Université de Rouen	Année en cours
Faculté des sciences	Théorie des langages
Licence Informatique $3^{\text{ème}}$ année	Résumé de cours pour le TD n° 9

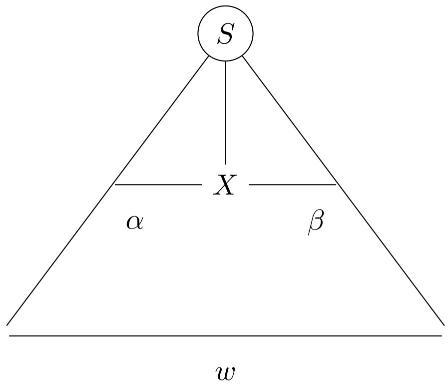
Réduire et rendre propre une grammaire algèbrique

1 Introduction

Nous avons vu dans le cours que plusieurs grammaires algébriques pouvaient engendrer un même langage. On peut alors demander aux grammaires d'avoir certaines propriétés comme de ne pas contenir de symboles inutiles ou bien d'être sous certaines formes, Forme Normale de Chomsky, Forme Normale de Greibach,. . . , en fonction des algorithmes dans lesquels on va les utiliser.

2 Réduire

Soit $G = (N, T, R, S)$ une grammaire algébrique. On dit que $X \in N \cup T$ est un symbole utile dans G si et seulement si il existe une dérivation $S \overset{*}{\Longrightarrow} \alpha X \beta \overset{*}{\Longrightarrow} w$ avec $w \in T^*$, autrement dit si X intervient dans un arbre de dérivation.



Si $S \overset{*}{\Longrightarrow} \alpha X \beta$, on dit que X est accessible à partir de S ou plus simplement *accessible*. Si $X \overset{*}{\Longrightarrow} u \in T^*$, on dit que X est *co-accessible*. Un symbole *utile* est accessible et co-accessible. Un symbole *inutile* est non accessible ou non co-accessible. Une grammaire G est *réduite* si elle ne contient aucun symbole inutile.
Théorème : Toute grammaire admet une grammaire équivalente réduite.
Preuve : algorithme de réduction.
Principe : Suppression des symboles non co-accessibles puis suppression des symboles non accessibles.

Algorithme 1 Suppression des symboles non co-accessibles

Requis : $G = (N, T, S, R)$ une grammaire

Assertion : Retourne $G' = (N', T, S, R')$ telle que $L(G) = L(G')$ où N' est l'ensemble des symboles co-accessibles et R' ne contient que des symboles de N' ou de T.

$NN \leftarrow$FileVide()
$R' \leftarrow \emptyset$
répéter
  $N' \leftarrow NN$
  pour tout $r = (P \rightarrow P_1 P_2 \cdots P_n)$ de R **faire**
    si $P_1 \cdots P_n \in (T \cup N')^*$ **alors**
      $NN \leftarrow NN \cup \{P\}$
      $R' \leftarrow R' \cup \{r\}$
    fin si
  fin pour
jusqu'à $N' = NN$
retourne $G' = (N', T, S, R')$

Algorithme 2 Suppression des symboles non accessibles

Requis : $G = (N, T, S, R)$ une grammaire

Assertion : Retourne $G' = (N', T', S, R')$ telle que $L(G) = L(G')$ où N' et T' sont les non-terminaux et terminaux accessibles et R' ne contient que des symboles de N' ou de T'.

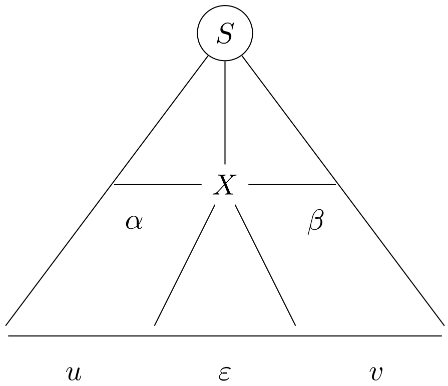
$N' \leftarrow$FileVide()
$T' \leftarrow$FileVide()
$R' \leftarrow$FileVide()
Enfiler(S, N')
répéter
  $NN \leftarrow N'$
  pour tout $r = (P \rightarrow P_1 P_2 \cdots P_n)$ de R **faire**
    si EstPresent(P, NN) **alors**
      $R' \leftarrow R' \cup \{r\}$
      pour i de 1 à n **faire**
        si EstPresent(P_i, N) **alors**
          $N' \leftarrow N' \cup \{P_i\}$
        sinon
          $T' \leftarrow T' \cup \{P_i\}$
        fin si
      fin pour
    fin si
  fin pour
jusqu'à $NN = N'$
retourne $G' = (N', T', S, R')$

3 Rendre propre

Nous avons vu comment supprimer d'une grammaire les symboles terminaux ou non-terminaux qui ne servent à rien pour engendrer des mots. Nous allons voir maintenant que certaines productions allongent inutilement les dérivations ralentissant par exemple le temps de compilation. Une grammaire algébrique est *propre* si elle ne contient ni ε-production ni production unitaire.

