Universidad Autónoma del Estado de México
Centro Universitario Valle de Chalco

Teoría de la computación

William Cruz-Santos

Maestría en Ciencias de la Computación
Contenido básico

26 de septiembre de 2018

Agenda

1. Gramáticas libre de contexto
2. Autómatas de pila
3. Autómatas de pila deterministas
   - Propiedades de las gramáticas libre de contexto
4. Crecimiento de funciones

Agenda

1. Gramáticas libre de contexto
2. Autómatas de pila
3. Autómatas de pila deterministas
   - Propiedades de las gramáticas libre de contexto
4. Crecimiento de funciones

Gramáticas libre de contexto

Una gramática libre de contexto (CFG) es una 4-tupla $G = (V, T, P, S)$ donde

- V es el conjunto de variables,
- T conjunto de terminales,
- P conjunto de producciones y
- S símbolo inicial.

Ejemplo: CFG para palíndromes $G_{pal} = (\{P\}, \{0, 1\}, A, P)$ donde A es el siguiente conjunto de producciones

$$\begin{aligned} P &\to \epsilon \\ P &\to 0 \\ P &\to 1 \\ P &\to 0P0 \\ P &\to 1P1 \end{aligned}$$

Derivaciones

Gramática para expresiones simples:

$$\begin{aligned} E &\rightarrow I \mid E+E \mid E*E \mid (E) \\ I &\rightarrow a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \end{aligned}$$

La cadena $a*(a+b00)$ se puede obtener por medio de derivaciones como sigue:

$$\begin{aligned} &E \Rightarrow E*E \Rightarrow I*E \Rightarrow a*E \Rightarrow \\ &a*(E) \Rightarrow a*(E+E) \Rightarrow a*(I+E) \Rightarrow a*(a+E) \Rightarrow \\ &a*(a+I) \Rightarrow a*(a+I0) \Rightarrow a*(a+I00) \Rightarrow a*(a+b00) \end{aligned}$$

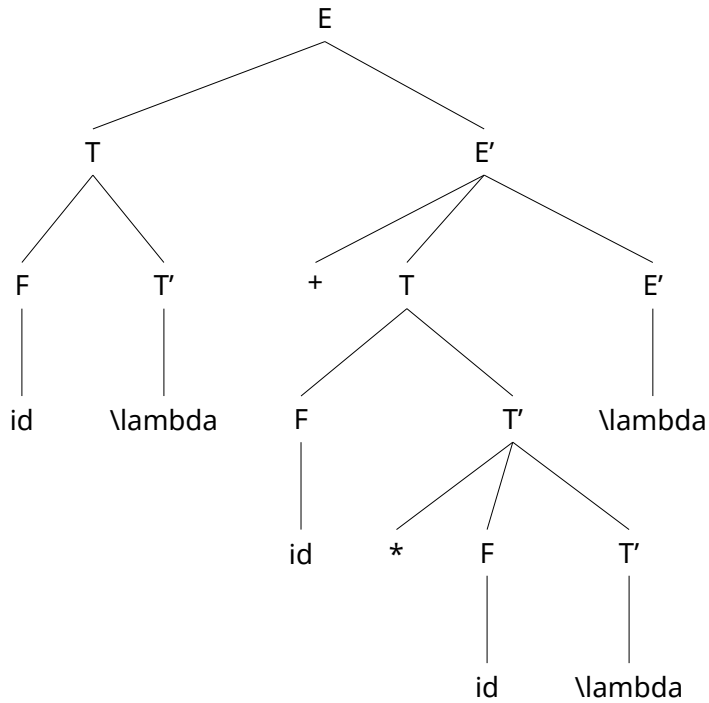
Se usa la notación $\overset{*}{\Rightarrow}$ para condensar una seria de producciones. Por ejemplo,

$$E \overset{*}{\Rightarrow} a*(a+b00)$$

Ejemplo de una derivación

Gramática:

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow +\ T\ E' \mid \lambda \\
T &\rightarrow F\ T' \\
T' &\rightarrow *\ F\ T' \mid \lambda \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
$$



