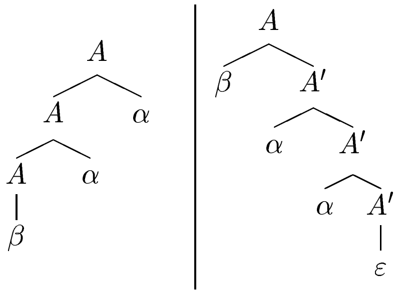
6.3.2.2 Élimination de la récursivité directe

Il vaut mieux adopter un principe plus général, et tenter de construire à partir de la grammaire récursive une grammaire non récursive.

Règle simple Au niveau d'une règle, cela peut se faire simplement :
$A \to A\alpha \mid \beta$, (avec $\beta \neq Au$), devient : $A \to \beta A'$
$A' \to \alpha A' \mid \varepsilon$

La règle $A \to A\alpha$ permet de générer un nombre quelconque de chaîne(s) α, mais pour que la dérivation soit effectivement productive, il faut nécessairement que la récursion s'arrête, c'est-à-dire que A donne β.
On charge alors une règle de commencer par ce β, et ensuite on a une règle récursive (mais pas gauche), pour engendrer autant de fois que nécessaire les α.



Bien entendu, on produit un arbre syntaxique différent dans les deux cas.

Cas général

$$A \longrightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ \ldots \ \mid \beta_k \text{ devient } \begin{cases} A \longrightarrow \beta_1 A' \mid \beta_2 A' \mid \ \ldots \ \mid \beta_k A' \\ A' \longrightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ \ldots \ \mid \alpha_m A' \mid \varepsilon \end{cases}$$

Exemple

E	$\longrightarrow$	$E+T\|T$	se trouve transformé en	E	$\longrightarrow$	TE'
				E'	$\longrightarrow$	$+TE'\|\varepsilon$
T	$\longrightarrow$	$T*F\|F$		T	$\longrightarrow$	FT'
				T'	$\longrightarrow$	$*FT'\|\varepsilon$
F	$\longrightarrow$	$(E)\|a$		F	$\longrightarrow$	$(E)\|a$

2e exemple où l'algo n'élimine pas toute récursivité.

$S \longrightarrow Aa \mid b$	qui devient	S	$\longrightarrow$	$Aa \mid b$	Règle conservée
$A \longrightarrow Ac \mid Sd \mid c$		A	$\longrightarrow$	$SdA' \mid cA'$	Règle transformée
		A'	$\longrightarrow$	$cA' \mid \varepsilon$	

La récursivité directe de la règle $A \longrightarrow A\alpha$ est supprimée, mais on n'a pas éliminé la récursivité indirecte $A \longrightarrow S\gamma \longrightarrow A\alpha'\gamma$.

6.3.2.3 Élimination de la récursivité indirecte

Idée de l'algorithme : pour les paires de règles du type $A \to A'\delta$ et $A' \to A\eta$, on « anticipe » les dérivations problématiques : $A \to A\eta\delta$, et on applique l'algorithme précédent.

Suppression de toutes les récursivités

Données $\mathcal{G}$ grammaire algébrique propre

Résultat $\mathcal{G}'$ grammaire sans récursivité à gauche, t.q. $L_{\mathcal{G}} = L_{\mathcal{G}'}$.

Méthode Soit $(A_1, A_2, ...A_n)$ la liste ordonnée des $A_i \in V$.

Pour tout i de 1 à n faire {
  Pour tout j de 1 à $i-1$ faire {
    Si $A_i \longrightarrow A_j\alpha$ existe, la remplacer par
    les règles : $A_i \longrightarrow \delta_1\alpha \mid \delta_2\alpha \mid \ldots \mid \delta_h\alpha$
    où $A_j \longrightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_h$
  }
  Éliminer la récursivité immédiate des A_i-productions
}

Exemple : grammaire précédente $\begin{array}{l} S \longrightarrow Aa \mid b \\ A \longrightarrow Ac \mid Sd \mid c \end{array}$

Ordre des non-terminaux (arbitraire) : $\{A_1 = S, A_2 = A\}$

- $i = 1 : \emptyset$
- $i = 2\,;\, j = 1$:
  - La règle $A \longrightarrow Sd$ est concernée $(A_{i(=2)} \to A_{j(=1)}\alpha)$
  - On la remplace par $A \longrightarrow Aad \mid bd$
  - On « dérécursive » toutes les $A_(i)$-productions :

    Les règles $A \to Aad \mid Ac \mid c \mid bd$ deviennent $\left\{ \begin{array}{l} A \longrightarrow cA' \mid bdA' \\ A' \longrightarrow adA' \mid cA' \mid \varepsilon \end{array} \right.$
- Fin

La grammaire résultante est $\left\{ \begin{array}{l} S \longrightarrow Aa \mid b \\ A \longrightarrow cA' \mid bdA' \\ A' \longrightarrow adA' \mid cA' \mid \varepsilon \end{array} \right.$ ou $\left\{ \begin{array}{l} S \longrightarrow Aa \mid b \\ A \longrightarrow cA' \mid bdA' \mid c \mid bd \\ A' \longrightarrow adA' \mid cA' \mid ad \mid c \end{array} \right.$

Exercice Appliquer le même algorithme à la grammaire suivante, grammaire de la liste.

$S \to (\,L\,) \mid a$
$L \to L, S \mid S$

Transition
Pour revenir à l'analyse descendante, on garantit une analyse qui termine si on a éliminé la récursivité gauche. Mais on peut aussi chercher à augmenter l'efficacité, en implémentant une analyse « prédictive » : plutôt que de choisir au hasard une règle dont le membre gauche est A lorsqu'on en est à la feuille A de l'arbre, on peut choisir une règle de la forme $A \longrightarrow a\alpha$, en regardant quel est le prochain symbole terminal à produire (ici a). Mais cela impose une forme particulière des règles. D'où les deux transformations considérées ici : Greibach et factorisation.