3.1 Suppression des ε-productions

On appelle *ε-production* toute production de la forme $X \to \varepsilon$. Tout symbole X de T^* est dit *effaçable* s'il existe une dérivation $X \overset{*}{\Longrightarrow} \varepsilon$. Les ε-productions ne sont vraiment utiles dans une grammaire que pour engendrer le mot vide comme le montre le schéma suivant :



Théorème : Toute grammaire admet une grammaire équivalente à ε près sans ϵ-production.
Preuve : algorithme de suppression des ε-productions.
Principe : L'idée de l'algorithme est de déterminer tout d'abord tous les symboles effaçables de la grammaire puis de construire un nouvel ensemble de règles à partir des règles d'origine en considérant toutes les combinaisons possibles en effaçant ou non les symboles effaçables.

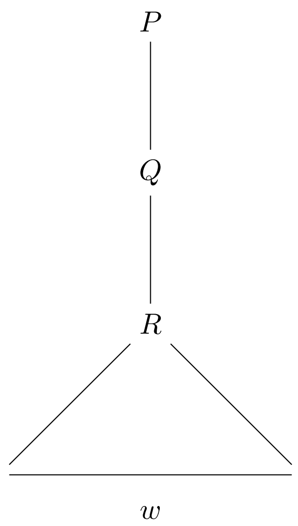
Algorithme 3 Suppression des ε-productions

Requis : $G = (N, T, S, R)$ une grammaire
Assertion : Retourne $G' = (N, T, S, R')$ telle que $L(G) = L(G')$ sans ε-production.

```
R' ←FileVide()
N_1 ←FileVide()
N_2 ←FileVide()
répéter
  N_2 ← N_1
  pour tout P → α de R faire
    si α ∈ N_2^* alors
      N_1 ← N_1 ∪ {P}
    fin si
  fin pour
jusqu'à  N_2 = N_1
pour tout r ∈ R faire
  si r ≠ (P → ε) alors
    R_1 ← {r}
    R_2 ← R_1
    tant que Non(EstVide(R_1)) faire
      (P → X_1 ··· X_n) = Premier(R_1)
      Defiler(R_1)
      si n ≠ 1 alors
        pour i de 1 à n faire
          si EstPresent(X_i, N_2) alors
            si Non(EstPresent(P → X_1 ··· X_{i-1}X_{i+1} ··· X_n, R_2)) alors
              Enfiler(P → X_1 ··· X_{i-1}X_{i+1} ··· X_n, R_1)
              Enfiler(P → X_1 ··· X_{i-1}X_{i+1} ··· X_n, R_2)
            fin si
          fin si
        fin pour
      fin si
    fin tant que
  fin si
  R' ← R' ∪ R_2
fin pour
si S ∈ N_2 alors
  R' ← R' ∪ {S → ε}
fin si
retourne  G' = (N, T, S, R')
```

3.2 Suppression des productions unitaires

On appelle production unitaire une production de la forme $P \to Q$ où P et Q sont des non-terminaux. Les règles unitaires alourdissent inutilement les dérivations comme le montre l'arbre ci-dessous :



Théorème : Toute grammaire admet une grammaire équivalente sans production unitaire.
Preuve : algorithme de suppression des productions unitaires.
Principe : L'idée de l'algorithme est de déterminer tout d'abord pour chaque non-terminal X l'ensemble N_X des non-terminaux que l'on peut atteindre à partir de X par une suite (éventuellement vide) de règles unitaires. On construit ensuite un nouvel ensemble de règles en disant que pour chaque X, l'ensemble des X-règles est l'ensemble des règles non unitaires des symboles de N_X.

Algorithme 4 Suppression des productions unitaires

Requis : $G = (N, T, S, R)$ une grammaire

Assertion : Retourne $G' = (N, T, S, R')$ telle que $L(G) = L(G')$ sans production unitaire.

pour tout P de N **faire**
  $N_P \leftarrow$ FileVide()
  Enfiler(P, N_P)
fin pour
pour tout $P \rightarrow \alpha$ de R **faire**
  si EstPresent(α, N) **alors**
    Enfiler(α, N_P)
  fin si
fin pour
pour tout P de N **faire**
  répéter
    fin$\leftarrow$ VRAI
    pour tout Q de N_P **faire**
      pour tout T de N_Q **faire**
        si Non(EstPresent(T, N_P)) **alors**
          Enfiler(T, N_P)
          fin $\leftarrow$FAUX
        fin si
      fin pour
    fin pour
  jusqu'à fin
fin pour
$R' \leftarrow$ FileVide()
pour tout P de N **faire**
  pour tout $Q \rightarrow \alpha$ de R **faire**
    si EstPresent(Q, N_P) **alors**
      si Non(EstPresent(α, N)) **alors**
        Enfiler$(P \leftarrow \alpha, R')$
      fin si
    fin si
  fin pour
fin pour
retourne $G' = (N, T, S, R')$