Cadena: $w = \mathbf{id} + \mathbf{id} * \mathbf{id}$
Derivacion:

$$
\begin{aligned}
E &\Rightarrow TE' \Rightarrow FT'E' \Rightarrow \mathbf{id}T'E' \Rightarrow \mathbf{id}E' \\
&\Rightarrow \mathbf{id} + TE' \Rightarrow \mathbf{id} + FT'E' \Rightarrow \mathbf{id} + \mathbf{id}T'E' \\
&\Rightarrow \mathbf{id} + \mathbf{id} * FT'E' \Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}T'E' \\
&\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}E' \\
&\Rightarrow \mathbf{id} + \mathbf{id} * \mathbf{id}
\end{aligned}
$$

Lenguaje de una gramática

Si $G = (V, T, P, S)$ es una CFG, el lenguaje de G, denotado por $L(G)$, es el conjunto de cadenas terminales que tienen una derivación a partir del símbolo inicial S. Es decir

$$L(G) = \{w \text{ en } T^* \mid S \stackrel{*}{\Rightarrow} w\}.$$

Por ejemplo, $L(G_{pal})$ es el lenguaje de palabras sobre $\{0, 1\}$ palíndromes.

Ejercicios:

- Proporcione una CFG para el lenguaje $\{0^n 1^n \mid n > 1\}$.
  Sol: $S \to 0S1 \mid 01$

Ejercicios (cont):

- Proporcione una CFG para el lenguaje $\{a^i b^j c^k \mid i \neq j \text{ o } j \neq k\}$.

$$
\begin{aligned}
S &\rightarrow AB \mid CD \\
A &\rightarrow aA \mid \epsilon \\
B &\rightarrow bBc \mid E \mid cD \\
C &\rightarrow aCb \mid E \mid aA \\
D &\rightarrow cD \mid \epsilon \\
E &\rightarrow bE \mid b
\end{aligned}
$$

Derivaciones por la *izquierda* y por la *derecha*:

- En las derivaciones por la izquierda, siempre se elige el no terminal por la izquierda. Si $\alpha \Rightarrow \beta$ es un paso en el que se sustituye el no terminal por la izquierda en α, escribimos $\alpha \underset{lm}{\Rightarrow} \beta$.

- En las derivaciones por la derecha, siempre se elige el no terminal por la derecha; en este caso escribimos $\alpha \underset{rm}{\Rightarrow} \beta$.

Para la CFG

$$E \to E + E \mid E * E \mid -E \mid (E) \mid \mathbf{id}$$

se tiene que

$$E \underset{lm}{\Rightarrow} -E \underset{lm}{\Rightarrow} -(E) \underset{lm}{\Rightarrow} -(E + E) \underset{lm}{\Rightarrow} -(\mathbf{id} + E) \underset{lm}{\Rightarrow} -(\mathbf{id} + \mathbf{id})$$

CFG's y expresiones regulares

Considere la siguiente expresión regular $(0 \mid 1)^*011$ y la siguiente gramática:

$$\begin{aligned} A_0 &\rightarrow 0A_0 \mid 1A_0 \mid 0A_1 \\ A_1 &\rightarrow 1A_2 \\ A_2 &\rightarrow 1A_3 \\ A_3 &\rightarrow \lambda \end{aligned}$$

describen el mismo lenguaje, el conjunto de cadenas de 0's y 1's que terminan en 011.

Dado un AFN $N = (Q, q_0, F, \delta, \Sigma)$, se puede construir una gramática libre de contexto G_N tal que $L(N) = L(G_N)$.

Procedimiento:

1. Para cada estado i en N, crear un no terminal A_i.

2. Si el estado i tiene una transición al estado j con la entrada a, agregar la producción $A_i \to aA_j$. Si el estado i pasa al estado j con la entrada λ, agregar la producción $A_i \to A_j$.

