Autómatas y lenguajes formales

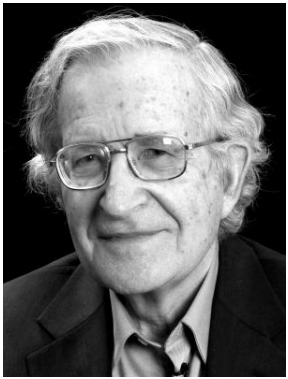
Víctor Mijangos de la Cruz

vmijangosc @ ciencias.unam.com

III. Lenguajes libres de contexto

Lenguajes libres de contextos

Lenguajes libres de contexto



Los Lenguajes Libres de Contexto (*Context-Free Languages* o CFL) son un tipo de lenguajes desarrollados por Noam Chomsky.

- Los CFL han jugado un papel esencial en la tecnología de compiladores.
- Implementan los parser/analizadores sintácticos de un compiladors.
- Definen DTDs (Document-Type Definition) que sirven para el análisis de documentos XML.

Gramática libre de contexto

Gramática libre de contexto

Una gramática libre de contexto (CFG) es una gramática $G = (\Sigma, \Delta, S, R)$ tal que:

- $\Sigma = \{a_1, a_2, ..., a_n\}$ es un conjunto finito de símbolos terminales.
- $\Delta = \{X_1, X_2, ..., X_k\}$ es un conjunto finito de símbolos no-terminales o variables.
- $S \in \Delta$ es el símbolo inicial (variable que define a todo el lenguaje).
- $R \subseteq \Delta \times (\Delta \cup \Sigma)^*$ son las producciones de la gramática, de la forma:

$$X \to \gamma$$

Donde $X \in \Delta$ y $\gamma \in (\Delta \cup \Sigma)^*$.

Producciones

Las producciones o reglas de un CFG cuentan con los siguientes elementos:

- **Cabeza:** Es un único símbolo no-terminal que está a la izquierda de la producción. No puede ser ni cadena, ni un terminal.
- **Cuerpo:** Es una cadena de 0 o más terminales y variables.

Ejemplo: La gramática $G = (\{E, I\}, \{a, b, 0, 1, +, \cdot, (,)\}, E, R)$ dada por:

$$E \to I|E + E|E \cdot E|(E)$$
$$I \to a|b|Ia|Ib|I0|I1$$

Sólo cuenta con símbolos E e I como cabezas y define fórmulas con operaciones '$+$' y '$\cdot$'.

Inferencia recursiva

Lenguaje de variable

Dada una variable $X \in \Delta$ de la gramática, diremos que el lenguaje de esta variable es:

$$L(X) = \{w \in \Sigma^* : X \models w\}$$

Inferencia recursiva

Dado una cadena $w \in \Sigma^*$ podemos ver si pertenece al lenguaje de la gramática si podemos derivarlo de las producciones de la gramática partiendo del inicial:

- Derivación en un paso: $X \to a$ para algún símbolo $a \in \Sigma$, por lo que $a \in L(X)$.
- Derivación en n pasos: Si hay Y_i tal que $w_i \in L(Y_i)$ y $X \to Y_1 Y_2 \cdots Y_k$, entonces $w_1 \cdots w_k \in L(X)$.

Si $w \in L(S) = L(G)$, entonces $G \models w$.

Ejemplo

De la gramática anterior para expresiones con $+$ y $\cdot$, podemos ver que $a \cdot (a + b00)$ es parte del lenguaje.

	Cadena	**Var**	**Regla**	**De**
(i)	a	I	$I \to a$	-
(ii)	b	I	$I \to B$	-
(iii)	$b0$	I	$I \to I0$	(ii)
(iv)	$b00$	I	$I \to I0$	(iii)
(v)	a	E	$E \to I$	(i)
(vi)	$b00$	E	$E \to I$	(iv)
(vii)	$a + b00$	E	$E \to E + E$	(v), (vi)
(viii)	$(a + b00)$	E	$E \to (E)$	(vii)
(ix)	$a \cdot (a + b00)$	E	$E \to E \cdot E$	(v), (viii)

Derivaciones

Derivación

x deriva en y en la gramática G sii $x = \alpha X \beta$, $y = \alpha Y \beta$ y $X \to Y \in R$. Entonces $x \Longrightarrow_G y$

Derivación por la izquierda (derecha)

Una derivación por la izquierda (izquierda) deriva primero la variable de la izquierda (derecha). Se denota $\Longrightarrow_{lm}$ ($\Longrightarrow_{rm}$).

Ejemplo: Una derivación por la izquierda de la cadena $a \cdot (a + b00)$ es:

$$E \Longrightarrow_{lm} E \cdot E \Longrightarrow_{lm} I \cdot E \Longrightarrow_{lm} a \cdot E \Longrightarrow_{lm} a \cdot (E) \Longrightarrow_{lm} a \cdot (E + E) \Longrightarrow_{lm} a \cdot (I + E)$$
$$\Longrightarrow_{lm} a \cdot (a + E) \Longrightarrow_{lm} a \cdot (a + I) \Longrightarrow_{lm} a \cdot (a + I0) \Longrightarrow_{lm} a \cdot (a + I00)$$
$$\Longrightarrow_{lm} a \cdot (a + b00)$$

Extensión de la derivación

Extensión de la derivación

Dada una gramática libre de contexto $G = (\Sigma, \Delta, S, R)$, con derivación $\Longrightarrow_G$, definimos la extensión $\Longrightarrow_G^*$ como:

1. $\forall X \in \Delta \cup \Sigma$, $X \Longrightarrow^* X$.
2. Si $X \Longrightarrow^* Y$ y también $Y \Longrightarrow^* Z$, entonces $X \Longrightarrow^* Z$.

Decimos que X deriva en Z en n pasos.

Ejemplo: De la derivación anterior, entonces podemos concluir que:

$$E \Longrightarrow^* a \cdot (a + I00)$$

Lenguaje de una gramática

Lenguaje de una CFG

Dada una CFG $G = (\Sigma, \Delta, S, R)$, el lenguaje generado por la gramática G se define como:

$$L(G) = \{w \in \Sigma^* : S \Longrightarrow_G^* w\}$$

Ejemplo: Considérese la siguiente gramática dada por las producciones:

$$S \to 0|1|\epsilon$$
$$S \to 0S0|1S1$$

Esta gramática puede producir la cadena 00100 como:

$$S \Longrightarrow 0S0 \Longrightarrow 00S00 \Longrightarrow 00100$$

El lenguaje producido es el de los palíndromos.

Forma sentencial y árboles sintácticos

Forma sentencial

Sea $G = (\Sigma, \Delta, S, R)$ una CFG. Una cadena $\alpha \in (\Sigma \cup \Delta)^*$ es una forma sentencial si se tiene que $S \Longrightarrow^* \alpha$.
Si $S \Longrightarrow^*_{lm} \alpha$ es una forma sentencial izquierda, y si $S \Longrightarrow^*_{rm} \alpha$ se llamará forma sentencial derecha.

Ejemplo: Dada la gramática de las expresiones de suma y producto, tenemos que $E \cdot (E + E)$ es una forma sentencial. Una forma sentencial izquierda es $a \cdot E$, y ubna derecha es $E \cdot (E + E)$. Para la gramática de palíndromos $00S00$ es una forma sentencial.

Árboles sintácticos

Árbol sintáctico

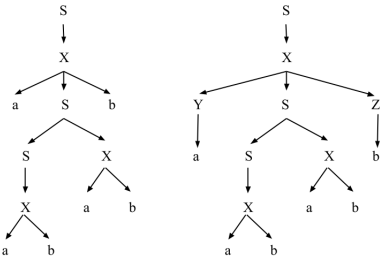
Un árbol sintáctico o de parseo es un árbol ordenado con raíz $T = (E, V)$, tal que sus nodos se asocian a las variables de una CFG $G = (\Sigma, \Delta, S, R)$ condicionados a:

1. El nodo raíz n_0 está etiquetado con el símbolo inicial S.
2. $\forall n \in V$, nodo interior, n está asociado a una variable $X \in \Delta$.
3. $\forall n \in V$, nodo hoja, n está etiquetado por $(\Sigma \cup \Delta) \cup \{\epsilon\}$.
4. Si el nodo está etiquetado con ϵ, entonces n debe ser el único hijo de su padre, y no será padre de ningún nodo.
5. Si $n \in V$ es nodo interior etiquetado con A y sus hijos con $X_1, X_2, ..., X_k$, entonces existe una producción $A \to X_1 X_2 \cdots X_k \in R$.

Ejemplos

Considérese la siguiente gramática libre de contexto dada por las producciones:

$$S \to SX|YSZ|X$$
$$X \to aSb|ab$$
$$Y \to a$$
$$Z \to b$$



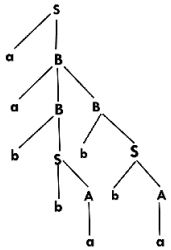
Producción de un árbol

Producción de u árbol

La producción de un árbol es la cadena que resulta de concatenar los nodos hijos a partir de la izquierda. De esta forma, la producción es una cadena terminal, todas las hojas del árbol son o terminales o ϵ.

Ejemplo: Dada la siguiente gramática:

$$S \rightarrow aB|bA$$
$$A \rightarrow a|aS|bAA$$
$$B \rightarrow b|bS|aBB$$



La producción del árbol de la izquierda es

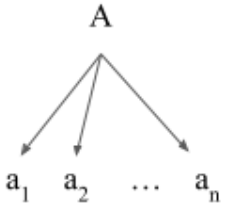
$$aabbabba$$

Equivalencia de derivación y árboles

Teorema

Dada una gramática libre de contexto $G = (\Sigma, \Delta, S, R)$, para $A \in \Delta$, $A \Longrightarrow_G^* w$ si y sólo si hay un árbol T con raíz A que produce w.

Demostraremos por inducción que si $A \Longrightarrow^* w$ entonces hay un árbol que produce w.



Base: Supongamos que w se deriva desde A en 1 paso. Entonces $A \Longrightarrow w$ y por tanto $\exists A \to w \in R$, si $w = a_1 a_2 \cdot a_n$, entonces el árbol que deriva tiene raíz A e hijos (hojas) $a_1, a_2, ..., a_n$.

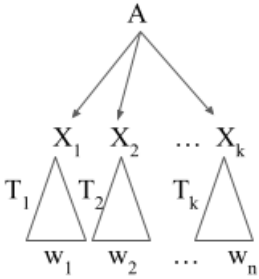
Derivación implica árbol (continuación)

Inducción: Asúmase que la derivación se da en $n+1$ pasos. Entonces $A \Longrightarrow X_1X_2\cdots X_k \Longrightarrow^* w$.

Divídase $w = w_1w_2\cdots w_k$ tal que $X_i \Longrightarrow^* w_i$. Con respecto a X_i puede haber varios casos:

1. X_i es terminal, entonces $X_i = w_i \in \Sigma$ y T_1 es un único nodo asociado a w_i
2. X_i es variable, entonces w_i es una cadena del lenguaje de X_i y $X_i \Longrightarrow w_i$ en a lo más n pasos, por lo que, por hipótesis de inducción, hay un árbol T_i que produce w_i.

