Cocke–Younger–Kasami

Leçons : 907 ; 915 ; 923 ; (926) ; (927) ; 931

Références : CARTON, *Langages formels, calculabilité et complexité* (p.199) [?] et LESESVRE–MONTAGNON–LE BARBENCHON–PIERRON, *131 développements pour l'oral* (p. 779) [?]

Introduction :

L'algorithme Cocke–Younger–Kasami permet de résoudre le problème du mot pour les langages algébriques et permet de dire que ce problème est dans P.

Définition 1. Une grammaire algébrique G est dite sous forme normale de Chomsky si elle ne contient que des règles de la forme $A \to a$ (avec a un terminal) ou $A \to A_1A_2$ (avec A_1, A_2 deux non-terminaux).

Théorème 1. L'algorithme CYK permet de répondre au problème du mot pour les grammaires algébriques

MOT : entrée : un mot $w = w_1w_2\dots w_n$, une grammaire G sous forme normale de Chomsky ; sortie : oui si $w \in \mathrm{L}(G)$, non sinon

Démonstration. Pour tous indices $1 \leqslant i \leqslant j \leqslant n$, on note

$$E_{i,j} = \{\text{non-terminaux qui engendrent } w_i \dots w_j\}$$

Ainsi on a donc $w \in \mathrm{L}(G)$ si et seulement si $S \in E_{1,n}$.[1] On veut donc calculer $E_{1,n}$.

Pour cela, on va calculer tous les $E_{i,j}$ en utilisant le lemme suivant :

Lemme 1. (i) Pour tout $i \in \{1, \dots, n\}$,

$$E_{i,i} = \{A, A \to w_i \text{ est une règle de production de } G\}$$

(ii) Pour tout $1 \leqslant i < j \leqslant n$, on a

$$E_{i,j} = \bigcup_{\substack{A \to B_1B_2 \\ B_1 \in E_{i,k} \\ B_2 \in E_{k+1,j} \\ i \leqslant k < j}} \{A\}$$

Démonstration du lemme. (i) Immédiat puisque la grammaire est sous forme normale de Chomsky.

(ii) $\supset$ S'il existe B_1, B_2, k tels que

$$B_1 \to^* w_i \dots w_k, \quad B_2 \to^* w_{k+1} \dots w_j \quad \text{et} \quad A \to B_1B_2$$

Alors on a $A \to^* w_i \dots w_j$ d'où $A \in E_{i,j}$.

$\subset$ Réciproquement, si $A \to^* w_i \dots w_j$ (i.e $A \in E_{i,j}$) (avec au moins 2 lettres car $i \neq j$) alors par le lemme de factorisation, il existe B_1, B_2, k tels que $B_1 \in E_{i,k}$, $B_2 \in E_{k+1,j}$ et $i \leqslant k < j$ car G est sous forme normale de Chomsky.

□

1. S est l'axiome de la grammaire G

Pour calculer $E_{i,j}$ (avec $i < j$), on a besoin des $E_{i,k}$ et $E_{k+1,j}$ pour tout $i \leqslant k < j$.

$$\begin{pmatrix} E_{1,1} & \cdots & E_{1,i} & \cdots & E_{1,j} & \cdots & E_{1,n} \\ & \ddots & \vdots & & \vdots & & \vdots \\ & & E_{i,i} & \to & E_{i,j} & \cdots & E_{i,n} \\ & & & \ddots & \uparrow & & \vdots \\ & & & & E_{j,j} & \cdots & E_{j,n} \\ & \emptyset & & & & \ddots & \vdots \\ & & & & & & E_{n,n} \end{pmatrix}$$

On utilise la programmation dynamique en commençant par les coefficients diagonaux et en remontant à chaque étape.

Ordre de calcul :

$$\begin{matrix} 1 & 2 & \dots & \dots & n \\ & \ddots & \ddots & & \vdots \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & 2 \\ & & & & 1 \end{matrix}$$

Procédure CYK(un mot w, une grammaire G)
- initialiser $E_{i,j} = \emptyset$;
- **Pour** i **de** 1 **à** n **faire**
  - **Pour** $A \to a \ \in G$ **faire**
    - **Si** $(a = w_i)$ **alors**
      - $E_{i,i} = E_{i,i} \cup \{A\}$
    - **Fin Si**
  - **Fin Pour**
- **Fin Pour**
- **Pour** d **de** 2 **à** n **faire**
  - **Pour** $E_{i,j}$ sur la diagonale d **faire**
    - **Pour** k **de** i **à** $j-1$ **faire**
      - **Pour** $A \to B_1 B_2 \ \in G$ **faire**
        - **Si** ($B_1 \in E_{i,k}$ et $B_2 \in E_{k+1,j}$) **alors**
          - $E_{i,j} = E_{i,j} \cup \{A\}$
        - **Fin Si**
      - **Fin Pour**
    - **Fin Pour**
  - **Fin Pour**
- **Fin Pour**
- **Renvoyer** $S \in E_{1,n}$

Fin

Pour implémenter les ensembles $E_{i,j}$, on voudrait pouvoir *ajouter les non-terminaux* et *vérifier si des non-terminaux sont dans un certain ensemble* $E_{i,j}$ en temps constant. Pour faire cela, on peut construire une matrice qui contient à la case (i, j) un tableau indexé par les non-terminaux et valant "vrai" si le non-terminal est dans $E_{i,j}$, "faux" sinon.

