Modèles de Langage et Analyse Syntaxique

Cours 3 - Algorithmes d'analyse syntaxique - grammaires hors-contexte

Antoine Rozenknop
antoine.rozenknop@lipn.univ-paris13.fr

21 octobre 2010

Plan

1 Algorithme Cock -Younger-Kasami **1**
1.1 Principes de l'algorithme CYK 1
1.2 Complexité pire cas 2

2 Algorithme de Earley **3**
2.1 Fondements 3
2.2 Interprétation 4
2.3 Un exemple très simple 4

1 Algorithme Cock -Younger-Kasami

1.1 Principes de l'algorithme CYK

L'algorithme CYK est un algorithme d'analyse syntaxique conçu pour les grammaires hors-contexte (type 2) sous forme normale (CNF).

Comme il est courant pour les algorithmes d'analyse syntaxique, l'algorithme CYK calcule *toutes les interprétations syntaxiques possibles* de *toutes les sous-séquences* de la séquence qu'on lui donne en entrée.

L'efficacité du calcul est fondée sur la propriété suivante :

> Si la grammaire est sous forme normale, le calcul des interprétations d'une séquence W de longueur l nécecessite seulement l'exploration de toutes les décompositions de W en *deux* sous-séquences exactement.
>
> Le nombre de paires de sous-séquences à explorer pour calculer les interprétations de W est donc $l-1$.

Il s'agit donc d'un algorithme du type « Divide-and-Conquer » des plus classiques !

L'idée de l'algorithme CYK est de garder les *traces* de toutes les analyses de toutes les sous-séquences à l'intérieur d'une table.

Plus précisément, l'analyse syntaxique d'une séquence de n mots $W = w_1 \dots w_n$ est représentée dans une table triangulaire de cellules $C_{i,l}$ $(1 \leq i \leq n,\ 1 \leq l \leq n-i+1)$.

La cellule $C_{i,l}$ contient **tous les non-terminaux pouvant dériver** la sous-séquence $w_i ... w_{i+l-1}$ (la séquence de l mots commençant au i^{e}mot de W).

Le calcul des interprétations syntaxiques se fait ligne par ligne, de bas en haut (c'est-à-dire pour des valeurs croissantes de l). Pour cette raison, on parle d'algorithme de type *ascendant* (*bottom-up* en anglais).

Exemple avec la phrase "le chat mange la souris dans le jardin", et la grammaire :

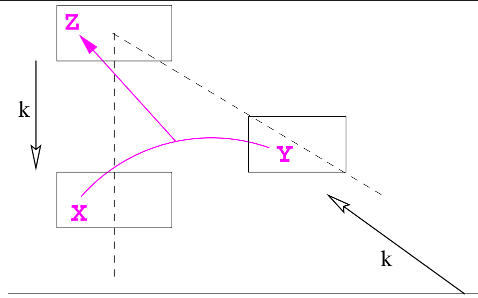
Règles de la grammaire originale				Grammaire sous forme normale			
R1 :	*S*	→	*SN SV*	R1.1 :	*S*	→	*SN SV*
				R1.2 :	*S*	→	*SN V*
R2 :	*SN*	→	*Det N*	R2 :	*SN*	→	*Det N*
R3 :	*SN*	→	*Det N SNP*	R3.1 :	*SN*	→	X_1 *SNP*
				R3.2 :	X_1	→	*Det N*
R4 :	*SNP*	→	*Prep SN*	R4 :	*SNP*	→	*Prep SN*
R5 :	*SV*	→	*V*				
R6 :	*SV*	→	*V SN*	R6 :	*SV*	→	*V SN*
R7 :	*SV*	→	*V SN SNP*	R7.1 :	*SV*	→	X_2 *SNP*
				R7.2 :	X_2	→	*V SN*

Et voici le résultat de l'algorithme :

8	**S**							
7								
6			**SV**,$\mathbf{X_2}$					
5	**S**			**SN**				
4								
3	**S**		**SV**,$\mathbf{X_2}$			**SNP**		
2	**SN**,$\mathbf{X_1}$			**SN**,$\mathbf{X_1}$			**SN**,$\mathbf{X_1}$	
1	**Det**	**N**	**V**	**Det**	**N**	**Prep**	**Det**	**N**
l/i	1	2	3	4	5	6	7	8
	le	chat	mange	la	souris	dans	le	jardin

Formellement, l'algorithme CYK est donné dans la figure suivante :

pour tout $2 \leq i \leq n$ (ligne) **faire**
  pour tout $1 \leq j \leq n - i + 1$ (colonne) **faire**
    pour tout $1 \leq k \leq i - 1$ (décomposition) **faire**
      pour tout $X \in \mathrm{C}[i - k][j]$ **faire**
        pour tout $Y \in \mathrm{C}[k][j + i - k]$ **faire**
          pour tout $Z \rightarrow X\ Y\ \in \mathcal{R}$ **faire**
            Ajouter Z à $\mathrm{C}[i][j]$



On peut aussi implémenter CYK pour une grammaire de type CNF *étendue*. Dans ce cas, chaque fois qu'un non-terminal X est inscrit dans une cellule, il faut aussi y ajouter tous les non-terminaux Y qui se réécrivent en X (c'est-à-dire tels que $Y \rightarrow X$ est une règle de la grammaire).