Para construir el árbol, basta ver que ya que $A \Longrightarrow X_1X_2\cdots X_k$, entonces $\exists A \to X_1X_2\cdots X_k \in R$ y por la construcción que dimos del árbol, tomamos como raíz a A y como hijos a $T_1, T_2, ..., T_k$ en este orden (cuyas raíces eran X_i), de tal forma que la producción de A será $w = w_1w_2\cdots w_k$.



Árbol implica derivación

Demostraremos por inducción que si hay un árbol sintáctico T que produzca w con raíz A, entonces podemos encontrar la derivación $A \Longrightarrow w$.

Base: Sea T de profundidad 1. Entonces el árbol tiene un nodo raíz A y sus hijos son $a_1 a_2 \cdots a_k = w$, entonces debe ser que $A \to w \in R$, por lo que tenemos la derivación $A \Longrightarrow w$ en un paso.

Inducción: Si la profundidad de T es $n+1$, sea A la raíz y T_i los hijos de A asociados a X_i ($i \in \{1, 2, ..., k\}$):

1. Si X_i es terminal, entonces $X_i = w_i \in \Sigma$.
2. Si X_i es variable, entonces X_i como raíz del árbol T_i produce w_i, por lo que por h.i. $\exists X_i \Longrightarrow^* w_i$.

A continuación, a partir de esto, construimos la derivación.

Árbol implica derivación (continuación)

Para construir la derivación, haremos uso de la derivación por la izquierda, $A \Longrightarrow X_1X_2\cdots X_k$ para hacer inducción sobre $i = 0, 1, ...k$.

Base: Si $i = 0$, entonces la derivación es de la forma $A \Longrightarrow_{lm} X_1X_2\cdots X_k$ que es posible, pues sabemos que $A \to X_1X_2\cdots X_k \in R$.

Inducción: Si $i > 0$ entonces tendremos que $A \Longrightarrow^*_{lm} w_1w_2\cdots w_{i-1}X_i\cdots X_k$ es la derivación en el paso i. Entonces:

1. Si $X_i = w_i$ es terminal es claro que tendremos: $A \Longrightarrow^*_{lm} w_1w_2\cdots w_{i-1}w_iX_{i+1}\cdots X_k$.
2. Si X_i es variable y como sabemos que tiene el árbol T_i con profundidad a los más n, por h.i, tenemos que $X_i \Longrightarrow^* w_i$, por lo que tendremos que:

$$A \Longrightarrow^*_{lm} w_1w_2\cdots w_{i-1}X_i\cdots X_k \Longrightarrow^*_{lm} w_1w_2\cdots w_{i-1}w_iX_{i+1}\cdots X_k$$

Que es lo mismo que $A \Longrightarrow^*_{lm} w_1w_2\cdots w_{i-1}w_iX_{i+1}\cdots X_k$.

Finalmente, ya que i va de 0 hasta k (número de variables) tenemos que:
$A \Longrightarrow^*_{lm} w1w_2\cdots w_k = w$.

Reconocimiento de cadenas

Corolario

Dada una CFG $G = (\Sigma, \Delta, S, R)$, una cadena $w \in \Sigma^*$ es producida por la gramática $G \models w$ si:

1. Existe una derivación de S a w.
2. Existe un árbol sintáctico con raíz S que produzca w.

En cualquiera de estos dos casos $w \in L(G)$.

De esta forma, sabemos que toda cadena que pueda ser parseada dentro de un árbol sintáctico debe ser parte de un lenguaje libre de contexto.

Ambigüedad

Ambigüedad

Decimos que una sentencia es ambigua si se le pueden atribuir dos o más valores semánticos (significados).

Ambigüedad estructural

Una cadena w es ambigua si su estructura es ambigua; esto es, si dada una gramática G se le pueden atribuir dos o más estructuras (árboles sintácticos).

Gramática ambigua

Una CFG $G = (\Sigma, \Delta, S, R)$ se dice que es ambigua si $\exists w \in \Sigma^*$ tal que hay dos o más árboles que producen w.

Ejemplo: ambigüedad

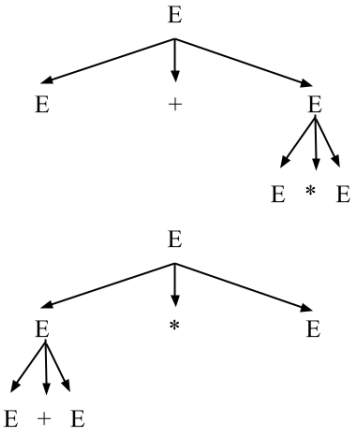
La gramática G dada por las producciones:

$$E \to I|E+E|E \cdot E|(E)$$
$$I \to a|b|Ia|Ib|I0|I1$$

Es ambigua, esto lo podemos ver con la cadena

$$E+E \cdot E$$

que produce al menos dos árboles.



Derivaciones y ambigüedad

Dada una cadena w, si existen diferentes derivaciones $S \Longrightarrow^* w$, no se implica que la gramática es ambigua. Es decir: La ambigüedad no se puede definir en términos de la derivación.

Considérese la cadena $a + b$ que genera la grmática. Esta cadena tiene dos derivaciones:

- $E \Longrightarrow E + E \Longrightarrow I + E \Longrightarrow a + E \Longrightarrow a + I \Longrightarrow a + b$
- $E \Longrightarrow E + E \Longrightarrow E + I \Longrightarrow E + b \Longrightarrow I + b \Longrightarrow a + b$

Sin embargo, la cadena $a + b$ tiene un único árbol.

Debe notarse que ambas derivaciones dependen de por dónde se hace la sustitución, pues la primera es una derivación por la izquierda y la segunda por la derecha.

Derivaciones por la izquierda y ambigüedad

Teorema

Sea $G = (\Sigma, \Delta, S, R)$ una CFG y $G \models w$. Entonces, w tiene dos árboles si y sólo si tiene dos derivaciones por la izquierda.

($\Longrightarrow$) Supóngase w tiene dos árboles T_1 y T_2, tal que n es el primer nodo en que difieren. Entonces los hijos de n son diferentes en T_1 y T_2 por lo que existen dos reglas $X_n \to \gamma$ donde γ es distinto para T_1 y T_2. Por lo tanto sus derivaciones difieren.
($\Longleftarrow$) Supóngase w tiene dos derivaciones por la izquierda. Ent. sabemos que $\exists X \to \beta \in R$ que producen γ a partir de X por la izquierda. Si X es un nodo del árbol, entonces debe ser que sus hijos corresponde a los elementos de γ, pero ya que hay dos de estas derivaciones, necesariamente hay dos árboles.

Reducir la ambigüedad

Las principales causas que provocan que una gramática libre de contexto sea ambigua son:

1. No se respeta la procedencia de los operadores (por ejemplo en la expresión $E + E * E$.
2. Una secuencia de operadores idénticos puede agrupar por la izquierda o por la derecha (por ejemplo $(a + b) + c$ o $a + (b + c)$. En general, se prefiere la agrupación por la izquierda.

Para reducir la ambigüedad, generalmente, se introducen más variables que permiten organizar mejor los elementos:

1. Introducir variables que indiquen que operación tiene precedencia.
2. Introducir variables que permitan agrupar por la izquierda (derecha).

Ejemplo de reducción de ambigüedad

La gramática G siguiente es ambigua:

$$E \to I|E + E|E \cdot E|(E)$$
$$I \to a|b|Ia|Ib|I0|I1$$

Se pueden introducir variables de factor (F), término (T) y expresión (E) para reducir la ambigüedad.

La siguiente gramática produce el mismo lenguaje y no es ambigua.

$$E \to T|E + T$$
$$T \to F|T \cdot F$$
$$F \to I|(E)$$
$$I \to a|b|Ia|Ib|I0|I1$$

Ambigüedad inherente

No siempre es posible encontrar una gramática no ambigua para un lenguaje.

Ambigüedad inherente

Un lenguaje libre de contexto L es inherentemente ambiguo si toda posible gramática G tal que $L = L(G)$ es ambigua. Es decir, si no hay gramáticas no ambiguas que lo generen.
Si existe al menos una gramática G no ambigua con $L = L(G)$ entonces L no es ambiguo.

Ejemplo: El lenguaje $L = \{a^n b^n c^m d^m : n, m \geq 1\} \cup \{a^n b^m c^m d^n : n, m \geq 1\}$ es inherentemente ambiguo.
Pues las cadenas pueden generarse: 1) a's y b's para que sean iguales y c's y d's iguales; o 2) a's y d's para que sean iguales y b's y c's iguales.

Formas normales de gramáticas libres de contexto

Símbolos inútiles

Símbolo útil

Sea $G = (\Sigma, \Delta, S, R)$ una CFG. Un símbolo $X \in \Delta \cup \Sigma$ es útil si existe una derivación $S \Longrightarrow^* \alpha X \beta \Longrightarrow^* w \in \Sigma^*$. Es decir, es útil si se encuentra entre los pasos de una derivación que produce un terminal desde el inicial.
En otro caso, X es un símbolo inútil.

Ejemplo: Consideremos la gramática dada por las siguientes reglas:

$$S \to XY|0$$
$$X \to 1$$

Donde Y es un símbolo inútil pues no deriva en ningún terminal.

Símbolos que generan y alcanzable

Símbolo que genera

Dada una CFG $G = (\Sigma, \Delta, S, R)$, un símbolo $X \in \Delta$ genera si existe la derivación $X \Longrightarrow^* w$ para alguna cadena termina $w \in \Sigma$.

Símbolo alcanzable

Dada una CFG $G = (\Sigma, \Delta, S, R)$, un símbolo $X \in \Delta$ alcanzable si existe la derivación $S \Longrightarrow^* \alpha X \beta$ para algunas cadenas $\alpha, \beta \in (\Sigma \cup \Delta)^*$.

Proposición

Dada una CFG $G = (\Sigma, \Delta, S, R)$, un símbolo $X \in \Delta$ es útil si genera y es alcanzable. De otra forma, es inútil.

Ejemplo: símbolos inútiles

La siguiente gramática dada por el siguiente conjunto de reglas:

$$S \to XY|0X|1X$$
$$X \to 1|\epsilon$$
$$Z \to 0|\epsilon$$

Tenemos dos tipos de símbolos inútiles:

- Y es un símbolo que no genera, pues no deriva en ningún símbolo terminal.
- Z es un símbolo inalcanzable, pues no es posible llegar a este desde el inicial S.