3. Si i es un estado de aceptación, agregar $A_i \to \lambda$.

4. Si i es el estado inicial, hacer que A_i sea el símbolo inicial de la gramática.

Recursividad

Una producción es *recursiva* por la izquierda si tiene la siguiente forma:

$$A \to A\alpha \mid \beta$$

Por ejemplo, la producción

$$exp \to exp + term \mid exp - term \mid term$$

La recursividad por la izquierda se puede eliminar usando el siguiente procedimiento:

$$\begin{aligned} A &\to \beta A' \\ A' &\to \alpha A' \mid \lambda \end{aligned}$$

donde A' es un nuevo símbolo no terminal.

Considere la regla recursiva de la gramática de expresión simple:

$$exp \rightarrow exp\ opsuma\ term \mid term$$

La forma de la regla es $A \rightarrow A\alpha \mid \beta$, con $A = exp$, $\alpha = opsuma\ term$ y $\beta = term$. Al volverla a escribir para eliminar la recursión por la izquierda obtenemos que

$$\begin{aligned} exp &\rightarrow term\ exp' \\ exp' &\rightarrow opsuma\ term\ exp' \mid \lambda \end{aligned}$$

Cuando se tienen producciones de la forma:

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m$$

donde ninguna de las $\beta_1, \ldots, \beta_m$ comienzan con A. En este caso la solución es semejante a la del caso simple, con las opciones extendidas en consecuencia:

$$\begin{aligned} A &\to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_m A' \\ A' &\to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_n A' \mid \lambda \end{aligned}$$

Por ejemplo, considere la regla gramatical

$$exp \to exp + term \mid exp - term \mid term$$

Eliminemos la recursión por la izquierda de la manera siguiente:

$$\begin{aligned} exp &\to term\ exp' \\ exp' &\to term\ exp' \mid -term\ exp' \mid \lambda \end{aligned}$$

La *factorización* por la izquierda se requiere cuando dos o más opciones de reglas gramaticales comparten una cadena de prefijo común, como en la regla:

$$A \to \alpha\ \beta \mid \alpha\ \gamma$$

Por ejemplo, para secuencias de sentencias

$$\begin{aligned} \textit{secuencia-sent} &\to \textit{sent}\ ;\ \textit{secuencia-sent} \mid \textit{sent} \\ \textit{sent} &\to \mathbf{S} \end{aligned}$$

La solución es factorizar la α por la izquierda y volver a escribir la regla como dos reglas:

$$\begin{aligned} A &\to \alpha A' \\ A' &\to \beta \mid \gamma \end{aligned}$$

La regla gramatical para las secuencias de sentencias escrita en forma recursiva por la derecha:

$$\begin{aligned} \textit{secuencia-sent} &\rightarrow \textit{sent} \ ; \ \textit{secuencia-sent} \mid \textit{sent} \\ \textit{sent} &\rightarrow \mathbf{S} \end{aligned}$$

que se puede factorizar por la izquierda de la manera siguiente

$$\begin{aligned} \textit{secuencia-sent} &\rightarrow \textit{sent} \ \textit{sec-sent}' \\ \textit{sec-sent}' &\rightarrow \ ; \ \textit{secuencia-sent} \mid \lambda \end{aligned}$$

Agenda

1. Gramáticas libre de contexto
2. Autómatas de pila
3. Autómatas de pila deterministas
   - Propiedades de las gramáticas libre de contexto
4. Crecimiento de funciones

Un autómata de pila (PDA) P se define como

$$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

donde

- Q: conjunto finito de estados
- Σ: conjunto finito de símbolos o alfabeto
- Γ: conjunto finito de símbolos de pila
- δ: función de transición
- q_0: estado inicial
- Z_0: símbolo inicial de pila
- F: conjunto de estados terminales

En cada iteración de un PDA, su configuración instantánea consiste de una tercia (q, w, γ) donde

- q es un estado
- w es el entrada que falta por procesar
- γ es el contenido de la pila

De manera formal, sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Suponga que $\delta(q, a, X)$ contiene (p, α). Entonces para todas las cadenas $w \in \Sigma^*$ y $\beta \in \Gamma^*$:

$$(q, aw, X\beta) \vdash (p, w, \alpha\beta)$$

Se usa el símbolo $\vdash^*$ para representar cero o mas movimientos de un PDA.