Une fois que la table a été remplie, on peut retrouver **un** arbre d'analyse si l'on a conservé dans la table, pour chaque non-terminal inscrit, la liste des paires de cellules qui ont mené à son inscription.

1.2 Complexité pire cas

- Le calcul des interprétations d'une cellule $C_{i,j}$ nécessite $(i - 1)$ explorations de paires de cellules $(1 \leq k \leq i - 1)$, le nombre total d'explorations est donc :

$$\sum_{i=2}^{n} \sum_{j=1}^{n-i+1} (i - 1) = \sum_{i=1}^{n} (n - i + 1).(i - 1) \sim \mathcal{O}(n^3).$$

- Une cellule contient au plus $|\mathcal{NT}|$ interprétations (nombre de non-terminaux de la grammaire)
- le coût de l'exploration croisée d'une paire de cellules est d'au plus $|\mathcal{NT}|^2$ accès à la grammaire.

Le coût d'un accès aux règles de grammaires peut être rendu constant avec des techniques efficaces (tables de hachage par exemple). Dans cette configuration, la complexité pire cas de l'analyse d'une séquence de longueur n est alors :

$$\mathcal{O}(n^3 |\mathcal{NT}|^2).$$

On peut voir là un inconvénient de la forme normale : la transformation en CNF augmente le nombre de non-terminaux de $\mathcal{NT}$, donc la complexité de l'analyse. Il existe des versions modifiées de l'algorithme CYK, pour lesquelles la forme normale n'est plus requise, ce qui permet de réduire la complexité dans le

cas général.

Notez qu'il faut faire attention à implémenter correctement les listes de paires de cellules qui mènent à l'introduction d'un non-terminal dans la table. : il est facile de se tromper de telle façon que la complexité devienne $\mathcal{O}(\exp n)$. De fait, si un non-terminal produit dans une cellule est **dupliqué** pour chaque paire de cellule menant à sa production, le nombre d'éléments d'une cellule peut devenir exponentiel, comme illustré par la figure suivante.

Grammaire :
S →S S
S →a
Phrase analysée : a a a a

Implémentation incorrecte

S S S S S			
S S	S S		
S	S	S	
S	S	S	S
a	a	a	a

Implementation correcte

S : •, •, •			
S : •, •	S : •, •		
S : •	S : •	S : •	
S : •	S : •	S : •	S : •
a	a	a	a

2 Algorithme de Earley

Alors que l'analyseur CYK est ascendant (*bottom-up*), l'algorithme de Earley est traité de **descendant** (*top-down*). Il analyse la phrase de gauche à droite, de manière **prédictive**.

Ce second algorithme présente trois avantages :
– il possède la meilleure complexité pire cas connue (de même que CYK) ;
– sa complexité s'adapte à des langages moins complexes (par ex. les langages réguliers) ;
– il ne requière pas une forme particulière de grammaire (pas de CNF).

Il présente également des inconvénients :
– il n'est pas très intuitif ;
– il n'y a pas moyen de corriger / reconstruire des phrases incorrectes syntaxiquement, ni d'en donner des analyses partielles.

2.1 Fondements

Bien que l'algorithme de Earley ne nécessite pas une grammaire sous CNF, cela ne signifie pas qu'il n'y ait aucune binarisation des règles. Mais, alors que cette binarisation doit être précalculée pour l'algorithme CYK, l'algorithme Earley le fait **en ligne**, c'est-à-dire pendant l'analyse. La binarisation est effectuée par l'intermédiaire de **règles à point** (*dotted rules*).

Définition 1. *Une règle à point a la forme :*

$$X \to X_1 \dots X_k \bullet X_{k+1} \dots X_m$$

où $X \to X_1 ... X_m$ *est une règle de la grammaire.*

Ces règles à point sont utilisées dans des **items** d'Earley.

Définition 2. *Un* ***item*** *est constitué :*
– *d'une règle à point ;*
– *d'un entier* i *(avec* $0 \leq i \leq n$*).*

Il signifie que la partie droite de la règle apparaissant avant le point permet de dériver une sous-chaîne de la chaîne initiale ***commençant à la position*** $i+1$.

Par exemple, $(SV \to V \bullet SN, 2)$ et $(SV \to V \bullet SNSNP, 2)$ sont des items pour la chaîne initiale :

le	chat	mangea	la	souris.
1	2	3	4	5

Le troisième concept fondamental est celui des **ensembles** d'items :

Définition 3. *un ensemble d'items E_j est l'ensemble de tous les items de Earley $(X \rightarrow \alpha \bullet \beta, i)$ tels que la partie droite de la règle apparaissant avant le point puisse produire une sous-chaîne de la chaîne initiale* ***se terminant à la position*** *j.*

Par exemple, l'item $(SV \rightarrow V \bullet SN, 2)$ précédent appartient à l'ensemble E_3.