Eliminar símbolos inútiles

Considérese la gramática dada por las siguientes reglas:

$$S \to XY|0X|1X$$
$$X \to 1X|\epsilon$$
$$Z \to 0X|\epsilon$$

Ya que Y no genera, se elimina la regla $S \to XY$ y como Z no es alcanzable se elimina la producción $Z \to 0|\epsilon$.
El resultado es la gramática:

$$S \to 0X|1X$$
$$X \to 1X|\epsilon$$

Que produce el lenguaje $L(G) = (0+1)1^*$

Equivalencia de gramática en forma estándar

Teorema

Sea $G = (\Sigma, \Delta, S, R)$ una CFG tal que $L(G) \neq \emptyset$. Si $G_1 = (\Sigma, \Delta_1, S, R_1)$ es la gramática producida al:

1. Eliminar símbolos que no generan de G,
2. eliminar los símbolos no alcanzables de G

Entonces G_1 sólo tiene símbolos útiles y $L(G) = L(G_1)$.

Para ver que la gramática sólo cuenta con símbolos útiles, observamos que:

- Ya que G_1 elimina los símbolos que no generan, sólo contiene símbolos que generan, estos es $X \in \Delta_1$ si $X \Longrightarrow^* w$.
- Ya que G_1 elimina símbolos no alcanzables, sólo cuenta con símbolos alcanzables, esto es $X \in \Delta_1$ si $S \Longrightarrow^* \alpha X \beta$ con $\alpha, \beta \in (\Sigma \cup \Delta_1)^*$.

Si α y γ son terminales, tenemos que $S \Longrightarrow^* \alpha X \beta \Longrightarrow \alpha w \beta \in \Sigma^*$. Si alguno no está compuesto por terminales, ya que son parte de G_1, necesariamente generan, por lo que $S \Longrightarrow^* \alpha X \beta \Longrightarrow w_\alpha w w_\beta \in \Sigma^*$.

Demostración (continuación)

Ahora, demostraremos que los lenguajes de la gramática original y la gramática sin símbolos inútiles son iguales.

- $L(G_1) \subseteq L(G)$. Ya que G_1 contiene las mismas reglas que G pero sin aquellas reglas inútiles, tenemos que $R_1 \subseteq R$, de donde se deduce que el lenguaje que deriva está contenido en el lenguaje de G. Esto es $L(G_1) \subseteq L(G)$.
- $L(G) \subseteq L(G_1)$. Sea $w \in L(G)$, por lo que necesariamente debe de darse que $S \Longrightarrow_G^* w$, ya que todos los símbolos en esta derivación son alcanzables y generan. Por lo tanto esta derivación también se da en G_1, esto es $S \Longrightarrow_{G_1}^* w$, por tanto $w \in L(G_1)$ y de aquí $L(G) \subseteq L(G_1)$.

De ambos casos, se obtiene que $L(G) = L(G_1)$.

Algoritmo para eliminar símbolos que no generan

Algorithm Eliminación de símbolos que no generan

```
1:  procedure GENERATING-SYMBOLS(G = (Σ, Δ, S, R))
2:      Symbols ← {}                                   (Guarda símbolos que generan)
3:      for a ∈ Σ ∪ {ϵ} do
4:          Symbols ← Symbols ∪ {a}                    (Base)
5:      end for
6:      for α ∈ Symbols do
7:          for A ∈ Δ do
8:              if A → α then
9:                  Symbols ← Symbols ∪ {A}            (Recursión)
10:             end if
11:         end for
12:     end for
13:     return Symbols
14: end procedure
```

Ejemplo

Considérese la siguiente gramática:

$$S \to 0B|1X$$
$$A \to B0|1SX|0$$
$$B \to 0SC|1BX$$
$$X \to SBD|0BX|1$$

1. Los símbolos $0, 1$ generan por la base del algoritmo.
2. Ya que $A \to 0$, $X \to 1$, A y X generan.
3. Ya que $S \to 1X$, S genera.

Los símbolos que generan son entonces:

$$\{S, A, X, 0, 1\}$$

Al considerar sólo estos símbolos en la gramática obtenemos la gramática con sólo símbolos generadores:

$$S \to 1X$$
$$A \to 1SX|0$$
$$X \to 1$$

Efectividad de eliminar símbolos que no generan

Teorema

Dada una CFG G, X es un símbolo que genera en G si y sólo si es encontrado por el algoritmo descrito.

Sea X símbolo que genera, ent. $X \Longrightarrow^* w$, demostramos por inducción que es encontrado por el algoritmo:

- Base: Si $X \in \Sigma$ (terminal), el algoritmo lo encuentra en el primer paso.
- Inducción: Si $X \in \Delta$ (no terminal) tenemos la derivación $X \Longrightarrow \alpha \Longrightarrow^* w$ de n+1 pasos. Ya que $\alpha \Longrightarrow^* w$ en n pasos, por h.i. α es encontrado por el algoritmo.
  Además, $X \to \alpha \in R$, por lo que X será encontrado por el algoritmo en el paso de recursión.

Demostración (continuación)

Ahora sea X un símbolo que ha sido encontrado por el algoritmo. Demostramos por inducción que X genera.

- Base: Supóngase que X fue encontrado en la fase base del algoritmo, entonces $X = a$ un símbolo en Σ o $X = \epsilon$. En cualquier caso, trivialmente, X genera.
- Inducción: Supóngase que X es encontrado por el algoritmo en la fase de recursión. Entonces debe ser que existe $X \to \alpha \in R$ con α un símbolo encontrado en una iteración previa.
  Por hipótesis de inducción α genera, por lo que tenemos que $\alpha \Longrightarrow^* w$, y además como tenemos que $X \to \alpha$ es una producción, tenemos la derivación:

$$X \Longrightarrow \alpha \Longrightarrow w$$

  Estos es, $X \Longrightarrow^* w$, por lo que X genera.

Algoritmo para eliminar símbolos no alcanzables

Algorithm Eliminación de símbolos no alcanzables

```
1:  procedure REACHABLE-SYMBOLS(G = (Σ, Δ, S, R))
2:      Symbols ← {S}                                          (Base)
3:      for A ∈ Symbols do
4:          for X ∈ Δ do
5:              if A → X then
6:                  Symbols ← Symbols ∪ {X}                    (Recursión)
7:              end if
8:          end for
9:      end for
10:     return Symbols
11: end procedure
```

Ejemplo

Considérese la siguiente gramática:

$$S \to 0B|1X$$
$$A \to B0|1SX|0$$
$$B \to 0SC|1BX$$
$$X \to SBD|0BX|1$$

1. El símbolo S es alcanzable.
2. Ya que $S \to 0B$, $S \to 1X$, B, X, así como $0, 1$ son alcanzables.
3. Como $B \to 0SC$ C es alcanzable, y como $X \to SBD$, D es alcanzable.

Los símbolos que alcanzables son entonces:

$$\{S, B, X, D, 0, 1\}$$

. Por lo que obtenemos:

$$S \to 0B|1X$$
$$B \to 0SC|1BX$$
$$X \to SBD|0BX|1$$

Al eliminar no generadores tenemos:

$$S \to 1X$$
$$X \to 1$$

Efectividad de eliminar símbolos no alcanzables

Teorema

Sea $G = (\Sigma, \Delta, S, R)$ una CFG, el símbolo X es alcanzable si y sólo si es encontrado por el algoritmo anterior.

Sea X un símbolo de G, demostramos por inducción que es encontrado por el algoritmo:

- Base: Si X se deriva en 0 pasos, entonces $X = S$ que es encontrado por el algoritmo en la base de éste.
- Inducción: Para $n+1$ pasos, como X alcanzable, ent. $X \Longrightarrow^* \alpha A \beta \Longrightarrow \alpha X \beta$, para algún símbolo A.
  Entonces, existe una producción $A \to X \in R$, pero ya que A se deriva en menos de n pasos por hipótesis de inducción, A es encontrado por el algoritmo. Y ya que $A \to X$ es producción, entonces X será encontrado por el algoritmo en la fase de recursión.

Demostración (continuación)

Sea X un símbolo encontrado por el algoritmo, por inducción mostramos que es alcanzable:

- Base: Si X es encontrado en la base de la recursión, entonces $X = S$, pues es el único símbolo que se agrega aquí. Y S es trivialmente alcanzable.
- Si X es encontrado en n pasos en la recursión del algoritmo, entonces existe la producción $A \to X$, siendo que A ha sido encontrado por el algoritmo en menos de n pasos.
  Por hipótesis de inducción A es alcanzable; esto es $S \Longrightarrow^* \alpha A \beta$, pero ya que tenemos la producción $A \to X$, entonces:

$$S \Longrightarrow^* \alpha A \beta \Longrightarrow \alpha X \beta$$

  Esto es: $S \Longrightarrow^* \alpha X \beta$, por lo que X es alcanzable.

Producciones ϵ

Variable anulable

Una variable X de una gramática libre de contexto es anulable si:

$$A \Longrightarrow^* \epsilon$$

Los símbolos anulables no contribuyen a generar cadenas de un lenguaje, pues dentro de una cadena el símbolo ϵ se anula. Sin embargo, habrá lenguajes que puedan tener la cadena ϵ, por lo que al eliminar variables anulables consideremos lenguajes sin cadenas ϵ.
Si tenemos reglas $X \to AB$, $A \to \epsilon$ se puede producir una cadena desde X al menos que B también sea anulable.

Algoritmo para detección símbolos anulables

Algorithm Eliminación de símbolos no alcanzables

1: **procedure** NULLABLE-SYMBOLS($G = (\Sigma, \Delta, S, R)$)
2: $Symbols \leftarrow \{\}$
3: **for** $X \in \Delta$ **do**
4: **if** $X \rightarrow \epsilon \in R$ **then**
5: $Symbols \leftarrow Symbols \cup \{X\}$ (Base)
6: **end if**
7: **end for**
8: **for** $A \in \Delta$ **do**
9: **if** $A \rightarrow C_1 C_2 \cdots C_k \in R$ **and** $\forall i = 1, ..., k, C_i \in Symbols$ **then**
10: $Symbols \leftarrow Symbols \cup \{A\}$ (Recursión)
11: **end if**
12: **end for**
13: **return** $Symbols$
14: **end procedure**

Construcción de gramática sin producciones ϵ

El método para construir la gramática G_1 sin producciones ϵ es el siguiente:

1. Para toda producción $A \to C_1 C_2 \cdots C_k$, con $m \geq k$ símbolos anulables.
2. Si $k = 1$, esto es si $A \to \epsilon$, no se incluirá la producción.
3. Se harán 2^m copias de esta producción de tal forma que cada C_i anulable está presente o ausente en todas las combinaciones posibles. Esto es, se incluirán producciones de la forma:
$$A \to C_1 C_2 \cdots C_{i-1} C_{i+1} \cdots C_k$$
donde se ha quitado el símbolo anulable C_i.
4. Si $m = k$ se excluirá la producción donde no se tenga ningún símbolo, pues sería anulable.

Ejemplo

Considérese la siguiente gramática libre de contexto:

$$S \to AB$$
$$A \to aAA|\epsilon$$
$$B \to bBB|\epsilon$$

1. Como $A \to \epsilon$, A es anulable.
2. Ya que $S \to AB$, S también es anulable.

Para construir las producciones sin transiciones ϵ procedemos como:

2. De $S \to AB$ obtenemos $S \to AB|A|B$.
3. De $A \to \epsilon$ obtenemos $A \to aAA|aA|a$.
4. De $B \to \epsilon$ obtenemos $B \to bBB|bB|b$.