Complexité : $O(n^3|G|)$ où $|G|$ est le nombre de règles de G. □

Proposition 1. Pour toute grammaire algébrique G, il existe une grammaire algébrique G' sous forme normale de Chomsky telle que $\mathrm{L}(G') = \mathrm{L}(G)\backslash\{\varepsilon\}$.

Démonstration.
Étape 1 : *Obtenir les règles $A \to a$*
On a $G = (\Sigma, S, V, T, P)$.[2] On pose alors $G'' = (\Sigma, S, V', T, P')$ où

$$V' = V \cup \{V_a, a \in \Sigma\}$$

et

$$P' = \tilde{P} \cup \{V_a \to a, a \in \Sigma\}$$

où $\tilde{P}$ sont les mêmes règles que P mais où l'on a remplacé le terminal a par le non-terminal V_a.
On obtient alors une grammaire telle que $\mathrm{L}(G') = \mathrm{L}(G)\backslash\{\varepsilon\}$ et avec des règles de la forme :

$$T \to T_1 \dots T_n \text{ et } T \to a$$

où les lettres majuscules sont des non-terminaux et les minuscules sont des terminaux.

Étape 2 : *Transformer les règles $T \to U$*
Pour chaque règle $T \to U$, on duplique toutes les règles qui contiennent des T à droite, et, pour les copies, on remplace T par U. On peut maintenant supprimer la règle $T \to U$ sans changer le langage engendré par la grammaire.

Étape 3 : *Transformer les règles $T \to T_1 \dots T_n$ pour $n > 2$*
Pour chaque règle $T \to T_1 \dots T_n$ (avec $n > 2$) de G'', on la remplace par

$$T \to T_1 T_2', \quad T_i' \to T_i T_{i+1}' \text{ et } \quad T_{n-1}' \to T_{n-1} T_n$$

où $i \in \{2, \dots, n-2\}$. On obtient ainsi une grammaire G' équivalente à G'' et qui est sous forme normale de Chomsky. □

Remarques :
Ainsi on peut résoudre le problème du mot pour n'importe quelle grammaire algébrique car la complexité de la transformation que l'on a faite est linéaire.
Une grammaire sous forme de Chomsky ne peut pas produire ε mais on peut ajouter aux règles la règle $S \to \varepsilon$ (à l'axiome) pour pouvoir produire ε (cependant, souvent on veut travailler avec des grammaires "propres" donc qui ne produisent pas ε)

Astuces de l'agrégatif :
En tournant l'algorithme comme solution au problème du mot pour les langages algébriques, je fais rentrer ce développement dans le leçon classe de complexité, exemple de problème P.
Je passe assez rapidement sur la transformation de la grammaire en forme normale de Chomsky, en fonction du temps qui me reste.

2. Σ est l'alphabet, S est l'axiome, V est l'ensemble des non-terminaux (ou variables), T est l'ensemble des terminaux et P est l'ensemble des règles de production de G

Questions possibles :

- Est-ce que cela marche pour le mot vide ε ?
- Donner un algorithme qui vérifie si ε appartient au langage engendré par une grammaire G ou non.
- Quelle est la définition de $|G|$?
- Quelle est la taille de l'entrée ? puisqu'on considère un mot w et une grammaire G
- Est ce que la version récursive de cet algorithme est mieux ?
- En pratique, est-ce qu'on utilise cet algorithme ?

  Une réponse possible est que pour l'analyse syntaxique, on utilise plutôt des algorithmes linéaires mais ce sont des algorithmes qui ne marchent que pour des grammaires $LL(1)$ et $LR(0)$ et toute grammaire n'est pas forcément de ce type
- Après avoir mis sous forme normale de Chomsky la grammaire suivante $G = (\Sigma, \mathcal{N}, \mathcal{T}, S, \mathcal{R})$ où $\Sigma = \{a, b, c, d\}$, $\mathcal{N} = \{S, A, B, C\}$, $\mathcal{T} = \Sigma$ et $\mathcal{R}$ est décrit par

$$\begin{aligned} S &\to AB \mid Ca \\ A &\to aAb \mid \varepsilon \\ B &\to bB \mid b \\ C &\to cC \mid d, \end{aligned}$$

  appliquer l'algorithme CYK afin de savoir si les mots *aabbbb*, *ccddda* et *aabb* sont dans L(G). Comprendre l'intérêt de l'analyse LL(1), voir Caractérisation de Premier pour l'analyse $LL(1)$.
- En réduisant le problème de Post, montrer l'indécidabilité de Inter, Univ et Egal.