Algorithme d'Earley

Julien Devevey

2018-2019

Ref : Biegel, Floyd - Le Langage des Machines p.321

Algorithme 1. ϵ représente le mot vide. Dans l'algorithme suivant, on supposera que $\epsilon \notin L(G)$. On supposera aussi que G ne contient ni règles vides, ni règles unitaires.

Algorithm 1 Algorithme d'Earley

Entrée: G une grammaire et w un mot
Sortie: True si $w \in G$, False sinon

1: $OBJET \leftarrow$ tableau de taille $n+1$ contenant des listes vides
2: **pour** $S \longrightarrow \beta$ règle de G **faire**
3: Rajouter $(0, 0, S, \epsilon, \beta)$ à $OBJET[0]$
4: **fin pour**
5: **pour** $j \leftarrow 0$ à n **faire**
6: **pour** $(i, j, P, \alpha, Q\gamma)$ dans $OBJET[j]$ (dont les objets rajoutés pendant cette boucle) et $Q \longrightarrow \delta$ règle de G **faire**
7: Ajouter $(j, j, Q, \epsilon, \delta)$ à $OBJET[j]$ (sauf s'il y est déjà)
8: **fin pour**
9: **pour** $(i, j, P, \alpha, c\gamma)$ de $OBJET[j]$ tq $c = w[j+1]$ **faire**
10: Ajouter $(i, j+1, P, \alpha c, \gamma)$ à $OBJET[j+1]$
11: **fin pour**
12: **pour** $(i, j+1, P, \alpha, \epsilon) \in OBJET[j+1]$ et $(h, i, R, \gamma, P\delta) \in OBJET[i]$ **faire**
13: Ajouter $(h, j+1, R, \gamma P, \delta)$ à $OBJET[j+1]$
14: **fin pour**
15: **fin pour**
16: **si** il existe un objet de la forme $(0, n, S, \alpha, \epsilon)$ dans $OBJET[n]$ **alors**
17: **return** True
18: **sinon**
19: **return** False
20: **fin si**

Les lignes 1 à 4 sont l'étape d'initialisation, 6 à 8 l'étape de fermeture, 9 à 11 l'étape de progression, 12 à 14 l'étape de complétion.

Remarque 2. L'algorithme a pour but d'engendrer des objets de la forme (i, j, P, α, β), qui sont stockés dans $OBJET[j]$ (on les classe suivant le deuxième coefficient du

quintuplet). Un objet de la forme (i, j, P, α, β) code deux choses : il signifie que $w[i+1..j]$ est dérivable à partir de α, c'est à dire que $\alpha \Rightarrow^* w[i+1..j]$ mais qu'il reste encore à trouver les k tels que $\beta \Rightarrow^* w[j+1..k]$ et si on trouve un tel k alors $P \Rightarrow^* w[i+1..k]$.

- L'étape d'initialisation (1-4) dit donc qu'on cherche tous les préfixes de w qui sont dérivables à partir des règles de G.
- L'étape de fermeture (6-8) quant à elle, regarde tout ce que contient $OBJET[j]$, et crée le nouvel objet $(j, j, Q, \epsilon, \delta)$, qui cherche alors les préfixes de $w[j+1..n]$ qu'on peut dériver à partir de la règle $Q \longrightarrow \delta$: on "épure" l'objet en enlevant ce qu'on a déjà trouvé (que de α on peut dériver $w[i+1..j]$) et les objectifs trop lointain (le γ).
- L'étape de progression (9-11) est le moment où on regarde la lettre $w[j+1]$. En effet, si on a une règle dans les objectifs de la forme $(i, j, P, \alpha, w[j+1]\gamma)$ alors cet objet peut être mis à jour, donc transformé, en $(i, j+1, P, \alpha w[j+1], \gamma)$.
- L'étape de complétion (12-14) correspond à la mise à jour des objectifs en recombinant avec ce qui a été trouvé précédemment. En effet, si on peut dériver $w[i+1..j]$ à partir de P et on a l'objet $(h, i, R, \gamma, P\delta)$ alors on peut créer l'objet $(h, j, R, \gamma P, \delta)$: on est arrivés au bout de la dérivation qui part de P.

