Algorithme de Cocke-Younger-Kasami

Manon Ruffini

Le but de l'algorithme est de déterminer si un mot w est engendré par une grammaire G. L'algorithme que l'on va détailler s'appuie sur le principe de programmation dynamique. C'est un algorithme générique (il fonctionne pour toutes les grammaires).

Définition 1 (*Forme normale quadratique*)

Un grammaire algébrique $G = (A, V, P)$ est en forme normale quadratique lorsque toutes ses règles sont d'une des formes suivantes :

- $S \to S_1 S_2$, *où* $S_1, S_2 \in V$
- $S \to a$, *où* $a \in A$

Théorème 1

Soit $G = (A, V, P)$ une grammaire algébrique, soit $S \in V$. Alors il existe une grammaire $\bar{G} = (A, \bar{V}, \bar{P})$ en forme normale quadratique, et $\bar{S} \in \bar{V}$ tels que :

$$L_{\bar{G}}(\bar{S}) = L_G(S) \setminus \{\varepsilon\}$$

Au mot vide près, on peut supposer que la grammaire G est propre. [1]

Étape 1

On se ramène à une grammaire dont les règles sont de la forme :

- $S \to a$, *pour* $a \in V$
- $S \to S_1 \dots S_n$, *où les* $S_i, 1 \leqslant i \leqslant n \in V$

Soit $V' = \{V_a, a \in A\}$ un nouvel ensemble de variables en bijection avec A. Soit $G' = (A, V \cup V', P')$, avec

$$P' = \{V_a \to a, a \in A\} \cup \{S \to \sigma(w), S \to w \in P\}$$

où σ est la substitution définie par : $\sigma(S) = S$ pour $S \in V$ et $\sigma(a) = V_a$ pour $a \in A$.

Étape 2

On remplace les règles $S \to S_1 \dots S_n$, avec $n \geqslant 3$ par les règles

$$\begin{cases} S \to S_1 N_2 \\ N_i \to S_i N_{i+1} \text{ pour } 2 \leqslant i < n-2 \\ N_{n-1} \to S_{n-1} S_n \end{cases}$$

1.

- D'abord, on définit la suite : $U_0 = \{S, S \to \varepsilon \in P\}$, et $\forall n$, $U_{n+1} = U_n \cup \{S, S \to w \text{ et } w \in U_n^*\}$. La suite $(U_n)_n$ est constante à partir d'un certain rang et $\underset{n \in \mathbb{N}}{U_n} = \{S, S \to^* \varepsilon\}$.

  On introduit la substitution suivante : $\sigma(a) = a$, pour $a \in V$ et pour $S \in V$, $\sigma(S) = \begin{cases} S + \varepsilon & \text{si } S \to^* \varepsilon \\ S & \text{sinon} \end{cases}$
- On supprime les règles $S \to \varepsilon$ et on ajoute toutes les règles $S \to u$, où $S \to w \in P$ et $u \in \sigma w$
- On veut supprimer les règles de la forme $S \to S'$. Puisqu'aucune variable ne produit le mot vide on a une dérivation $S \to^* S'$ ssi on a une suite $S \to S_2 \cdots \to S_{n-1} \to S'$. Donc, on supprime les règles de la forme $S \to S'$ et on ajoute les règles $S \to w$, dès que $S \to^* S'$ et $S' \to w$, avec $w \notin V$

On vérifie facilement que la grammaire obtenue est propre et qu'elle engendre le même langage.

On a bien mis la grammaire G sous forme normale quadratique.

On s'intéresse maintenant à l'algorithme CYK.
Soit $G = (A, V, P)$ une grammaire sous forme normale quadratique ; soit $w = a_1 \dots a_n$ un mot, avec $n \geqslant 1$. Pour tous entiers $1 \leqslant i \leqslant j \leqslant n$, on note $w[i, j] = a_i \dots a_j$ et on définit :

$$E_{i,j} = \{X \in V, w[i, j] \in L_G(X)\}$$

— Pour $i = j$, $w[i, j] = a_i$, donc une variable $X \in E_{i,i}$ ssi $X \to a_i \in P$
— Pour $i < j$, une variable $X \in E_{i,j}$ ssi il existe deux variables $X_1, X_2 \in V$ et un entier $k \in [|i, j-1|]$ tels que $X \to X_1X_2 \in P$, $w[i, k] \in L_G(X_1)$ et $w[k+1, j] \in L_G(X_2)$.
FAIRE LE DESSIN !!

Ainsi, l'algorithme calcule les ensembles $E_{i,j}$ par récurrence sur $j - i$.

Algorithme 1 : Cocke-Younger-Kasami

Entrées : Une grammaire G d'axiome S_0 ; un mot $w = a_1 \dots a_n$
Résultat : Oui, si le mot w est engendré par la grammaire G ; non sinon.
si $w = \varepsilon$ **alors**
  retourner $\varepsilon \in L_G(S)$
Mettre G sous forme normale quadratique;
pour $1 \leqslant i \leqslant j \leqslant n$; // $O(n^2)$
 faire
  $E_{i,j} \leftarrow \emptyset$
fin
pour $1 \leqslant i \leqslant n$; // $O(n|G|)$
 faire
  pour tous $S \to a \in P$; // $O(|G|)$
   faire
    si $a_i = a$ **alors**
     $E_{i,i} \leftarrow E_{i,i} \cup \{S\}$
  fin
fin
pour $1 \leqslant d \leqslant n-1$ // $O(n^3|G|)$
 faire
  pour $1 \leqslant i \leqslant n-d$ // $O(n^2|G|)$
   faire
    pour $i \leqslant k \leqslant i+d$ // $O(n|G|)$
     faire
      pour tous $S \to S_1S_2 \in P$ // $O(|G|)$
       faire
        si $S_1 \in E_{i,k}$ *et* $S_2 \in E_{k+1,i+d}$ **alors**
         $E_{i,i+d} \leftarrow E_{i,i+d} \cup \{S\}$
      fin
    fin
  fin
fin
retourner $S_0 \in E_{1,n}$

Complexité de l'algorithme
On considère que la taille de la grammaire est le nombre de symboles terminaux et non terminaux qui apparaissent dans l'écriture de ses règles.

— Si $w = \varepsilon$: ??????
— Mise sous forme normale quadratique :

  D'abord, on ajoute des règles de la forme $V_a \to a$, ce qui correspond au plus à $2|G|$ symboles ; les règles dans $\{S \to \sigma(w)\}$ ont le même nombre de symboles que les règles de G.

  Puis, en transformant les règles $S \to S_1 \dots S_n$, on remplace $n+1$ symboles par $3n$. Donc, $|G'| \leqslant 3|G|$.

  La mise sous forme normale quadratique a une complexité en $O(|G|)$.
— La première boucle de l'algorithme est en $O(n^2)$
— La deuxième boucle est en $O(n|G|)$
— La troisième est en $O(n^3|G|)$

Finalement, l'algorithme CYK a une complexité en $O(n^3|G|)$.

Références

[1] Olivier Carton, *Langages formels*. Vuibert, 2014.