Análisis de cadenas en gramáticas incontextuales

Para la realización de la práctica 4 de la asignatura Teoría de Lenguajes hemos decidido extender la clase GIC usada en la práctica anterior. Para ello, hemos añadido un nuevo método, llamado "accepts", que nos devuelve un booleano indicando si la cadena pasada por parámetro es aceptada o no por la gramática.

El método *accepts*

Este es el método que implementa el algoritmo CYK, y que por tanto nos dice si una cadena pertenece a la gramática o no. Para llamar a este método, la gramática tiene que estar en Forma Normal de Chomsky (FNC) [1]. Si esto no es así, el resultado de la llamada a este método es un comportamiento indefinido.

El algoritmo CYK es un algoritmo basado en la programación dinámica. Lo que hace es declarar una matriz de $n \times n$, donde n es la longitud de la cadena a analizar, y la rellena. En la celda (i, j) estarán los símbolos auxiliares que nos permiten generar la subcadena $x_i x_{i+1} .. x_{i+j-1}$, es decir, la subcadena cuyo primer símbolo es el x_i y tiene longitud j.

Lo primero que vemos es que habrá celdas que siempre permanecerán vacías, como por ejemplo la $(n-1, 5)$. De hecho, en vez de declarar una matriz $n \times n$ podríamos declarar una matriz triangular inferior. Esto nos permite reducir a la mitad el consumo de memoria, pero no afecta al consumo asintótico.

Decidiremos que la cadena es generada por la gramática si y sólo si en la celda $(1, n)$ está presente el símbolo inicial, es decir, que con el símbolo inicial podemos generar $x_1 x_2 .. x_n = x$.

El proceso de relleno de la matriz se puede hacer eficientemente precisamente por estar la gramática en FNC.

El tiempo para determinar si un símbolo auxiliar genera una cadena de longitud 1 (todas las celdas de la forma $(i, 1)$) es constante, ya que basta mirar las producciones de ese mismo símbolo.

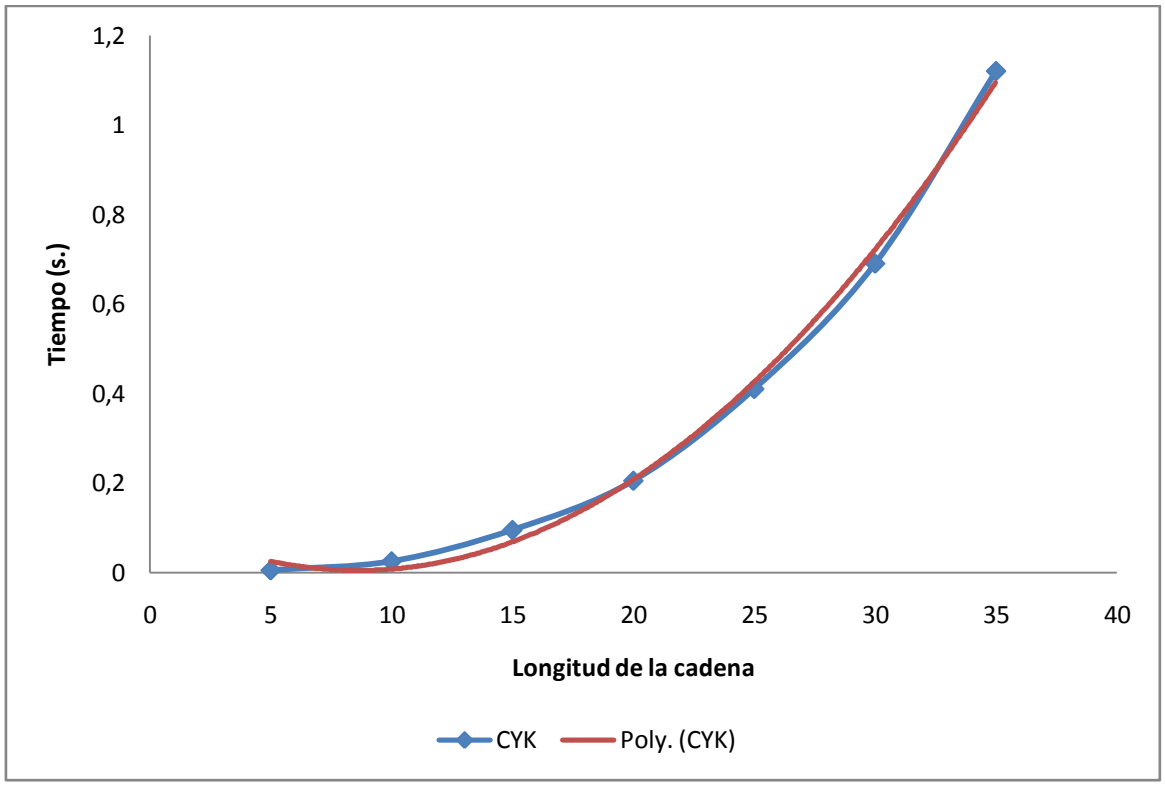
Para determinar si un símbolo auxiliar genera una cadena de longitud j mayor que 1 nos apoyaremos sobre los datos ya hemos introducido previamente en la matriz. En este caso, el tiempo para determinarlo es lineal respecto a j.