Une fois sorti de la boucle, on vérifie si on peut dériver $w[1..n] = w$ tout entier, et on renvoie la réponse.

Théorème 3. Cet algorithme est correct.

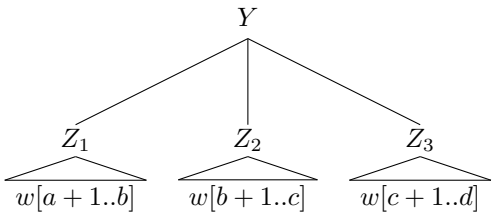
Démonstration.

Le premir constat qu'on effectue est que chaque objet n'est construit par l'algorithme que si l'on sait que la dérivation codée est possible, l'algorithme ne peut alors accepter que des mots qui sont dérivables dans la grammaire. Il reste alors à prouver que tous les mots de $L(G)$ sont acceptés par l'algorithme. Pour ça, on se donne $w \in L(G)$: il possède donc un arbre de dérivation qu'on appelle T.

On procède par récurrence structurelle sur cette arbre de dérivation. On pose l'hypothèse suivante pour les sous-arbre de dérivation de racine Y du mot w et de production $w[a+1..d]$. "Si pour tout β pour lequel il existe une règle du type $Y \longrightarrow \beta$, les objets $(a, a, Y, \epsilon, \beta)$ ont été produits par l'algorithme. Alors l'algorithme produira un objet de la forme $(a, d, Y, \beta, \epsilon)$, c'est à dire montrera qu'on a $Y \Rightarrow^* w[a+1..d]$."

L'hypothèse de récurrence est vraie pour toutes les feuilles de l'arbre, qui sont en fait des terminaux, et on montre bien que l'algorithme engendre toujours les objets de la forme (a, a, c, c, ϵ) où c est une lettre lors de l'étape de progression.

Soit donc un arbre de dérivation de w et Y un sous-arbre. On suppose de plus que l'hypothèse de récurrence est vraie pour tous les sous-arbres propres de Y. On va supposer, que les fils de Y sont des variables ou des terminaux $Z_1, Z_2, \ldots, Z_n$, comme dans le schéma suivant pour $n = 3$:



Observons le cas où Z_1 est un terminal, c'est à dire $Z_1 = w[a+1]$. L'étape de progression va directement créer ce dont on a besoin pour continuer, c'est à dire l'objet $(a, a+1, Y, Z_1, Z_2 \dots Z_n)$ à partir de l'objet $(a, a, Y, \epsilon, Z_1 Z_2 \dots Z_n)$, obtenu lors de la fermeture de Y.

Examinons maintenant le cas où c'est une variable : L'étape de fermeture crée pour $j = a$ l'objet $(a, a, Z_1, \epsilon, \beta)$ pour chaque β tel que $Z_1 \longrightarrow \beta$ est une règle de G (on doit examiner tout ce qu'on peut dériver depuis β). Alors l'hypothèse de récurrence s'appliquant, entraîne que l'algorithme produira un objet de la forme $(a, b, Z_1, \beta, \epsilon)$: c'est à dire que $Z_1 \Rightarrow^* w[a+1..b]$. L'étape de complétion combinera cet objet pour $j = b$ avec $(a, a, Y, \epsilon, Z_1 \dots Z_n)$ pour donner $(a, c, Y Z_1, Z_2 \dots Z_n)$.

Les mêmes choses se reproduisent pour Z_2, Z_3, jusqu'à Z_n, ce qui donne à la fin un objet $(a, d, Y, Z_1 \dots Z_n, \epsilon)$. Ce qui nous donne l'hypothèse de récurrence et conclut la preuve. □

Remarque 4. Cet algorithme tourne dans le pire cas en temps $O(n^3)$. Il est pourtant possible de prouver que dans le pire cas pour une grammaire non ambigüe, il tourne en temps $O(n^2)$ et même en temps $O(n)$ si la grammaire est rationnelle, où n est la taille du mot w en entrée.