El lenguaje de un PDA P que *acepta por estado terminal* se define por

$$L(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \alpha)\}$$

para algún esta q en F y cualquier cadena $\alpha \in \Gamma^*$.

De igual forma, se define

$$N(P) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$$

como el lenguaje de P *aceptado por pila vacía.*

Sea $G = (V, T, Q, S)$ una gramática libre de contexto (CFG). Entonces, existe un PDA P tal que $N(P) = L(G)$.

Construcción: defina $P = (\{q\}, T \cup V, \delta, q, S)$ tal que

1. Para cada variable A,

$$\delta(q, \epsilon, A) = \{(q, \beta) \mid A \to \beta \text{ es una producción de } G\}$$

2. Para cada terminal a,

$$\delta(q, a, a) = \{(q, \epsilon)\}$$

Sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un PDA. Entonces, existe una CFG G tal que $L(G) = N(P)$.

Construcción: defina $G = (V, \Sigma, R, S)$ tal que

1. El conjunto de variables V consiste del símbolo inicial S y los símbolos de la forma $[pXq]$ donde $p, q \in Q$ y $X \in \Gamma$.
2. Las producciones de G son como sigue:
   1. Para todo $p \in Q$: G tiene la producción $S \to [q_0 Z_0 p]$.
   2. Para todas las listas de estados $r_1, r_2, \ldots, r_k$, G tiene la producción

$$[qXr_k] \to a[rY_1r_1][r_1Y_2r_2]\cdots[r_{k-1}Y_kr_k]$$

   siempre que $\delta(q, a, X)$ contenga $(r, Y_1Y_2\cdots Y_k)$.

Agenda

1. Gramáticas libre de contexto
2. Autómatas de pila
3. Autómatas de pila deterministas
   - Propiedades de las gramáticas libre de contexto
4. Crecimiento de funciones

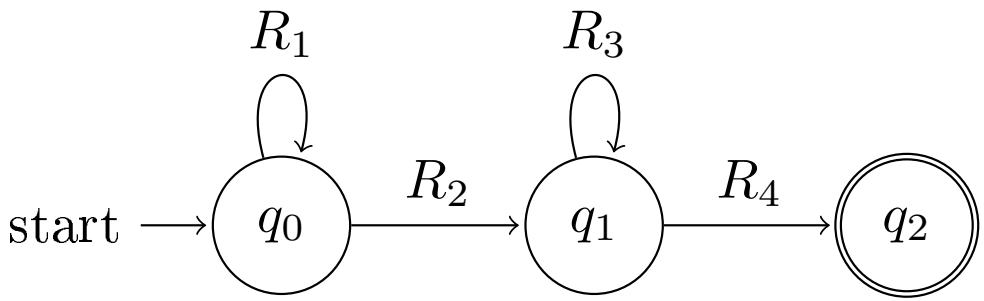
Un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ es determinista si y sólo si se cumplen las siguientes condiciones:

1. Para todo $q \in Q$, $a \in \Sigma \cup \{\epsilon\}$, $X \in \Gamma$: $\delta(q, a, X)$ tiene a lo más un elemento.
2. Si $\delta(q, a, X)$ es no-vacío para algún $a \in \Sigma$, entonces $\delta(q, \epsilon, X)$ debe ser vacío.

Los PDA's deterministas son usados principalmente como *parsers* en la construcción de lenguajes de programación.

PDA determinista que reconoce el lenguaje

$$L_{wcwr} = \{wcw^R \mid w \text{ esta en } (\mathbf{0}+\mathbf{1})^*\}$$



donde

R_1 :

$0, Z_0/0Z_0$

$1, Z_0/1Z_0$

$0, 0/00$

$0, 1/01$

$1, 0/10$

$1, 1/11$

R_2 :

$c, Z_0/Z_0$

$c, 0/0$

$c, 1/1$

R_3 :

$0, 0/\epsilon$

$1, 1/\epsilon$

R_4 :

$\epsilon, Z_0/Z_0$

Si L es un lenguaje regular, entonces $L = L(P)$ para algún DPDA P.