La grámatica final es:

$$S \to AB|A|B$$
$$A \to aAA|aA|a$$
$$B \to bBB|bB|b$$

Gramáticas sin producciones ϵ

Teorema

El algoritmo descrito anteriormente encuentra un símbolo si y sólo si éste es anulable.

Teorema

Si G_1 es una gramática construida a partir de G mediante el procedimiento descrito anteriormente, entonces $L(G_1) = L(G) \setminus \epsilon$.

Demostraremos que si $A \Longrightarrow_{G_1}^* w$ y $w \neq \epsilon$, ent. $A \Longrightarrow_G^* w$, por inducción sobre las derivaciones.

- Base: En un paso $A \Longrightarrow_{G_1}^* w$, por lo que $A \to w \in R_1$ de G_1, entonces debe haber α tal que en G, $A \to \alpha \in R$ con $A \Longrightarrow_G \alpha \Longrightarrow_G^* w$.
- Inducción: Derivación en n pasos, implica $A \Longrightarrow_{G_1} C_1 \cdots C_k \Longrightarrow_{G_1}^* w$, de la primera parte, $A \to C_1 \cdots C_k \in R_1$, tómese $w = w_1 \cdots w_k$ con $C_i \Longrightarrow_{G_1}^* w_i$, ya que estas derivaciones son en menos de n pasos, por h.i, tenemos $C_i \Longrightarrow_G^* w_i$, y por la construcción deben existir $Y_1 \cdots Y_m$ tal que $A \Longrightarrow_{G_1} Y_1 \cdots Y_m \Longrightarrow_{G_1}^* C_1 \cdots C_k \Longrightarrow_G^* w$.

Producciones unitarias

Producción unitaria

Una variable unitaria $A \in \Delta$ es una variable que se encuentra en una producción de la forma

$$A \to B$$

Para alguna variable $B \in \Delta$.

Par unitario

Un par unitario es un par de variables (A, B) tal que existe una derivación

$$A \Longrightarrow^* B$$

que se da a partir de sólo producciones unitarias.

Algoritmo para encontrar pares unitarios

Algorithm Encontrar pares unitarios

1: **procedure** UNITARY-PAIRS($G = (\Sigma, \Delta, S, R)$)
2: $Symbols \leftarrow \{\}$
3: **for** $A \in \Delta$ **do**
4: $Symbols \leftarrow Symbols \cup \{(A, A)\}$ (Base)
5: **end for**
6: **for** $B \to C \in R$ **do**
7: **if** $(A, B) \in Symbols$ para alguna variable A **then**
8: $Symbols \leftarrow Symbols \cup \{(A, C)\}$ (Recursión)
9: **end if**
10: **end for**
11: **return** $Symbols$
12: **end procedure**

Ejemplo

Considérese la siguiente gramática libre de contexto dada por las reglas:

$$I \to a|b|Ia|Ib|I0|I1$$
$$F \to I|(E)$$
$$T \to F|T \cdot F$$
$$E \to T|E + T$$

Aplicando el algoritmo:

1. En la base $(E, E), (T, T), (F, F), (I, I)$
2. Como (E, E) y $E \to T$, entonces (E, T) es par unitario.
3. Como (E, T) y $T \to F$, entonces (E, F) es par unitario.
4. Como (E, F) y $F \to I$, entonces (E, I) es par unitario.
5. Como (T, T) y $T \to F$, entonces (T, F) es par unitario.
6. Como (T, F) y $F \to I$, entonces (T, I) es par unitario.
7. Como (F, F) y $F \to I$, entonces (F, I) es par unitario.

Eficiencia para encontrar pares unitarios

Teorema

Dada una CFG, un par de variables (A, B) es unitario si y sólo si es encontrado por el algoritmo anterior.

Demostramos por inducción que si (A, B) unitario, entonces el par es encontrado por el algoritmo.

- Base: La derivación en 0 pasos implica que $A \Longrightarrow^* A$, por lo que el par (A, A) es encontrado en la base del algoritmo.
- Inducción: Sea (A, B) unitario con derivación $A \Longrightarrow^* B$ en n pasos. Como el par es unitario, entonces la derivación se da sólo en producciones unitarias. Por tanto $\exists C \in \Delta$ tal que $A \Longrightarrow^* C \Longrightarrow B$.

  Como $A \Longrightarrow^* C$ es una derivación en menos de n pasos, por h.i., el par (A, C) es encontrado por el algoritmo, pero por la derivación $C \Longrightarrow B$, tenemos la producción $C \to B$ unitaria, y por tanto el par (A, B) es encontrado en la recursión del algoritmo.

Demostración (continuación)

Demostraremos que si el par (A, B) es encontrado por el algoritmo, entonces debe ser que (A, B) es un par unitario.

- Base:Si el par es encontrado en la base del algoritmo, entonces $B = A$, por lo que el par (A, A) es trivialmente unitario, puesto que tenemos $A \Longrightarrow^* A$.
- Inducción: Supóngase que el par (A, B) es encontrado por el algoritmo en n pasos (de la recursión). Entonces, existe una producción $C \to B$ de la gramática tal que el par (A, C) ha sido encontrado en menos de n pasos.
  Por hipótesis de inducción, el par (A, C) es unitario. Por tanto, hay una derivación de producciones unitarias $A \Longrightarrow^* C$. Pero ya que además tenemos $C \to B$, entonces $C \Longrightarrow B$ es una derivación con producción unitaria.
  De aquí obtenemos que la derivación

$$A \Longrightarrow^* B$$

se da sólo con producciones unitarias, por lo que el par (A, B) es unitario.

Eliminar producciones unitarias

El método para eliminar producciones unitarias consiste en, dada una CFG $G = (\Sigma, \Delta, S, R)$, la nueva gramática sin producciones unitarias G_1 tiene producciones construidas como:

1. Se encuentran todos los pares unitarios de G por el algoritmo anterior.
2. Para cada par (A, B) se añade a R_1, las producciones de G_1, las producciones $A \to \alpha$, donde α es producción no unitaria y $B \to \alpha$ en G.

Teorema

La gramática G_1 generada por este procedimiento a partir de G no contiene producciones unitarias y además $L(G) = L(G_1)$.

Ejemplo

De la gramática anterior para las expresiones de suma y multiplicación, usando los pares unitarios, podemos obtener las siguientes producciones:

Par	Producción
(E, E)	$E \to E + T$
(E, T)	$E \to T \cdot F$
(E, F)	$E \to (E)$
(E, I)	$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(T, T)	$T \to T \cdot F$
(T, F)	$T \to (E)$
(T, I)	$T \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(F, F)	$F \to (E)$
(F, I)	$F \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
(I, I)	$I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

Gramáticas en forma simplificada

Gramática en forma simplificada

Dada una gramática libre de contexto $G = (\Sigma, \Delta, R, S)$ la forma simplificada de G es la gramática que:

- No contiene producciones ϵ.
- No contiene producciones unitarias.
- No contiene símbolos inútiles.

Para obtener la forma simplificada de una gramática se usarán los métodos para:

1. Eliminar producciones ϵ.
2. Eliminar producciones unitarias.
3. Eliminar símbolos inútiles.

Ejemplo: Forma simplificada

Considérese la gramática dada por las siguientes producciones:

$$
\begin{aligned}
S &\to E \\
E &\to V|C|XS|SYS|(S) \\
V &\to x|y|z \\
C &\to 0|1|2|\cdots|9 \\
X &\to -|\epsilon \\
Y &\to +|\cdot
\end{aligned}
$$

$X \to \epsilon$ es anulable, por lo que esta producción se elimina.

Como $E \to XE$ contiene una variable anulable, entonces agregamos las reglas $E \to E$ y $E \to XE$.
La gramática resultante es:

$$
\begin{aligned}
S &\to E \\
E &\to V|C|S|XS|SYS|(S) \\
V &\to x|y|z \\
C &\to 0|1|2|\cdots|9 \\
X &\to -|\epsilon \\
Y &\to +|\cdot
\end{aligned}
$$

Ejemplo: eliminar unitarios

Par	Producción
(S,S)	$\emptyset$
(E,E)	$E \to XS\|SYS\|(S)$
(V,V)	$V \to x\|y\|z$
(C,C)	$C \to 0\|1\|2\|\cdots\|9$
(X,X)	$X \to -$
(Y, Y)	$Y \to +\|\cdot$
(S,E)	$S \to XS\|SYS\|(S)$
(E,V)	$E \to x\|y\|z$
(E,C)	$E \to 0\|1\|2\|\cdots\|9$
(S,V)	$S \to x\|y\|z$
(S,C)	$S \to 0\|1\|2\|\cdots\|9$

La gramática resultante es la siguiente:

$$S \to XS|SYS|(S)|x|y|z|0|1|\cdots|9$$
$$E \to XS|SYS|(S)|x|y|z|0|1|\cdots|9$$
$$X \to -$$
$$Y \to +|\cdot$$

Ejemplo: Eliminar símnolos inútiles

Encontramos los símbolos alcanzables:

1. S es alcanzable.
2. Como $S \to XS$, entonces X es alcanzable
3. Como $S \to SYS$, entonces Y es alcanzable
4. E es no alcanzable.

Eliminando las producciones con símbolos inalcanzables obtenemos finalmente la forma simplificada de la gramática:

$$S \to XS|SYS|(S)|x|y|z|0|1|\cdots|9$$
$$X \to -$$
$$Y \to +|\cdot$$

Equivalencia de forma simplificada

Teorema

Sea G una CFG y sea G_1 la forma simplificada de G, entonces $L(G) \setminus \{\epsilon\} = L(G_1)$.

Sea $L(G)$ el lenguaje de la gramática G. Tomemos a G_3 como el lenguaje sin producciones ϵ producida a partir de G, sabemos que $L(G_3) = L(G) \setminus \{\epsilon\}$.
Si G_2 es la gramática sin producciones unitarias construida a partir de G_3, también sabemos que $L(G_3) = L(G_2)$.
Finalmente, G_1 es la gramática sin símbolos inútiles a partir de G_2, por lo que tenemos que $L(G_1) = L(G_2)$.
Ya que G_1 es la forma simplificada, y por transitividad tenemos que $L(G_1) = L(G) \setminus \{\epsilon\}$.

Forma Normal de Chomsky

Forma normal de Chomsky

Una gramática libre de contexto $G = (\Sigma, \Delta, S, R)$ se dice que está en forma normal de Chomsky (CFN), si todas sus producciones son de una de las siguientes dos formas:

$$X \to YZ$$
$$X \to a$$

Para variables $X, Y, Z \in \Delta$ y símbolo terminal $a \in \Sigma$.

