho Ait el hadj, Analyse syntaxique et traduction

[C] O. Carton, Langages formels...

[FB] R. Floyd / R. Biegel, Le langage des machines

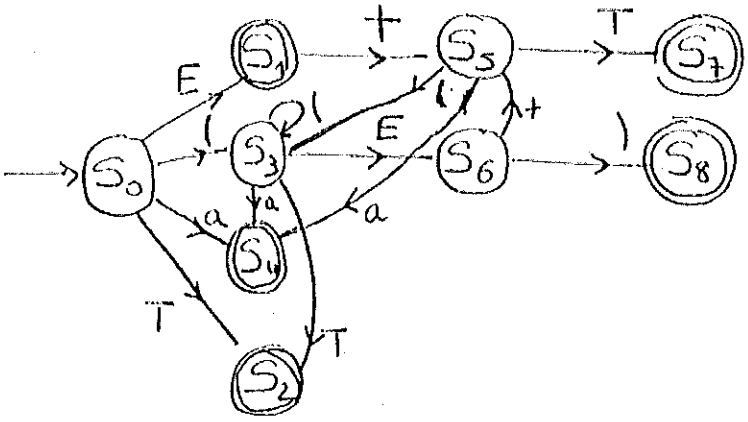
[WM] R. Wilhelm / D. Maurer, Les compilateurs : théorie, construction, génération

[H] J.E. Hopcroft / R. Motwani / J. D. Ullman, Automata Theory, Languages and Computation.

Construction d'un automate LR(0) sur un exemple

I - Construction de l'automate.

G : $S \to E$
$E \to E+T \mid T$
$T \to (E) \mid a$ (a = identificateur)



$S_0 = \{S \to_\uparrow E$
$E \to_\uparrow E+T$
$E \to_\uparrow T$
$T \to_\uparrow (E)$
$T \to_\uparrow a\}$

$S_1 = \{S \to E_\uparrow$
$S \to E_\uparrow + T\}$

$S_2 = \{E \to T_\uparrow\}$

$S_3 = \{T \to (_\uparrow E)$
$E \to_\uparrow E+T$
$E \to_\uparrow T$
$T \to_\uparrow (E)$
$T \to_\uparrow a\}$

$S_4 = \{T \to a_\uparrow\}$

$S_5 = \{E \to E+_\uparrow T$
$T \to_\uparrow (E)$
$T \to_\uparrow a\}$

$S_6 = \{T \to (E_\uparrow)$
$E \to E_\uparrow + T\}$

$S_7 = \{E \to E+T_\uparrow\}$

$S_8 = \{T \to (E)_\uparrow\}$

$S_1 \Rightarrow$ CONFLIT !! $\rightsquigarrow$ **G n'est pas LR(0)**.

II - Construction de l'automate SLR(1).

On regarde un caractère en avant ; on ajoute "Suivant$_{LR(1)}$" dans l'automate pour décider.

S Calcul des suivants :

S	E	T	a
#	+,)	+,)	+,)
#	+,), #	+,), #	+,), #
#	+,), #	+,), #	+,), #

S_1 devient : $S \to E_\uparrow$ / #
$E \to E_\uparrow + T$ / +,), #.

Il n'y a plus de conflit

↳ **G est SLR(1)**

II - Exemple d'analyse :

$m = (a+a)$.

$(a+a)\# \xrightarrow[\text{on lit } (a \to S_4,\ T \to a]{} (T+a)\# \xrightarrow[\text{on lit } (T \to S_2,\ E \to T]{} (E+a)\# \xrightarrow[\text{on lit } (E+a \to S_4,\ T \to a]{} (E+T)\#$

$\xrightarrow[\text{on lit } (E+T \to S_3,\ E \to E+T]{} (E)\# \xrightarrow[(E) \rightsquigarrow S_8,\ T \to (E)]{} T\# \xrightarrow[E \to T]{} E\# \xrightarrow[S \to E]{} S\#$

Autrement dit, on obtient la dérivation suivante :

$S \xrightarrow[S \to E]{} E \xrightarrow[E \to T]{} T \xrightarrow[T \to (E)]{} (E) \xrightarrow[E \to (E+T)]{} (E+T) \xrightarrow[T \to a]{} (E+a) \xrightarrow[E \to a]{} (T+a) \xrightarrow[T \to a]{} (a+a)$

Question : - Vous programmez ? Quels outils ?

CYK = Cocke Younger Kasami

Réf : Carton p 139 / p 90 (FNC) et Floyd/Biegel p 318 / p 373 (FNC)

But : décider si un mot w est engendré par une grammaire G quelconque

Motivation : Il marche pour toutes les grammaires, c'est un algo générique

→ Il est basé sur la programmation dynamique

> **Rq** Ici on se restreint au cas d'une grammaire algébrique propre car on n'a pas le temps de prouver que toute grammaire est équivalente à une grammaire propre.

Entrée : G une grammaire algébrique propre (ie n'a pas de règles de la forme $X \to \varepsilon$ ou $X \to X'$), $G = (V_N, V_T, P, S)$.

$w = w_1 \dots w_n$ un mot de longueur $n \geq 1$ (ie $w \neq \varepsilon$ car G propre donc $G \not\to^* \varepsilon$)

> **Rq** le cas du mot vide peut être traité en regardant Premier(S) (pour une gram. alg. gén.).

Sortie : OUI si $w \in L(G)$; NON sinon

Etape 1 : On se ramène à une grammaire sous forme normale de Chomsky (notée FNC)

Déf Une grammaire algébrique est sous forme normale de Chomsky (FNC) si toutes ses règles sont de la forme $X \to a$ pour $a \in V_T$ et $X \to X_1 X_2$ pour $X_i \in V_N$.