Esencialmente, un DPDA puede simular un DFA, ignorando la pila y usando sus estados.

Simulación:
Sea $A = (Q, \Sigma, \delta_A, q_0, F)$ un DFA. Se construye un DPDA

$$P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$$

donde $\delta_P(q, a, Z_0) = \{(p, Z_0)\}$ para todo $p, q \in Q$ tal que $\delta_A(q, a) = p$

Entonces, P simula A, siempre que

$$(q_0, w, Z_0) \vdash_P^* (p, \epsilon, Z_0) \text{ si y sólo si } \hat{\delta}_A(q_0, w) = p.$$

Conclusiones:

- DPDA's reconocen lenguajes regulares.
- El lenguaje L_{wcwr} no es regular y se puede reconocer por un DPDA.
- El conjunto de lenguajes regulares es un subconjunto propio del conjunto de lenguajes libres de contexto.

Preguntas de investigación:

Sea Σ un alfabeto y sea $PDA(\Sigma)$ el conjunto de todos los autómatas de pila sobre el alfabeto Σ. ¿Existe un alfabeto S y una función $f : PDA(\Sigma) \to S^*$ tal que el conjunto $\{(f(A), w) : w \in L(A)\}$ es aceptado por un PDA no-determinista?

Ver Manfred Kudlek, On the Existence of Universal Finite or Pushdown Automata, arXiv:1207.7149

Agenda

1. Gramáticas libre de contexto
2. Autómatas de pila
3. Autómatas de pila deterministas
   - Propiedades de las gramáticas libre de contexto
4. Crecimiento de funciones

Teorema (Pumping lemma)

Si A es un lenguaje libre de contexto, entonces existe un número p donde para cada cadena s en A de longitud al menos p, entonces s se puede dividir en cinco piezas como $s = uvxyz$ tal que se satisfacen las siguientes condiciones:

1. Para cada $i \geq 0$, $uv^i xy^i z \in A$
2. $|vy| > 0$, y
3. $|vxy| \leq p$.

Este teorema se usa comúnmente para demostrar que ciertos lenguajes no son libres de contexto.

Ejercicio 1: Demuestre que el lenguaje $L_{01} = \{0^n 1^n \mid n \geq 1\}$ satisface el *pumping lemma* para gramáticas libres de contexto.

Ejercicio 2: Use el *pumping lemma* para mostrar que el lenguaje $L_{abc} = \{a^n b^n c^n \mid n \geq 0\}$ no es libre de contexto.

Ejercicio 3: Use el *pumping lemma* para mostrar que el lenguaje $L_{xyz} = \{x^i y^j z^k \mid 0 \leq i \leq j \leq k\}$ no es libre de contexto.

Agenda

1. Gramáticas libre de contexto
2. Autómatas de pila
3. Autómatas de pila deterministas
   - Propiedades de las gramáticas libre de contexto
4. Crecimiento de funciones

Notación asintótica

Representamos por $T : \mathbb{N} \to \mathbb{Z}^+$ el tiempo de ejecución de un algoritmo.

Para cualquier $n \in \mathbb{N}$, $T(n)$ es el tiempo de ejecución de un algoritmo dado y n es el tamaño del problema.

Usaremos la notación asintótica para describir el tiempo de ejecución de un algoritmo, y también el espacio que ocupa.

Orden de magnitud de un algoritmo

- Suele usarse la notación "O" (big-O)