Esto queda patente con el siguiente ejemplo. Imaginemos que queremos determinar si el símbolo auxiliar A puede generar una cadena de longitud 4, y que una de las producciones de A es $A \rightarrow BC$. Pudiera ser que B generara el primer símbolo de la cadena y C los 3 restantes, pero también pudiera ocurrir que B generara los 2 primeros símbolos y C los 2 últimos, o incluso que B generara los 3 primeros y C el último. Por lo tanto, el coste de determinar si una producción genera una cadena de longitud j es de $j - 1$.

Medidas de complejidad de ejecución

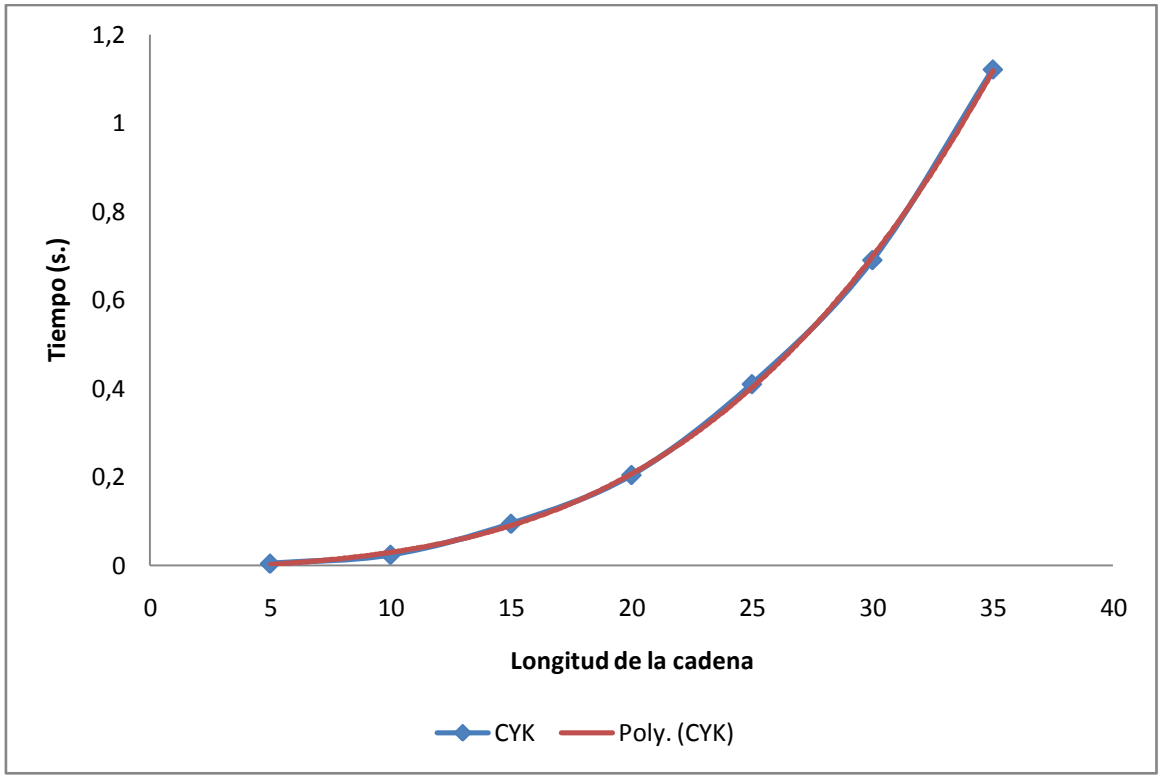
Las medidas de complejidad temporales teóricas nos dicen que el algoritmo CYK tiene un coste cúbico respecto a la longitud de la cadena a analizar, siempre y cuando consideremos la gramática como un parámetro externo [2].