Algoritmo de CNF

Algorithm Convertir en CFN

1: **procedure** $\mathrm{CFN}(G = (\Sigma, \Delta, S, R)$ en forma simplificada)
2: $\quad R_{CFN} \leftarrow \{\}$
3: $\quad$ **for** $X \to \alpha \in R$ **do**
4: $\quad\quad$ **if** $\alpha = X_1 \cdots a \cdots X_k$ con $a \in \Sigma$ **then**
5: $\quad\quad\quad R_{CFN} \leftarrow R_{CFN} \cup \{X \to X_1 \cdots Y \cdots X_k, Y \to a\}$
6: $\quad\quad$ **end if**
7: $\quad$ **end for**
8: $\quad$ **for** $X \to \alpha \in R$ **do**
9: $\quad\quad$ **if** $\alpha = X_1 X_2 \cdots X_k$ con X_i variables y $k > 2$ **then**
10: $\quad\quad\quad R_{CFN} \leftarrow R_{CFN} \cup \{X \to X_1 Y_1, Y_1 \to X_2 Y_2, ..., X_{k-2} \to X_{k-1} X_k\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end procedure**

Ejemplo

Considérese la gramática en forma simplificada siguiente:

$$S \to XS|SYS|(S)|x|y|z|0|1|\cdots|9$$
$$X \to -$$
$$Y \to +|\cdot$$

Aplicando el método anterior para obtener la CFN:

1. De $S \to (S)$, se crearán $S \to Z_1SZ_2$ y $Z_1 \to (, Z_2 \to)$.
2. De $S \to SYS$, se crean $S \to SZ_3$, $Z_3 \to YZ$.
3. De $S \to Z_1SZ_2$ se crean $S \to Z_1Z_4$ y $Z_4 \to SZ_2$

La gramática resultantes es:

$$S \to XS|SZ_3|Z_1Z_4$$
$$Z_3 \to YS$$
$$Z_4 \to SZ_2$$
$$Z_2 \to)$$
$$Z_1 \to ($$
$$S \to x|y|z|0|1|\cdots|9$$
$$X \to -$$
$$Y \to +|\cdot$$

Equivalencia de CNF

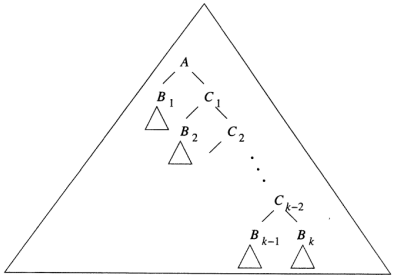
Teorema

Sea $G = (\Sigma, \Delta, S, R)$ una CFG tal que $\epsilon \notin L(G)$, entonces existe una gramática en forma normal de Chomsky G_1 tal que $L(G) = L(G_1)$.

Por el teorema anterior, sabemos que existe G_2 en forma simplificada y que cumple que $L(G) = L(G_2)$. Demostraremos que por medio de la construcción anterior, obtenemos una gramática en CFN G_1 que cumple que $L(G_2) = \in L(G_1)$.
Supóngase $w \in L(G_2)$, entonces existe $S \Longrightarrow^*_{G_2} w$ en G_2. Dentro de esta derivación, tenemos derivaciones en un paso de la forma $A \Longrightarrow_{G_2} X_1 \cdots X_k$. Si $k > 2$, por la construcción de G_1, la derivación se da en al menos $k-1$, $A \Longrightarrow^*_{G_1} X_1 \cdots X_k$.
Si X_i es terminal, entonces se crea $Y_i \to X_i$. En otro caso, las derivaciones son de la forma $A \Longrightarrow_{G_1} X_1 Y_1 \Longrightarrow_{G_1} X_1 X_2 Y_2 \Longrightarrow_{G_1} X_1 X_2 X_3 Y_3 \Longrightarrow^*_{G_1} X_1 X_2 \cdots X_k \Longrightarrow^*_{G_1} w$. Por lo que $w \in L(G_1)$; i.e. $L(G_2) \subseteq L(G_1)$.

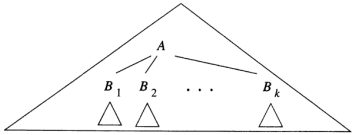
Demostración (continuación)

Supóngase ahora que $w \in L(G_1)$, entonces existe un árbol con raíz S que produce w. Sea A un nodo de este árbol con hijos B_1, C_1, y C_1 con hijos $B_2 C_2$, etc.



Eliminamos las variables C_i introducidas en CFN. Tal que tenemos el nodo A con hijos $B_1, ..., B_k$.

Si $A \to a$ es introducido en el CFN, en el árbol A tiene hijo a, quitamos A y sustituimos por a.



El árbol resultante produce w. Esto es $w \in L(G_2)$.

Concluimos que $L(G_1) = L(G_2) = L(G)$.

Algoritmo de CYK

Algorithm Algoritmo de CYK

procedure PARSE($w = w_1 \cdots w_n$, $G = (\Sigma, \Delta, S, R)$ en CFN)
    $T \leftarrow$ TABLE($n \times n$)
    for j **from** 0 **to** n **do**
        for w_j **in** TOKENIZE(string) **do**:
            $T[j, j] \leftarrow T[j, j] \cup \{X\}$; **si** $X \rightarrow w_j$
        end for
        for i **from** j **to** 0 **do**
            for k **from** i **to** $j + 1$ **do**
                $T[i, j] \leftarrow T[i, j] \cup \{X\}$; **si** $X \rightarrow Y_0 \cdot Y_1$ **and** $Y_0 \in T[i, k]$, $Y_1 \in T[k + 1, j]$
            end for
        end for
    end for
    return $S \in T[1, n]$
end procedure

Ejemplo

Considérese la siguiente CFG en forma norma de Chomsky:

$$S \to AB|BC$$
$$A \to BA|a$$
$$B \to CC|b$$
$$C \to AB|a$$

Aplicaremos el algoritmo a la cadena:

$$baaba$$

La tabla resultante es la siguiente:

	b	a	a	b	a
b	$\{B\}$	$\{S, A\}$	$\{S, C\}$		$\{S, A, C\}$
a		$\{A, C\}$	$\{B\}$	$\{S, B\}$	$\{S, A, C\}$
a			$\{A, C\}$	$\{S, C\}$	$\{B\}$
b				$\{B\}$	$\{A, S\}$
a					$\{A, C\}$

Ya que $S \in T[1, 5]$, la cadena *baaba* es aceptada por la gramática.

Forma normal de Greibach

Forma normal de Greibach

Una gramática libre de contexto $G = (\Sigma, \Delta, S, R)$ se dice que está en forma normal de Greibach si todas sus producciones son de la forma:

$$X \to a\alpha$$

Donde $a \in \Sigma$ y $\alpha \in \Delta^*$.

La forma normal de Greibach considera producciones que siempre cuentan con un símbolo, y que son seguidos de 0 o más variables no terminales. Por ejemplo, la regla:

$$X \to 0X_1X_2 \cdots X_k$$

Está en forma normal de Greibach.

Recursión por la izquierda

Recursión por la izquierda

Sea G una CFG, una producción de la gramática se dice que es recursiva por la izquierda si es de la forma:

$$X \to X\alpha$$

Un símbolo X es recursivo por la izquierda si se tiene la derivación:

$$X \Longrightarrow^* X\alpha$$

Dada una producción: $X \to X\alpha|\beta$, para eliminar la recursión por la izquierda se genera un nuevo símbolo y las dos nuevas producciones:

$$X \to \beta Y$$
$$Y \to \alpha Y|\epsilon$$

Conversión a forma normal de Greibach

Para convertir una gramática libre de contextos G a su forma normal de Greibach se seguirán los siguientes pasos:

1. Convertir G a la forma normal de Chomsky (CFN), pues partiremos de esta para la forma de Greibach.
2. Eliminar las recursiones por la izquierda de las producciones de la gramática.
3. Expandir la primera varibale de todo cuerpo de producción hasta obtener un símbolo terminal.

Ejemplo

Considérese la siguiente CFG en CNF:

$$
\begin{aligned}
S &\to XS|SZ_3|Z_1Z_4\\
S &\to x-z|0-9\\
Z_3 &\to YS\\
Z_4 &\to SZ_2\\
Z_2 &\to)\\
Z_1 &\to (\\
X &\to -\\
Y &\to +|\cdot
\end{aligned}
$$

Tenemos la recursión $S \to SZ_3$, por lo que producimos las nuevas reglas $S \to XSS'|Z_1Z_4S'|x-zS'|0-9S'$, y $S' \to Z3S'|Z_3S'|\epsilon$. Además expandimos para obtener la gramática en forma normal de Greibach:

$$
\begin{aligned}
S &\to -SS'|(Z_4S'\\
S &\to x-zS'|0-9S'\\
S' &\to +SS'|\cdot SS'|\epsilon\\
Z_4 &\to -SS'Z_2|(Z_2S'Z_2|x-zS'Z_2|0-9S'Z_2\\
Z_2 &\to)
\end{aligned}
$$

Autómapas de pila

Autómada de pila

Autómata de pila

Un autómata de pila es una 7-tupla $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_o, F)$ donde sus elementos son:

1. $Q = \{q_0, q_1, ..., q_n\}$ un conjunto finito de estados.
2. $\Sigma = \{a_1, a_2, ..., a_m\}$ conjunto de símbolos del alfabeto.
3. $\Gamma = \{Z_0, Z_1, ..., Z_k\}$ alfabeto de símbolos de la pila.
4. La función de transición $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to Q \times (\Gamma \cup \{\epsilon\})^*$. Toma un estado, un símbolo del alfabeto (incluyendo ϵ) y un símbolo en la pila y regresa un estado y una cadena de símbolos de pila $\delta(q, a, X) = (p, Y)$.
5. $q_0 \in Q$ es el estado inicial y $Z_0 \in \Gamma$ es el símbolo de inicio de la pila.
6. $F \subseteq Q$ es el conjunto de estados finales.

Ejemplo de Autómata de pila

Considérese el lenguaje (cadenas con tantos 0's como 1's):

$$L = \{a^n b^n \in \Sigma^* : a, b \in \Sigma, a \neq b, n > 0\}$$

con alfabeto $\Sigma = \{0, 1\}$. El autómata de pila será $P = (\{q_0, q_1, q_2\}, \Sigma, \{0, 1, Z_0\}, \delta, q_0, Z_0, \{q_2\})$. Las transiciones estarán dadas de la siguiente forma:

$$\begin{aligned}
\delta(q_0, 0, Z_0) &= (q_0, 0Z_0) & \qquad \delta(q_0, 1, Z_0) &= (q_0, 1Z_0) \\
\delta(q_0, 0, 0) &= (q_0, 00) & \qquad \delta(q_0, 0, 1) &= (q_0, 01) \\
\delta(q_0, \epsilon, Z_0) &= (q_1, Z_0) & \qquad \delta(q_0, \epsilon, 0) &= (q_1, 0) \\
\delta(q_0, \epsilon, 1) &= (q_1, 1) & \qquad \delta(q_1, \epsilon, Z_0) &= (q_2, Z_0) \\
\delta(q_1, 0, 1) &= (q_1, \epsilon) & \qquad \delta(q_1, 1, 0) &= (q_1, \epsilon)
\end{aligned}$$

Cada vez que ve un el símbolo a al inicio, se pone en la pila, acabando estos símbolos, de aquí transita hacia q_1 donde al pone el signo contrario al que ve en la pila tantas veces cómo lo encuentre en la pila.

