Algoritmos de Parsing Genéricos

El Análisis CYK

Universidad de Cantabria

Outline

El Problema

En nuestra busqueda por encontrar la estructura exploraremos como hallar “una” derivación.

El Problema

Queremos que sea aplicable a cualquier lenguaje, que este dado por una gramática libre de contexto.

El Problema

También queremos que sea “rápido” y que no consuma muchos recursos (recordad que con una gramática $\lambda-$libre esto se puede hacer utilizando busquedas).

El Problema

Aquí utilizaremos una gramática en forma normal de Chomsky, ya que esta tiene mejores propiedades a la hora de calcular la eficiencia.

Idea

La idea es explorar todas las posibles formas de derivar partes de la palabra

La Tabla CYK

El input del algoritmo está formado por una gramática libre de contexto $G = (V, \Sigma, q_0, P)$, y por una forma terminal, es decir, $\omega := a_1 a_2 \cdots a_{n-1} a_n \in \Sigma^*$.

La Tabla CYK

El output del algoritmo es una tabla triangular. Para interpretarlo, podemos entender la tabla de output como una aplicación:

$$t : \{(i,j) \ : 1 \leq i \leq n, 1 \leq j \leq n-i+1\} \longrightarrow \mathcal{P}(V),$$

donde V son los símbolos no terminales de la gramática y $\mathcal{P}(V)$ son los subconjuntos de V. A la imagen $t(i,j) \in \mathcal{P}(V)$ la denotaremos con sub-índices. Es decir, escribiremos $t_{i,j} \in \mathcal{P}(V)$.

La Tabla CYK

En el conjunto $t_{i,j}$ escribiremos todas las variables $A \in V$ tales que

$$A \vdash^G a_i a_{i+1} \cdots a_{i+j-1}.$$

Nótese que, en realidad, t depende de G y de ω. También hay que hacer notar que el algoritmo necesitará una cantidad de memoria de $O(n^2)$.

La Tabla CYK

En el conjunto $t_{i,j}$ escribiremos todas las variables $A \in V$ tales que

$$A \vdash^{G} a_i a_{i+1} \cdots a_{i+j-1}.$$

Nótese que, en realidad, t depende de G y de ω. También hay que hacer notar que el algoritmo necesitará una cantidad de memoria de $O(n^2)$.

La Tabla CYK: Algoritmo

Inicializar: Para i, $1 \leq i \leq n$,
$t_{i,1} := \{A \ : \ A \mapsto a_i \in P\}$.
Para $j = 1$ **hasta** n **hacer**
    Para $i = 1$ **hasta** $n - j + 1$ **hacer**
        $t_{i,j} := \{A \ : \ \exists k, \ 1 \leq k < j, \ B \in t_{i,k}, \ C \in t_{i+k,j-k},$
        and $A \mapsto BC \in P\}$.
        siguiente i
    fin hacer
    siguiente j
fin hacer
fin

Ejemplo de Algoritmo CYK

Hallar la tabla para la gramática G cuyas producciones son:

$$Q_0 \mapsto AA \mid AQ_0 \mid b,\ A \mapsto Q_0A \mid AQ_0 \mid a.$$

y la palabra $\omega = abaab$.

Ejemplo de Algoritmo CYK

$Q_0 \mapsto AA \mid AQ_0 \mid b,\ A \mapsto Q_0A \mid AQ_0 \mid a, \omega = abaab.$

Aquí está la tabla CYK:

$t_{i,j}$	1	2	3	4	5
1	A	Q_0, A	Q_0, A	Q_0, A	Q_0, A
2	Q_0	A	Q_0	Q_0, A	
3	A	Q_0	Q_0, A		
4	A	Q_0,A			
5	Q_0				

Complejidad del Algoritmo

Theorem

El algoritmo calcula la tabla de análisis sintáctico CYK en tiempo polinomial. Mas concretamente, el tiempo de ejecución del algoritmo es $O(n^3)$ y la memoria usada es $O(n^2)$.

El Problema de la Palabra

Theorem

$\omega \in L(G)$ si y solamente si $Q_0 \in t_{1,n}$. En particular, el cálculo de la tabla por el método CYK resuelve el problema de palabra para gramáticas en forma normal de Chomsky y $\lambda-$libres en tiempo $O(n^3)$ y espacio $O(n^2)$.

Árbol de Derivación

Nuestra pregunta es, ahora que podemos resolver el problema de la palabra queremos dar un algoritmo que devuelva el árbol de derivación de la palabra con esa gramática.

Árbol de Derivación

Lo que haremos será definir una serie de aplicaciones: $gen(i, j, -)$ definidas en los subconjuntos $t_{i,j}$ de la tabla construida a partir del algoritmo CYK antes definido.
La idea es que $gen(i, j, -)$ de una derivación a la derecha de la subpalabra de ω desde la posición i con longitud j.

Observaciones

Una dificultad es que no es aplicación puesto que podría haber más de una producción $A \mapsto BC$ con las propiedades prescritas.

Observaciones

Para evitarlo, se introduce una enumeración del conjunto de las producciones. Así, podremos hablar de la producción $m-$ésima como el elemento de P que ocupa el lugar m.

Observaciones

Si $A \in t_{i,j}$, consideremos todas las producciones de P tales que existe un k, $1 \leq k < j$ y la producción $A \mapsto BC$ con $B \in t_{i,k}$, $C \in t_{i+k,j-k}$. Sea m el mínimo de las enumeraciones de tales producciones. Entonces, definiremos

$$gen(i, j, A) = [m, gen(i, k, B), gen(i + k, j - k, C)].$$

Algoritmo

Si $Q_0 \notin t_{1,n}$ **ERROR**

en otro caso

$gen(i, 1, A) := [m]$ si m es la producción $A \mapsto a_i$.

Si $j > 1$, hallar la producción de numeración más pequeña m tal que:

* Existe k, $1 \leq k < j$ tal que:
* Existen $B \in t_{i,k}$ y $C \in t_{i+k,j-k}$ tales que:
* La producción $m-$ésima es $A \mapsto BC$.

$gen(i, j, A) := [m, gen(i, k, B), gen(i + k, j - k, C)]$

fin si

Devolver $gen(1, n, Q_0)$.

Observaciones

- El algoritmo es un algoritmo de parsing ascendente.
- Este algoritmo se puede usar para resolver el problema de palabra.
- La memoria utilizada solo aumenta marginalmente, es decir, hallar el árbol de derivación requiere reservar $O(n^2)$.
- La complejidad en tiempo de cálculo tampoco aumenta.
- El problema con este algoritmo es que requiere una gramática en forma normal de Chomsky.