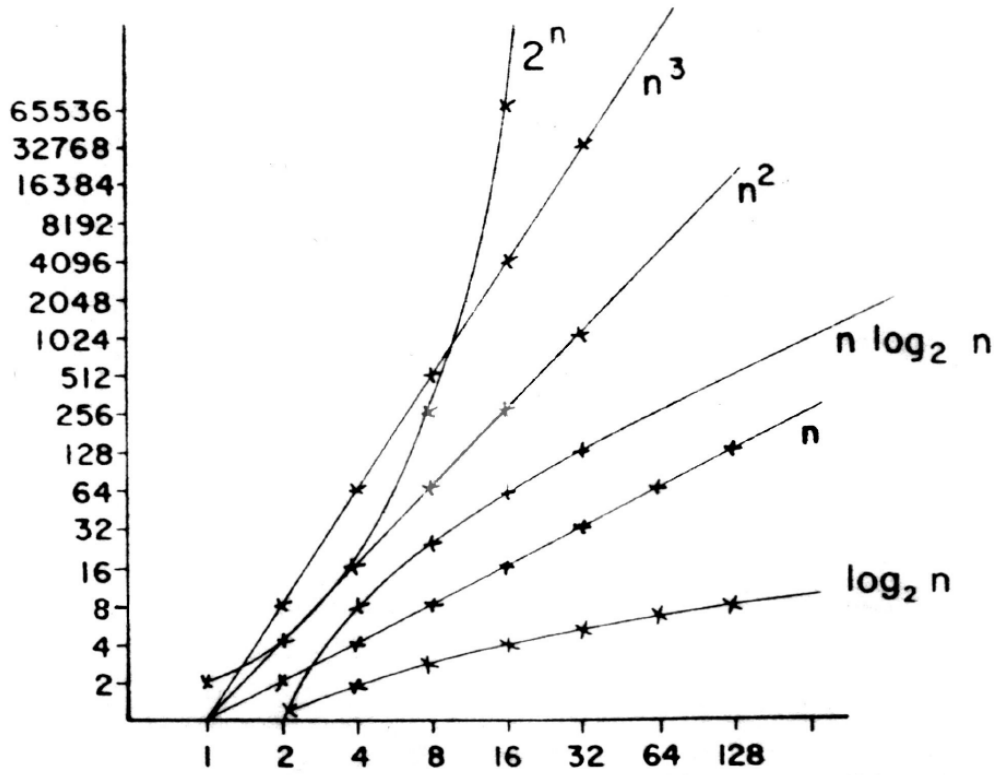
- Si un algoritmo tiene complejidad $O(g(n))$ significa que al correrlo en una computadora con los mismos datos, pero valores incrementales de n, los tiempos resultantes de ejecución serán siempre menores que $|g(n)|$

Orden de magnitud de un algoritmo

Los tiempos más comunes de los algoritmos son:

$$O(1) < O(\log n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n)$$

Orden de magnitud de un algoritmo



Notación-O (1)

Sea $g(n)$ una función, se denota por $O(g(n))$ al conjunto de funciones:

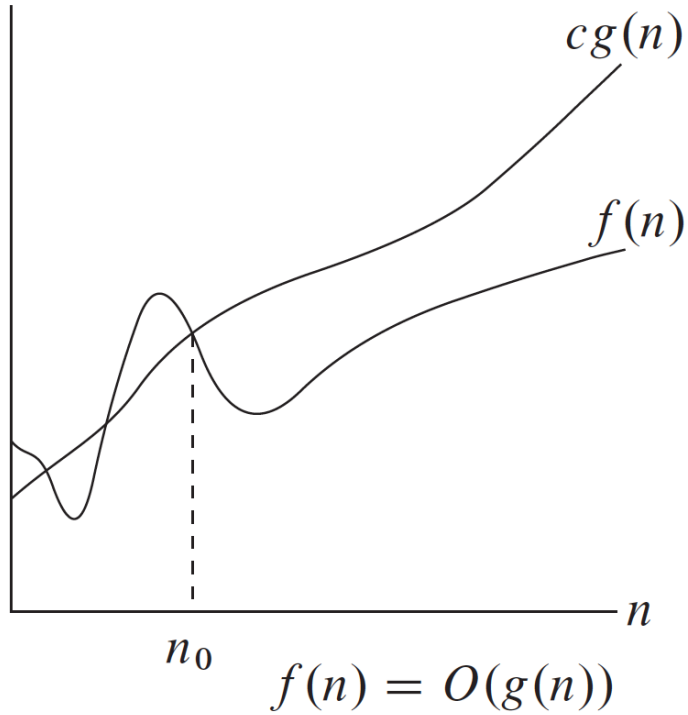
$$O(g(n)) = \{f(n) \mid \text{ existen constantes } c, n_0 \text{ tal que } 0 \leq f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$$

En otras palabras, $f(n)$ está en $O(g(n))$ si existen constantes c, n_0 tal que $cg(n)$ acota superiormente a $f(n)$ para $n \geq n_0$.