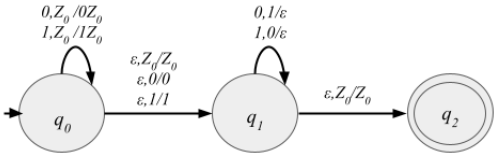
Diagrama de autómata de pila

Los autómatas de pila pueden representarse a través de un diagrama de transiciones con las siguientes consideraciones:

1. Los nodos de la gráfica corresponden a los estados Q; un nodo conecta a otro si se da la transición de ese estado al siguiente.
2. El inicial se indica con una flecha y el final con doble círculo.
3. Cada arista se etiqueta de la forma

$$a, X/Y$$

donde $a \in \Sigma \cup \{\epsilon\}$, $X \in \Gamma$ y $Y \in \Gamma^*$ siempre que $\delta(p, aX) = (q, Y)$.



Configuración de un autómata de pila

Configuración

Dado un autómata de pila P, su configuración es una tripleta (q, w, γ) donde:

1. q es el estado en que se encuentra P.
2. w es la subcadena que resta de la entrada después de pasar al estado q.
3. γ es el contenido de la pila en ese estado.

Ejemplo: Del autómata anterior, si el input es la cadena 111000, la configuración inicial del autómata será $(q_0, 111000, Z_0)$, la primera vez que pasa al estado q_1, su configuración será:

$$(q_1, 000, 111Z_0)$$

Descripción instantánea de un autómata de pila

Descripción instantánea

Dado un autómata de pila $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ una descripción instantánea es la configuración del autómata en un instante dado.
Si el autómata tiene una transición de la forma: $\delta(q, a, X) = (p, Y)$ con una cadena de entrada $w = yax$, denotamos el cambio de configuración de q a p con el símbolo a como:

$$(q, aw, X\beta) \vdash_P (p, w, Y\beta)$$

Las descripciones instantáneas nos permiten denotar el proceso que un autómata con pila hace para procesar una cadena.

Ejemplo

Del autómata de pila anterior que procesa el lenguaje $L = \{a^n b^n\}$, podemos ver que a partir de las descripción instantánea, la cadena 111000 se procesa como:

$$\begin{aligned}(q_0, 111000, Z_0) &\vdash (q_0, 11000, 1Z_0) \vdash (q_0, 1000, 11Z_0) \vdash (q_0, 000, 111Z_0) \\ &\vdash (q_1, 000, 111Z_0) \vdash (q_1, 00, 11Z_0) \vdash (q_1, 0, 1Z_0) \\ &\vdash (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0)\end{aligned}$$

Que llega al estado final. Por su parte la cadena 001 tiene la siguiente descripción instantánea:

$$\begin{aligned}(q_0, 001, Z_0) &\vdash (q_0, 01, 0Z_0) \vdash (q_0, 1, 00Z_0) \\ &\vdash (q_1, 1, 00Z_0) \vdash (q_1, \epsilon, 0Z_0)\end{aligned}$$

Que no llega al estado final.

Extensión de la descripción instantánea

Extensión de descripción instantánea

Dado un autómata de pila $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ con descripción $\vdash_P$, la extensión de ésta, denotada $\vdash_P^*$ se define como:

- Para toda configuración C, se tiene $C \vdash_P^* C$. (Reflexividad)
- Si $C_1 \vdash_P^* K$ y $K \vdash_P^* C_2$, entonces $C_1 \vdash_P^* C_2$. (Transitividad)

De esta forma, de los casos anteriores tenemos que:

$$(q_0, 111000, Z_0) \vdash^* (q_2, \epsilon, Z_0)$$

Y por otra parte:

$$(q_0, 001, Z_0) \vdash^* (q_1, \epsilon, 0Z_0)$$

Teorema

Sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un autómata de pila tal que $(q, x, \alpha) \vdash_P^* (p, y, \beta)$, entonces para toda $w \in \Sigma^*, \gamma \in \Gamma^*$ se cumple:

$$(q, xw, \alpha\gamma) \vdash_P^* (p, yw, \beta\gamma)$$

Lo demostraremos por inducción sobre la longitud de la descripción.

Base: Si la descripción se da en un paso $(q, xw, \alpha\gamma) \vdash_P^* (q, xw, \alpha\gamma)$ que por definición de $\vdash^*$ se cumple.

Inducción: Supóngase que $(q, x, \alpha) \vdash^* (p, y, \beta)$ conlleva n pasos. Por definición, tenemos que $(q, x, \alpha) \vdash^* (q', ay, X) \vdash (p, y, \beta)$. Por hipótesis de inducción $(q, xw, \alpha\gamma) \vdash^* (q', ayw, X\rho\gamma)$ pero ya que $\delta(q', a, X) = (p, \beta)$ (el resto de la cadena no le importa y sólo observa el tope de la pila), tenemos que $(q', ayw, X\rho\gamma) \vdash (p, yw, \beta\gamma)$. Por tanto:

$$(q, xw, \alpha\gamma) \vdash_P^* (p, yw, \beta\gamma)$$

Lenguajes aceptados por PDAs

Lenguaje aceptado por autómata de pila

Sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un autómata de pila, diremos que el lenguaje de este autómata está definido com:

$$L(P) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash_P^* (q_f, \epsilon, \gamma), q_f \in F\}$$

En este caso, asumiremos que $\gamma \in \Gamma^*$

El lenguaje aceptado por un autómata de pila es está constituido por todas las cadenas w tal que se puede llegar desde el estado inicial hasta el final consumiendo los símbolos.
No es necesario que los símbolos de la pila se consuman también.

Ejemplo

Proposición

El autómata de pila descrito anteriormente (pp. 71) acepta al lenguaje:

$$L(P) = \{a^n b^n : n > 0, a \neq b\}$$

donde a, b son símbolos del alfabeto $\Sigma = \{0, 1\}$

Para mostrar que este autómata acepta el lenguaje dicho, usaremos inducción sobre n:
Base: Cuando $n = 1$, tenemos cadenas ab (00, 01, 10 ó 11, pues $a \neq b$). En cualquier caso, usando las transiciones $\delta(q_0, a, Z_0) = (q_0, aZ_0), \delta(q_1, b, a) = (q_1, \epsilon)$ y $\delta(q_1, \epsilon, Z_0) = (q_2, Z_0)$ tenemos la descripción:

$$(q_0, ab, Z_0) \vdash (q_0, b, aZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0)$$

Esto es $(q_0, ab, Z_0) \vdash^* (q_2, \epsilon, Z_0)$ y ya que $q_2 \in F$, la cadena es aceptada.

Ejemplo (continuación)

Inducción: Ahora supongamos que tenemos la cadena $a^{n+1}b^{n+1}$. Demostraremos que esta cadena es aceptada por el autómata dado.
En primer lugar, tomemos la cadena $a^n b^n$ que, por hipótesis de inducción es aceptada por el autómata y tiene una descripción $(q_0, a^n b^n, Z_0) \vdash^* (q_2, \epsilon, Z_0)$.
Descomponemos los pasos de esta descripción:

- Para pasar a q_1 tenemos: $(q_0, a^n b^n, Z_0) \vdash (q_0, b^n, a^n Z_0) \vdash (q_1, b^n, a^n Z_0)$.
- Para pasar a q_2 tenemos: $(q_1, b^n, a^n Z_0) \vdash^* (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0)$

Por lo que al tomar $a^{n+1}b^{n+1}$ tenemos que:

- Para pasar a q_1 se debe consumir un símbolo de más en q_0 $(\delta(q_0, a, a) = (q_0, a))$:
  $(q_0, a^{n+1}b^{n+1}, Z_0) \vdash (q_0, ab^{n+1}, a^n Z_0) \vdash (q_1, b^{n+1}, aa^n Z_0) \vdash (q_1, b^{n+1}, a^{n+1} Z_0)$.
- En q_1 el símbolo a agregado permite consumir el símbolo b $(\delta(q_1, b, a) = (q_1, \epsilon))$:
  $(q_1, b^{n+1}, a^n a Z_0) \vdash^* (q_1, b, aZ_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0)$

Por tanto, el autómata llega al estado final y acepta la cadena.

Aceptación por vaciado de pila

Aceptación al vaciar la pila

Sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un autómata de pila, definimos el lenguaje aceptado por P al vaciar la pila como:

$$N(P) = \{w \in \Sigma^* : (q_0, w, Z_0) \vdash_P^* (q, \epsilon, \epsilon)\}$$

Tal que $q \in Q$ (no necesariamente final).

Ejemplo: El autómata de pila para las cadenas $a^n b^n$ no vacía nunca la pila, pues siempre permanece Z_0.

Pero podemos crear un autómata que acepte al vaciar la pila sustituyendo la última transición por:

$$\delta(q_1, \epsilon, Z_0) = (q_2, \epsilon)$$

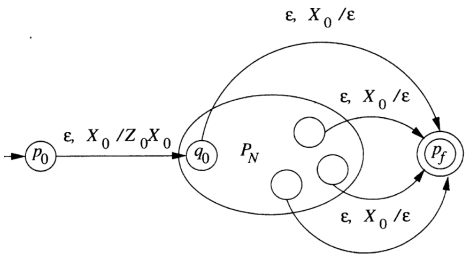
Equivalencia de aceptación por estado final y por vaciar pila

Teorema

Sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ un autómata de pila que acepta el lenguaje $L = N(P)$ al vaciar la pila, entonces existe un autómata de pila P_F que acepta L por estado final.

Construcción del autómata: Sea $X_0 \notin \Gamma$ un símbolo nuevo (no se encuentra en P, y tómese este X_0 como símbolo de pila inicial de P_F.
Considérese un nuevo símbolo inicial p_0 para P_F y un final p_f. Se introducirán las transiciones:

- $\delta_F(p_0, \epsilon, X_0) = (q_0, Z_0 X_0)$
- Para todo $q \in Q$ (en donde la pila esté vacía), se tiene $\delta_F(q, \epsilon, X_0) = (p_f, \epsilon)$
- Todas las demás trnasiciones permanecen igual.



Demostración de la construcción

Los autómatas son equivalentes: Mostraremos que $w \in N(P)$ si y sólo si $w \in L(P_F)$.
Si $w \in N(P)$, entonces $(q_0, w, Z_0) \vdash_P^* (q, \epsilon, \epsilon)$ con $q \in Q$. De aquí sabemos que

$$(q_0, w, Z_0X_0) \vdash_P^* (q, \epsilon, \epsilon X_0) = (q, \epsilon, \epsilon X_0)$$

por lo que por la construcción de P_F tenemos:

$$(p_0, w, X_0) \vdash_{P_F} (q_0, w, Z_0X_0) \vdash_{P_F}^* (q, \epsilon, X_0) \vdash_{P_F} (p_f, \epsilon, \epsilon)$$

Es decir, $w \in L(P_F)$.
Ahora, si $w \in L(P_F)$ tenemos que $(p_0, w, X_0) \vdash_{P_f} (q_0, w, Z_0X_0) \vdash_{P_F}^* (q, \epsilon, X_0) \vdash_{P_f} (p_f, \epsilon, \epsilon)$. Pero por la construcción de P_F sabemos que (q, ϵ, X_0) implica (q, ϵ, ϵ) en P. Por lo que tenemos

$$(q_0, w, Z_0) \vdash_P^* (q, \epsilon, \epsilon)$$

Por lo que $w \in N(P)$.