Th Toute grammaire algébrique propre G est équivalente à une grammaire G' sous FNC.

On a de plus $|G'| = O(|G|)$ où $|G|$ = taille de G = nbre total de caractères utilisés ds les règles.

Dém 1) Suppression des terminaux dans les membres droits des règles.

On considère $\{X_a, a \in V_T\}$ un ensemble de nouveaux non-terminaux deux à deux distincts.

On pose $G' = (V_N \cup \{X_a, a \in V_T\}, V_T, P', S)$ avec $P' = \{X_a \to a \mid a \in V_T\} \cup \{X \to \sigma(u) \mid X \to u \in P\}$

où σ est une substitution définie par $\sigma(a) = X_a$ pour $a \in V_T$ et $\sigma(X) = X$ pour $X \in V_N$.

- On a bien : - $L(G') = L(G)$ (car ds les arbres de dérivat°, on ajoute la dérivat° $X_a \to a$ à chaque fois qu'un X_a apparaît (et inversement)).
  - $|G'| = O(|G|)$ (car on ajoute 2 symboles à chaque apparition d'un V_T d'où au max $2|G|$ symboles)

2) Suppression des règles de membre droit ayant une longueur ≥ 3.

Si $X \to X_1 \dots X_k \in P'$ et $k \geq 3$, on la remplace par $X \to X_1 N_1$, $N_1 \to X_2 N_2$ … $N_{k-1} \to X_{k-1} X_k$

où les N_i sont de nouveaux non-terminaux.

- On a bien : - $L(G') = L(G)$ (car ds l'arbre de dérivat°, on remplace $X \to X_1 \dots X_k$ par $X \to X_1 N_1 \dots N_{k-1} \to X_{k-1} X_k$ et inversement)

- $|G'| = O(|G|)$ (car on remplace $k+1$ symboles ($\leq |G|$) par $3k$ symboles ($\leq 3|G|$))
- G' sous FNC.

8.

Rappel : Correction de l'algorithme. Soit G' la FNC équiv de G.

Déf : $\forall 1 \leq i \leq j \leq n$, $E_{i,j} = \{ X \in V_N \;/\; w_i \dots w_j \in L(G(X)) \}$ ie l'ens des non-terminaux à partir desquels on peut dériver $w_i \dots w_j$.

Th : On a la relation de récurrence : $E_{i,i} = \{ X \in V_N \;/\; X \to w_i \in P' \}$, $\forall 1 \leq i \leq n$

et $\forall i < j$: $E_{i,j} = \bigcup_{k=i}^{j} \{ X \in V_N \;/\; X \to X_1 X_2 \in P',\ X_1 \in E_{i,k},\ X_2 \in E_{k+1,j} \}$

dém : cas $i = j$: $w_i \in V_T$ réduit à une lettre et G' FNC donc les seules dérivations de w_i possibles sont règles $X \to w_i$ avec $X \in V_N$ donc $E_{i,i} = \{ X \in V_N \;/\; X \to w_i \in P' \}$.

cas $i < j$: $X \in E_{i,j} \iff X \Rightarrow^*_{G'} w_i \dots w_j \iff \exists i \leq k \leq j,\ X \to X_1 X_2 \in P',\ X_1 \Rightarrow^*_{G'} w_i \dots w_k$ et $X_2 \Rightarrow^*_{G'} w_{k+1} \dots w_j$

($\iff X \Rightarrow^*_{G} (w_i \dots w_j)$ car $L(G') = L(G)$)

$\iff \exists i \leq k \leq j,\ X \to X_1 X_2 \in P',\ X_1 \in E_{i,k},\ X_2 \in E_{k+1,j}$ ∎.

→ On calcule les $E_{i,j}$ par récurrence sur $d = j - i$ (ie $j = d + i$)

Rq : On a $w \in L(G)$ ssi $S \Rightarrow^*_{G} w$ ssi $S \Rightarrow^*_{G'} w \iff S \in E_{1,n}$

Algo : $CYK(w, G)$ =

$G' \leftarrow$ FNC de G — $O(|G|)$

Pr $i = 1$ à n — $O(n^2)$
  Pr $j = 1$ à n — $O(n)$
    $E[i,j] := \emptyset$.

Pr $i = 1$ à n — $O(n|G|)$
  Pr les règles $X \to a \in P$ — $O(|G|)$
    Si $w_i = a$
      $E[i,i] \leftarrow E[i,i] \cup \{X\}$

Pr $1 \leq d \leq n-1$ — $O(n^3|G|)$
  Pr $1 \leq i \leq n-d$ — $O(n^2|G|)$
    Pr $i \leq k \leq d+i$ — $O(n|G|)$
      Pr les règles $X \to X_1 X_2$ — $O(|G|)$
        Si $X_1 \in E[i,k]$ et $X_2 \in E[k+1, i+d]$
          $E[i,i+d] \leftarrow E[i,i+d] \cup \{X\}$.

Retourner $S \in E_{1,n}$

→ en pratique E tableau à n^2 cases
$E(i,j)$ tableau à $|V_N|$ cases
↳ on numérote les $X \in V_N$
et $E(i,j)(k) = 1$
" si $X_k \in E(i,j)$ ".

→ on peut modifier CYK pour obtenir les arbres de dérivation.
↳ mettre les arbres ds $E[i,j]$

→ complexité $O(n^3|G|)$ (temporelle)
($O(n^2|G|)$ (spatiale))

Regarder : Earley.