2.2.3 Récursivité gauche

Un symbole non terminal A d'une grammaire est dit **récursif** si $A \xrightarrow{*} \alpha A \beta$ avec $\alpha, \beta \in (X \cup V)^*$. Si $\alpha = \varepsilon$, A est dit **récursif (à) gauche**. Une grammaire comportant au moins un symbole non terminal récursif gauche est dite récursive gauche. (Idem droite).

La récursivité gauche pose divers problèmes en analyse, et dans l'application de certains algorithmes de transformation de grammaire. C'est la raison pour laquelle on cherche à éliminer cette récursivité, ce qui est toujours possible, car on dispose d'un théorème :

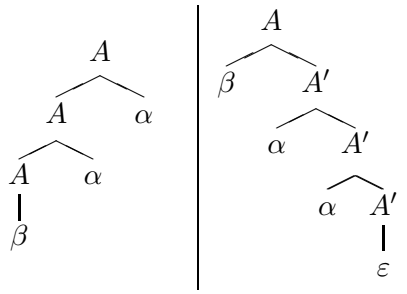
> *Tout langage algébrique peut être généré par une grammaire algébrique non récursive gauche*

La démonstration de ce théorème, comme souvent en pareil cas, repose sur un algorithme qui, étant donnée une grammaire algébrique récursive gauche, produit une grammaire algébrique non récursive gauche qui reconnaît le même langage.

Il faut procéder en deux temps : d'une part, on va éliminer la **récursivité directe** (ou immédiate), puis éliminer la **récursivité indirecte**. On présente ici l'algorithme en commençant par un cas simple, qu'on généralise avant d'éliminer finalement toutes les récursivités.

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 est nécessaire 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.

Généralisation Pour chaque non terminal récursif, on propose :

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

Exemple

$$\begin{array}{lcl} E & \longrightarrow & E + T \mid T \\ T & \longrightarrow & T * F \mid F \\ F & \longrightarrow & (E) \mid a \end{array} \quad \text{se trouve transformé en} \quad \begin{array}{lcl} E & \longrightarrow & TE' \\ E' & \longrightarrow & +TE' \mid \varepsilon \\ T & \longrightarrow & FT' \\ T' & \longrightarrow & *FT' \mid \varepsilon \\ F & \longrightarrow & (E) \mid a \end{array}$$

Appliqué à chaque non terminal récursif, cet algorithme supprime les sources de récursivité

gauche *directes*. Mais, comme on peut le voir avec l'exemple suivant, il y a potentiellement dans les grammaires des sources de récursivité indirectes, qu'il faudra éliminer aussi.

$$\begin{array}{lll} S \longrightarrow Aa \mid b & \text{qui devient} & \begin{array}{lll} S & \longrightarrow & Aa \mid b \\ A & \longrightarrow & SdA' \mid cA' \\ A' & \longrightarrow & cA' \mid \varepsilon \end{array} \\ A \longrightarrow Ac \mid Sd \mid c & & \end{array} \quad \begin{array}{l} \textit{Règle conservée} \\ \textit{Règle transformée} \end{array}$$

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

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 : $A_i \longrightarrow \delta_1\alpha \mid \delta_2\alpha \mid \ldots \mid \delta_h\alpha$
      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.

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