Vamos a comprobar si los resultados teóricos se corresponden con los prácticos. La siguiente gráfica muestra los datos empíricos del coste temporal (en azul) y la función cuadrática que mejor se ajusta a los datos (en rojo):

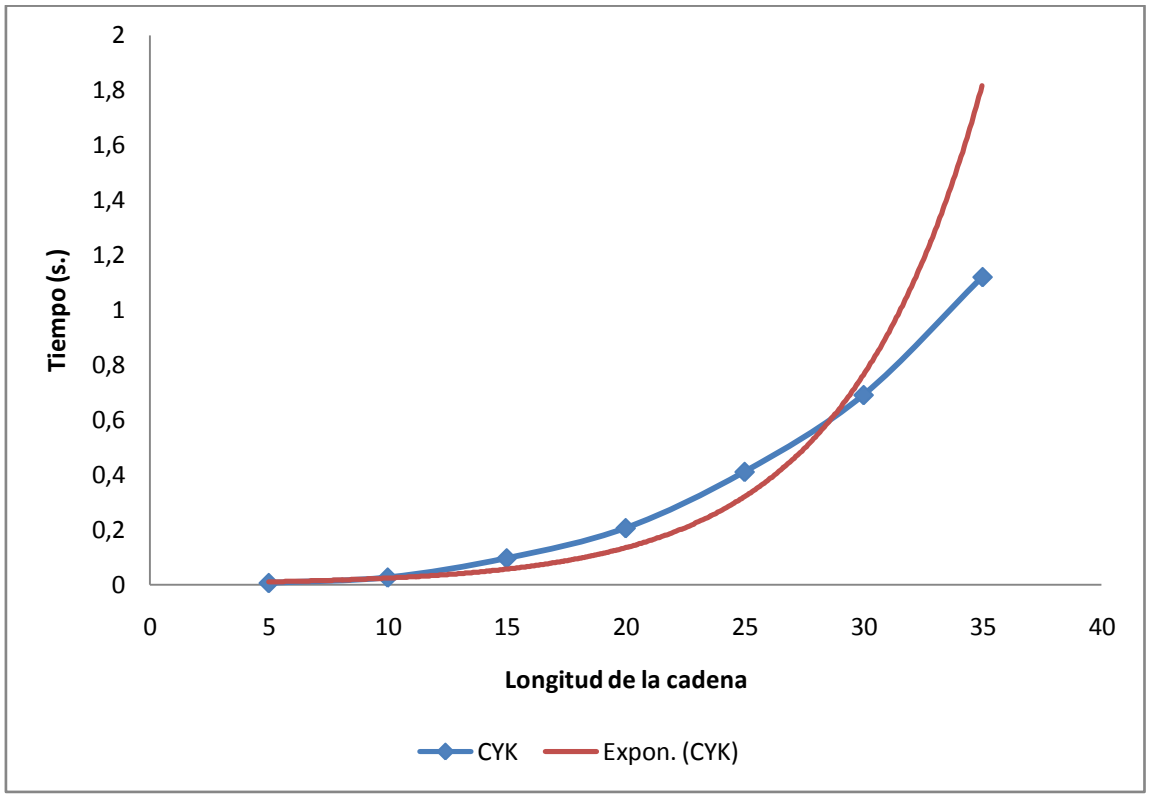


Se puede observar que aunque el ajuste es bastante decente, existen puntos en los que la función cuadrática se aleja de los datos empíricos, sobre todo en 10, 15 y 30.

Vamos a hacer la misma comparación con la función cúbica que mejor se ajusta a los datos:



En este caso se ve claramente que el ajuste de la recta a los datos empíricos es prácticamente perfecto. Sin embargo, para despejar cualquier duda, vamos a comparar también los datos empíricos con la función exponencial que mejor se ajusta a estos datos:



Es obvio, por tanto, que las medidas teóricas de la complejidad temporal del algoritmo CYK coinciden plenamente con los resultados empíricos.

Algunos ejemplos de ejecución

Ejemplo 1

$N = \{S, A, B\}$ $\Sigma = \{a, b\}$

$P = \{S \to a \mid b \mid AB, A \to a \mid AA, B \to b \mid BB\}$

El algoritmo determina que:

$aaab \in L(G)$

$a \in L(G)$

$b \in L(G)$

$aaaaaaaaabbbbbb \in L(G)$

$bbbbbbbbbaaaaa \notin L(G)$

$aba \notin L(G)$

Ejemplo 2

$N = \{S, A, B, D, E\}$ $\Sigma = \{a, b, c\}$

$P = \{S \to DA \mid EB \mid c, A \to a, B \to b, D \to AS, E \to BS\}$

El algoritmo determina que:

$bcb \in L(G)$

$aca \in L(G)$

$babbababacababababbab \in L(G)$

$bbbacabbb \in L(G)$

$c \in L(G)$

$aacbb \notin L(G)$

$abba \notin L(G)$

$abccba \notin L(G)$

Referencias

[1] D. Younger: *Recognition and parsing of context-free languages in time* n^3. Information and Control 10(2): pp.189–208., 1967

[2] J.M. Sempere: *Gramáticas incontextuales*. Apuntes de "Teoría de Lenguajes", 5º curso de la Facultad de Informática de Valencia, Universidad Politécnica de Valencia, 2008