La notación O se usa comúnmente para representar el peor caso de corrida de un algoritmo.

Notación-O (2)

- Se dice que $g(n)$ es una *cota superior* para $f(n)$
- Si $f(n) = \Theta(g(n))$ implica que $f(n) = O(g(n))$.
- Θ es una noción más fuerte que O



Notación-Θ (1)

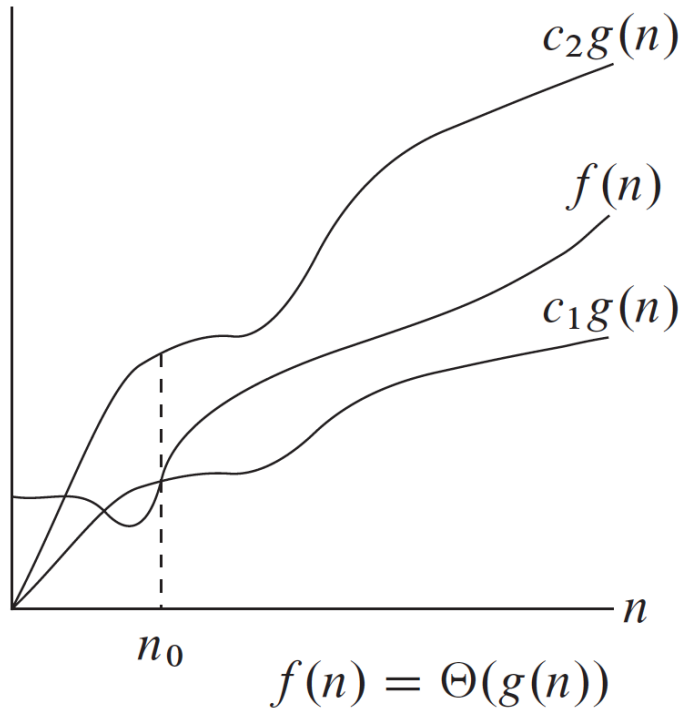
Sea $g(n)$ una función, se denota por $\Theta(g(n))$ al conjunto de funciones:

$$\Theta(g(n)) = \{f(n) \mid \text{ existen constantes } c_1, c_2, n_0 \text{ tal que } 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ para todo } n \geq n_0\}$$

En otras palabras, $f(n)$ está en $\Theta(g(n))$ si existen constantes c_1, c_2 tal que se forme un *sandwich* entre $c_1 g(n)$ y $c_2 g(n)$ para $n \geq n_0$.

Notación-Θ (2)

- Usamos $f(n) = \Theta(g(n))$ para denotar $f(n) \in \Theta(g(n))$
- n_0 es el punto a partir del cual $g(n)$ acota a $f(n)$
- Se dice que $g(n)$ es una *cota estrecha* para $f(n)$



Notación-Θ (3)

Ejemplo:

Demuestre que $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

A demostrar, determine constantes positivas c_1, c_2, n_0 tal que

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n_2, \text{ para todo } n \geq n_0$$

Haciendo a $c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$ y $n_0 = 7$.

Notación-Ω (1)

Sea $g(n)$ una función, se denota por $\Omega(g(n))$ al conjunto de funciones:

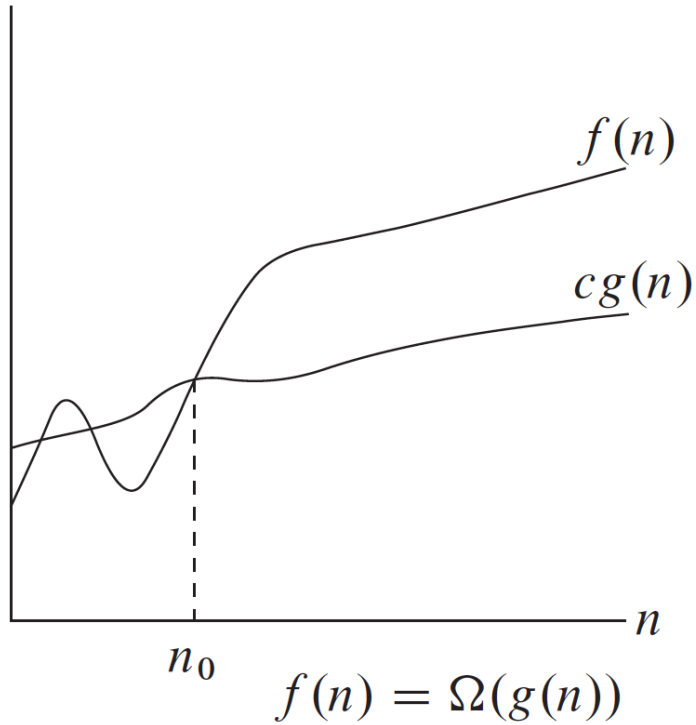
$$\Omega(g(n)) = \{f(n) \mid \text{ existen constantes } c, n_0 \text{ tal que } 0 \leq cg(n) \leq f(n) \text{ para todo } n \geq n_0\}$$

En otras palabras, $f(n)$ está en $\Omega(g(n))$ si existen constantes c, n_0 tal que $cg(n)$ acota inferiormente a $f(n)$ para $n \geq n_0$.

La notación O se usa comúnmente para representar el mejor de los casos de corrida de un algoritmo.

Notación-Ω (2)

- Se dice que $g(n)$ es una *cota inferior* para $f(n)$
- Si $f(n) = \Theta(g(n))$ implica que $f(n) = \Omega(g(n))$.
- Θ es una noción más fuerte que Ω



De las notaciones O y Ω se tiene lo siguiente:

Teorema: para cualesquiera dos funciones $f(n)$ y $g(n)$, se satisface $f(n) = \Theta(g(n))$ si y sólo si $f(n) = O(g(n))$ y $f(n) = \Omega(g(n))$.

Aplicando el teorema anterior si, $an^2 + bn + c = \Theta(g(n))$, implica que $an^2 + bn + c = O(g(n))$ y $an^2 + bn + c = \Omega(g(n))$.

De manera contraria, para demostrar que $f(n) = \Theta(g(n))$ *se necesita demostrar que* $f(n) = O(g(n))$ *y que* $f(n) = \Omega(g(n))$.

Ejercicios (1)

Demuestre las siguientes relaciones:

1. $f(n) = \Theta(g(n))$ y $g(n) = \Theta(h(n))$ implica $f(n) = \Theta(h(n))$
2. $f(n) = O(g(n))$ y $g(n) = O(h(n))$ implica $f(n) = O(h(n))$
3. $f(n) = \Omega(g(n))$ y $g(n) = \Omega(h(n))$ implica $f(n) = \Omega(h(n))$

Ejercicios (2)

Resuelva los siguientes problemas:

1. Muestre que para cualesquiera constantes reales a y b, donde $b > 0$, $(n + a)^b = \Theta(n^b)$.
2. Determine si $2^{n+1} = O(2^n)$.
3. Determine si $2^{2n} = O(2^n)$.
4. Sean $f(n)$ y $g(n)$ funciones positivas, muestre que $\text{máx}\{f(n), g(n)\} = \Theta(f(n) + g(n))$.

Conclusiones

- En esta presentación se explican conceptos generales sobre autómatas finitos, gramáticas libres de contexto y lenguajes formales.
- Estos temas son la base de la teoría de la computación.
- La fundamentación teórica permite construir modelos formales de computación para proponer soluciones tomando en cuenta la complejidad algoritmica.

Referencias

1. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman, "Introduction to Automata Theory, Languages, and Computation", Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
2. Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman, "Compilers: Principles, Techniques, and Tools", Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.
3. D. Grune, K.V. Reeuwijk, H. Bal, C. Jacobs, K. Langendoen, "Modern Compiler Design", Springer, 2012.
4. R. Wilhelm, "Compiler Design", Addison Wesley, 2011.