Equivalencia de aceptaciones

Teorema

Si $L = L(P_F)$ es un lenguaje aceptado por un autómata de pila P_F por estado final, entonces existe un autómata P que acepta L al vaciar la pila.

De esta forma, preferiremos el uso de autómatas de pila que acepten por estado final.

Autómatas de pila y lenguajes libres de contexto

Autómatas de pila y CFGs

Hasta ahora hemos visto que para los lenguajes libres de contexto:

1. Son descritos por una gramática libre de contexto $G = (\Sigma, \Delta, S, R)$ con reglas de la forma $X \to \gamma$.
2. Existe una gramática en forma normal simplificada.
3. Existe una gramática en forma normal de Chomsky.
4. Existe una gramática en forma normal de Greibach.

Mostraremos ahora que:

1. Para todo lenguaje $L(G)$ producido por una gramática libre de contexto existe un autómata de pila P tal que $L(G) = L(P)$.
2. Para todo autómata de pila P con lenguaje $L(P)$ existe una gramáica libre de contexto G tal que $L(P) = L(G)$.

Formas sentenciales izquierdas

Forma sentencial izquierda

Dada una CFG $G = (\Sigma, \Delta, S, R)$, una forma sentencial izquierda es una cadena no terminal $xA\alpha$ donde $x \in \Sigma^*$ es terminal, $\alpha \in \Sigma^* \cup \Delta^*$ y $A \in \Delta$.

En la forma sentencial izquierda, llamamos cola a la cadena $A\alpha$.
Si $A\alpha = \epsilon$, entonces $xA\alpha = x \in \Sigma^*$ por lo que se tiene una cadena terminal.
La configuración de un autómata de pila:

$$(q, y, A\alpha)$$

representa una forma sentencial izquierda donde $x = wy$. Si $\alpha \in \Delta^*$ podemos pensar en una regla dentro de una gramática en forma normal de Greibach.

Construcción de PDA a partir de CFG

Algorithm Algoritmo para construir autómata de pila

1: **procedure** $\mathrm{PDA}(G = (\Sigma, \Delta, S, R))$
2: $Q, q_0 \leftarrow \{q\}, q$
3: $\Gamma \leftarrow \Sigma \cup \Delta$
4: $Z_0 \leftarrow S$
5: **for** $A \in \Delta$ **do**
6: $\delta(q, \epsilon, A) \leftarrow \{(q, \beta); A \to \beta \in R\}$ (variables)
7: **end for**
8: **for** $a \in \Sigma$ **do**
9: $\delta(q, a, a) \leftarrow (q, \epsilon)$ (terminales)
10: **end for**
11: **return** $P = \{Q, \Sigma, \Gamma, \delta, q_0, F, \{\}\}$
12: **end procedure**

Ejemplo

Supóngase la siguiente gramática libre de contextos:

$$E \to I|E \cdot E|E + E|(E)$$
$$I \to a|b|Ia|Ib$$

1. $\Gamma = \{a, b, 0, 1, (,), +, \cdot, I, E\}$ con $Z_0 = E$
2. Las transiciones del autómata están dadas como:

$$\delta(q, \epsilon, E) = \{(q, I), (q, E \cdot E), (q, E + E), (q, (E))\} \qquad \delta(q, \epsilon, I) = \{(q, a), (q, b), (q, Ia), (q, Ib)\}$$
$$\delta(q, a, a) = (q, \epsilon) \qquad \delta(q, b, b) = (q, \epsilon)$$
$$\delta(q, (, () = (q, \epsilon) \qquad \delta\big(q,),)\big) = (q, \epsilon)$$
$$\delta(q, +, +) = (q, \epsilon) \qquad \delta(q, \cdot, \cdot) = (q, \epsilon)$$

Ejemplo de aceptación

Con el autómata construido supóngase que se tiene la cadena:

$$a \cdot (a + b)$$

Entonces, tenemos la siguiente descripción:

$$\begin{aligned}(q, a \cdot (a + b), E) &\vdash (q, a \cdot (a + b), E \cdot E) \vdash (q, a \cdot (a + b), I \cdot E) \vdash (q, a \cdot (a + b), a \cdot E) \\ &\vdash (q, \cdot(a + b), \cdot E) \vdash (q, (a + b), E) \vdash (q, (a + b), (E)) \\ &\vdash (q, a + b), E)) \vdash (q, a + b), E + E)) \vdash (q, a + b), I + E) \\ &\vdash (q, a + b), a + E)) \vdash (q, +b), +E)) \vdash (q, b), E)) \\ &\vdash (q, b), I)) \vdash (q, b), b)) \vdash (q,),)) \\ &\vdash (q, \epsilon, \epsilon)\end{aligned}$$

Por lo que acepta la cadena al vaciar la pila.

Existencia del autómata

Teorema

Dada una gramática libre de contexto G, el autómata de pila P es construido por el procedimiento anterior, entonces $L(G) = L(P)$.

Supongamos $w \in L(G)$, entonces demostramos que el autómata obtiene las formas sentenciales por inducción sobre la derivación:

$$S = \gamma_1 \Longrightarrow_{lm} \cdots \Longrightarrow_{lm} \gamma_n = w$$

Base: Si $i = 1$, $\gamma_1 = S$ deriva entonces la derivación es $\epsilon = w$, por lo que $(q, w, S) \vdash^* (q, \epsilon, S)$.
Inducción: Supóngase $(q, w, S) \vdash^* (q, y_i, \alpha_i)$, donde α_i es una cola, por lo que $\alpha_i = A\beta_i$, $A \in \Gamma$. El paso $\gamma_i \Longrightarrow_{lm} \gamma_{i+1}$ es de la forma $A\beta_i \Longrightarrow_{lm} \rho\beta_i$, t.q. $A \in \rho \in R$.
Por la asignación de variables en P, A se reemplaza por ρ en el tope de la pila. De esta forma se tiene la configuración $(q, y_{i+1}, \gamma_{i+1})$ que representa la forma sentencial $y_i y_{i+1} \gamma_{i+1}$ con γ_{i+1} cola. Por lo que $w \in L(P)$.

Demostración (continuación)

Demostraremos ahora que si en el autómata $(q, x, A) \vdash^* (q, \epsilon, \epsilon)$, entonces $A \Longrightarrow_{lm}^* x$ por inducción sobre las configuraciones:

Base: Si $(q, x, A) \vdash (q, \epsilon, \epsilon)$ entonces $x = \epsilon$ y tenemos las reglas de la forma $A \to \epsilon \in R$ y por tanto $A \Longrightarrow_{lm} \epsilon$.

Inducción: Supóngase que P toma $n > 1$ configuraciones. El primer movimiento será de la forma en que A se reemplaza por el cuerpo de las producciones en el tope de la pila. Esto es, $(q, x, A) \vdash (q, x, Y_1 Y_2 \cdots Y_k)$ tal que $A \to Y_1 Y_2 \cdots Y_k$.

Los siguientes $n - 1$ movimientos de P consumen x y sacan los símbolos Y_i de la pila. Si $x = x_1 x_2 \cdots x_k$ t.q. x_i se consume al sacar Y_i.

Así, para todo $i = 1, .., k$ $(q, x_1 \cdots x_k, Y_i) \vdash^* (q, x_{i+1} \cdots x_k, \epsilon)$, y ya que esto se da en $n - 1$ movimientos, por h.i., tenemos que $Y_i \Longrightarrow_{lm}^* x_i$.

Por lo tanto, $A \Longrightarrow_{lm}^* Y_1 \cdots Y_k \Longrightarrow_{lm}^* x_1 \cdots x_k = x$. Si tomamos $A = S$, vemos que una cadena w aceptada por el autómata al vaciar la pila es generada por la gramática. Esto es $w \in L(G)$.

Símbolos de la CFG a partir de PDA

Convertir un autómata de pila en una gramática requiere generar símbolos de la gramática. Estos símbolos partirán de los elementos del autómata:

- Se tomará en cuenta el símbolo que es sacada de la pila en una configuración.
- Se toma en cuenta el cambio de estado de p a q cuando el símbolo X es sacado de la pila.

Representaremos los símbolos de la gramática como $[pXq]$; se trata de un sólo símbolo que busca indicar los estados del autómata y los símbolos de la pila.

Construcción de CF a partir de PDA

Algorithm Algoritmo para construir una gramática

procedure GRAMMAR($P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$)
  $\Delta \leftarrow \{S\} \cup \{[pXq] : p, q \in Q, X \in \Gamma\}$
  for $p \in Q$ **do**
    $R \leftarrow S \rightarrow [q_0 Z_0 p]$
  end for
  for $\delta(q, a, X) = (r, Y_1 \cdots Y_k) \in \delta, a \in \Sigma$ **do**
    for $r_i \in Q, i =$ **to** $k - 1$ **do**
      $R \leftarrow [qXr] \rightarrow a[rY_1r_1][r_1Y_2r_2] \cdots [r_{k-1}Y_kr]$
    end for
  end for
  return $G = \{\Sigma, \Delta, S, R\}$
end procedure

Ejemplo

Considérese el autómata de pila $P = (\{q\}, \{i, e\}, \{Z\}, \delta, q, Z, \{\})$ con las transiciones:

$$\delta(q, 0, Z) = (q, ZZ)$$
$$\delta(q, 1, Z) = (q, \epsilon)$$

1. S es inicial, y $\Gamma = \{qZq\}$.
2. Las reglas son:

$$S \to [qZq]$$
$$[qZq] \to 0[qZq][qZq]$$
$$[qZq] \to 1$$

Esta gramática se puede abreviar como $S \to 0SS|1$ que produce el lenguaje $L(G) = \{0^n1^n : n > 0\} = L(P)$.

Correción de conversión de PDA a CFG

Teorema

Sea $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ un autómata de pila y $G = (\Sigma, \Delta, S, R)$ la gramática construida por el algoritmo anterior. Entonces $L(P) = L(G)$.