Le principe général de l'algorithme est de démarrer par toutes les règles possibles $(S \rightarrow \bullet X_1 ... X_m, 0)$ et de construire en parallèle toutes les règles à point qui peuvent produire des sous-chaînes de longueurs croissantes de la chaîne initiale, jusqu'à ce que la phrase entière ait été dérivée.

La chaîne initiale, de longueur n, est syntaxiquement correcte *si et seulement si* E_n contient au moins un item $(S \rightarrow X_1 ... X_m \bullet, 0)$.

Le détail est donné dans la figure suivante :

INITIALISATION : construction de E_0 :
pour tout $S \rightarrow X_1 ... X_n \in \mathcal{R}$ **faire**
  ajouter $(S \rightarrow \bullet X_1 ... X_n, 0)$ à E_0.
répéter
  pour tout $(X \rightarrow \bullet Y \beta, 0) \in E_0$ **faire**
    pour tout $Y \rightarrow \gamma \in \mathcal{R}$ **faire**
      ajouter $(Y \rightarrow \bullet \gamma, 0)$ à E_0.
jusqu'à convergence de E_0
ITERATIONS : construction des ensembles E_j $(1 \leq j \leq n)$:
pour $j = 1$ à n **faire**
  pour tout $(X \rightarrow \alpha \bullet w_j \beta, i) \in E_{j-1}$ **faire**
    ajouter $(X \rightarrow \alpha w_j \bullet \beta, i)$ à E_j.
  répéter
    pour tout $(X \rightarrow \gamma \bullet, i) \in E_j$ **faire**
      pour tout $(Y \rightarrow \alpha \bullet X \beta, k) \in E_i$ **faire**
        ajouter $(Y \rightarrow \alpha X \bullet \beta, k)$ à E_j
    pour tout $(X \rightarrow \alpha \bullet Y \beta, i) \in E_j$ **faire**
      pour tout $Y \rightarrow \gamma \in \mathcal{R}$ **faire**
        ajouter $(Y \rightarrow \bullet \gamma, j) \in E_j$.
  jusqu'à convergence de E_j.

2.2 Interprétation

Le remplissage d'un ensemble E_j dans la seconde partie de l'algorithme est une manière de construire les dérivations pour la sous-chaîne d'entrée $w_1 \dots w_j$.

Cette opération se fait en trois temps :

1. **Ancrage vers les mots** :
   c'est l'étape d'initialisation de l'ensemble E_j ; elle consiste à regarder quels sont les items dans E_{j-1} qui peuvent « manger » w_j, c'est-à-dire qui sont compatibles avec l'entrée *jusqu'à* w_j. En déplaçant le point vers la droite, on obtient un item de E_j.
2. **Progression dans l'interprétation** :
   Chaque fois qu'un non-terminal X peut dériver $w_{i+1} \dots w_j$ et qu'il existe un item se finissant en w_i qui peut « manger » X, on peut ajouter un item à E_j en déplaçant le point d'une position vers la droite (ce qui « mange » effectivement X).
3. **Prédiction** (des choses utiles) :
   Cette étape consiste à ajouter de nouveaux « buts » à E_j, c'est-à-dire de nouveaux items avec une partie vide avant le point. Ces nouveaux items sont ceux qui peuvent produire (plus tard) un non-terminal Y qui sera alors mangé par un item de E_j.

Notez que l'initialisation de E_0 est une étape de prédiction, dans laquelle le but initial est S, l'axiome de la grammaire.

2.3 Un exemple très simple

Regardons le processus à l'œuvre lors de l'analyse de la phrase « *Je pense* » avec la grammaire suivante :

S→ SN SV	Pron → *Je*
SN → Pron	V → *pense*
SN → Det N	
SV → V	
SV → V S	
SV → V SN	

L'algorithme remplit successivement les ensembles E_0, E_1 et E_2.

E_0

Prédiction pour le symbole S :

(S →• **SN** SV, 0)

Prédictions induites :

(SN →• **Pron**, 0) (SN→• Det N, 0)

(Pron →• **Je**, 0)

E_1

Liage avec w_1 *(*Je*) :*

(Pron →Je •, 0)

Pas dans la dérivation :

(SN →Pron •, 0) (S→SN• **SV**, 0)

Prédictions :

(SV→• **V**, 1) (SV→• **V** S, 1)

(SV→• **V** SN, 1) (V→• **pense**, 1)

E_2

Liage avec w_2 *(*pense*) :*

(V →pense •, 1)

Pas dans la dérivation :

(SV →V •, 1) (SV →V • **S**, 1)

(SV →V • **SN**, 1) **(S→SN SV •, 0)**

Prédiction :

(S →• **SN** SV, 2) (SN →• **Pron**, 2)

(SN →• Det N, 2) (Pron →• Je , 2)