Supóngase que $w \in L(P)$. Entonces $(q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$. Por inducción:
Base: Si se da en un paso, entonces $\delta(q_0, w, Z_0) = (p, \epsilon)$ por lo que $w = a \in \Sigma \cup \{\epsilon\}$. Así que por construcción de G, $[q_0 Z_0 p] \to a$, esto es $S \Longrightarrow [q_0 Z_0 p] \Longrightarrow a = w$, por tanto $w \in L(G)$.
Inducción: Si $(q_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$ en n pasos, entonces $(q_0, w, Z_0) \vdash (r_0, x, Y_1 \cdots Y_k) \vdash^* (p, \epsilon, \epsilon)$ con $w = ax$.
Por construcción de G, tenemos la regla $[q_0 Z_0 p] \to a[r_0 Y_1 r_1] \cdots [r_{k-1} Y_k r_k]$, tal que cada símbolo Y_i es sacado de la pila con r_i.
Dividiendo $w = w_1 \cdots w_k$ tal que w_i es consumido al sacar Y_i, esto es: $(r_{i-1}, w_i, Y_i) \vdash^* (r_i, \epsilon, \epsilon)$. Por h.i. $[r_{i-1} Y_i r_i] \Longrightarrow_G^* w_i$, de tal forma que: $S \Longrightarrow [q_0 Z_0 r_k] \Longrightarrow a[r_0 Y_1 r_1] \cdots [r_{k-1} Y_k r_k] \Longrightarrow^* w_1 \cdots w_k = w$. Por lo que $w \in L(G)$.

Demostración (continuación)

Supóngase ahora que $w \in L(G)$, demostramos por inducción sobre los pasos de la derivación que también está en el lenguaje del autómata.

Base: Supóngase que se da la derivación en 1 paso. Tal que $[qXp] \Longrightarrow w$, por lo que $[qXp] \rightarrow w$ y $w \in \Sigma \cup \{\epsilon\}$. Entonces, en el autómata tenemos la transición $\delta(q, w, Z_0) = (p, \epsilon)$ y por tanto se da $(q, w, Z_0) \vdash (q, \epsilon, \epsilon)$ que acepta a w al vaciar la pila.

Inducción: Supóngase que la derivación $[qZ_0p] \Longrightarrow^* w$ en n pasos. Utilizando la primera forma sentencial y $w = aw_1w_2 \cdots w_k$, tenemos que $(p = r_k)$:

$$[qZ_0p] \Longrightarrow a[r_0Y_1r_1] \cdots [r_{k-1}Y_kp] \Longrightarrow^* w$$

Siendo que $[r_{i-1}Y_ir_i] \Longrightarrow^* w_i$, que son derivaciones en menos de n pasos. Por h.i. entonces tenemos que para el autómata $(r_{i-1}, w_i, Y_i) \vdash^* (r_i, \epsilon, \epsilon)$...

Demostración (continuación)

... Utilizando el teorema de equivalencia de configuraciones, tenemos que.

$$(r_{i-1}, w_i w_{i+1} \cdots w_k, Y_i Y_{i+1} \cdots Y_k) \vdash^* (r_i, w_{i+1} \cdots w_k, Y_{i+1} \cdots Y_k)$$

Pero ahora podemos poner estas secuencias juntas:

$$\begin{aligned}(q, Z_0, aw_1 \cdot w_k, Z_0) &\vdash (r_0, Z_0, w_1 \cdot w_k, Y_1 \cdots Y_k) \\ &\vdash^* (r_1, w_2 \cdots w_k, Y_2 \cdots Y_k) \\ &\vdash^* (r_{k-1}, w_k, Y_k) \\ &\vdash^* (p, \epsilon, \epsilon)\end{aligned}$$

Es decir, tenemos que $(q, Z_0, w, Z_0) \vdash^* (p, \epsilon, \epsilon)$, por lo que $w \in L(P)$.

Lema de Ogden

Profundidad de un árbol sintáctico

Teorema (profundidad de CNF)

Sea $G = (\Sigma, \Delta, S, R)$ una CFG en Forma Normal de Chomsky, y sea $w \in L(G)$ una cadena producida por un árbol T. Si la longitud del camino más largo en el árbol (su profundidad) es n, entonces:

$$|w| \leq 2^{n-1}$$

Demostraremos el teorema por inducción sobre la profundidad del árbol n.

Base: Si la profundidad del árbol sintáctico es $n = 1$, entonces el árbol cuenta sólo con una raíz y con un hijo hoja. Por construcción del árbol, existe la regla:

$$S \to w$$

Con $w \in \Sigma$, pues la gramática está en CFN, por lo que S sólo puede derivar un símbolo. Por tanto, notamos que:

$$|w| = 1 \leq 2^{1-1} = 1$$

Demostración (continuación)

Inducción: Supóngase que la profundidad del árbol es $n > 1$. Sean A y B símbolos no terminales tales que $S \to AB \in R$ (por CFN), entonces el árbol tiene como raíz a S y como hijos a A y B.
Considérense los subárboles determinados por A y B, entonces la profundidad de A y B debe ser a lo más $n-1$. Siendo $w = xy$ tal que x es producido por A y y producido por B, por hipótesis de inducción, sabemos que:

$$|x| \leq 2^{n-2} \ , \ |y| \leq 2^{n-2}$$

Por tanto, tenemos que para w se da:

$$|w| = |x| + |y| \leq 2^{n-2} + 2^{n-2} = 22^{n-2} = 2^{n-1}$$

Lema de Ogden

Al lema de Ogden también se le conoce como lema del bombeo (pumping lemma) para lenguajes libres de contexto.

Teorema (Lema de Ogden)

Sea L un lenguaje libre de contexto, con gramática $G = (\Sigma, \Delta, S, R)$. Entonces, existe una constante n que depende de L, tal que si $z \in L$ con $|z| \geq n$, entonces z puede separarse en 5 subcadenas:

$$z = uvwxy$$

Tal que éstas están sujetas a las siguientes condiciones:

1. $|vwx| \leq n$
2. $v \neq \epsilon$ y $x \neq \epsilon$ (subcadenas no vacías que pueden bombearse).
3. $\forall i \geq 0$, se tiene que $uv^i wx^i y \in L$.

Demostración

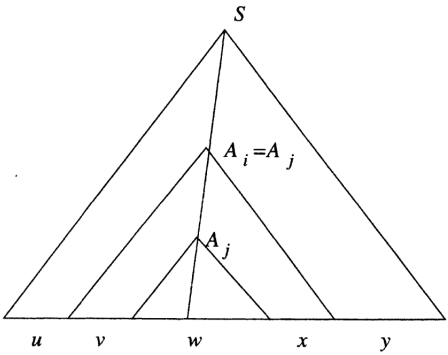
Gramática: Supóngase que $L \neq \emptyset$ y $L \neq \{\epsilon\}$ (pues de otra forma se cumple trivialmente). Y supóngase que la gramática $G = (\Sigma, \Delta, S, R)$ está en Forma Normal de Chomsky (de otra forma, se genera esta gramática). Por tanto, $L(G) = L \setminus \{\epsilon\}$.

Construcción de constante: Sea $m = |\Delta|$ y tomamos $n = 2^m$.

Elección de cadena z: Sea la cadena $z \in L$ tal que $|z| \geq n$; por lo tanto, por el teorema anterior, cualquier árbol que produzca la cadena z con un camino de longitud m, tiene una profundidad de $2^{m-1} = \frac{n}{2}$.
Ya que el árbol en CNF es binario $\#hojas = n \leq 2^{n-1}$. Pero ya que $|z| \geq n$, la profundidad no puede ser $\frac{n}{2}$, por lo que debe ser que la longitud del camino sea al menos $m + 1$.

Demostración (continuación)

Repetición de variables: Tomemos el valor $k \geq m + 1$ el camino de mayor longitud en el árbol. Confórmese este camino por las variables $A_0, A_1, A_2, ..., A_k$



Ya que $k > m = |\Delta|$, entonces debe haber al menos dos variables en el camino que se repitan. Sean estas variables $A_i = A_j$, con $k - m \leq i < j < k$ ($i \neq j$.

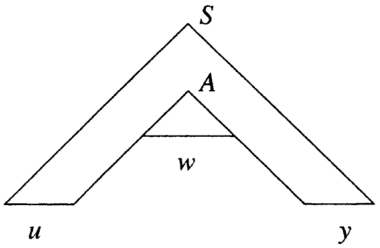
Subcadenas: Entonces divídase z en las siguientes subcadenas: w es producido por subárbol con raíz A_j; v, x producidos a la izquierda y derecha de w.

Demostración (continuación)

$\mathbf{v}, \mathbf{x} \neq \epsilon$: Esto se debe a que la gramática en CNF no tiene producciones unitarias, y como $i \neq j$, entonces $v \neq \epsilon \neq x$.

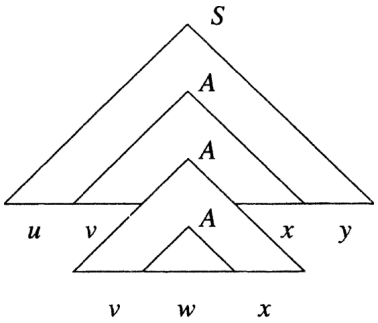
Longitud de subcadena: Tomando A_i, debe observarse que tenemos que $k - i \leq m$; es decir, el camino partiendo de A_i que deriva en vwx tiene longitud m, por lo que por el teorema anterior tendremos que $|vwx| \leq 2^m = n$.

Bombeo 0 veces: Para poder ver que se puede realizar el bombeo, quítese del árbol el subárbol con raíz en A_i y sustitúyase con el árbol con raíz en A_j, lo que es posible puesto que $A_i = A_j$. La cadena resultante es $uwy = uv^0wx^0y$.



Demostración continuación

Bombeo varias veces: Ahora bien, para bombear v y x en la cadena, ya que $A_i = A_j$, con $i < j$, podemos substituir el árbol con raíz en A_j por una copia del árbol con raíz en A_i. Por lo tanto, se obtendrá la cadena uv^2wx^2y. Si se repite este proceso dentro del nuevo subárbol, veremos que v y x se bombean, tal que para toda $r \geq 0$ se tendrá uv^rwx^ry.



Ejemplo

Considérese la siguiente gramática en Forma Normal de Chomsky definida por las producciones:

$$S \to XA|YB|XY$$
$$A \to SX$$
$$B \to SY$$
$$X \to 0$$
$$Y \to 1$$

Tomando $n = 5$, para la cadena 010110 podemos ver que la división de la cadena es como sigue:

- $u = 0$, $y = 0$
- $v = 1$, $x = 1$ y $w = 01$ tal que $|vwx| = 4 \leq 5$
- Se pueden bombear tal que $01^i011^i0 \in L(G)$ para toda $i \geq 0$.

Uso del lema de Ogden para descartar lenguajes

El lema de Ogden se puede usar para demostrar que un lenguaje no es libre de contexto. Para esto, dado el lenguaje L, seguiremos los siguientes pasos:

1. Escoger una n constante, que puede variar según el lenguaje.
2. Seleccionar alguna cadena z con longitud mayor a n y para la que posiblemente no se cumpla el lema de Ogden.
3. Separar en subcadenas $z = uvwxy$ tal que se cumplan las restricciones $|vwx| \leq n$ y $v \neq \epsilon \neq x$.
4. Si encontramos alguna i tal que se cumple que $uv^iwx^iy \notin L$, concluimos que L no es libre del contexto.

Ejemplo

El lenguaje L siguiente no es libre de contexto:
$L = \{a^n b^n c^n : n \geq 1\}$
Tómese como n la n de las exponenciales de los símbolos. Y sea $z = a^n b^n c^n$ dividiéndose en $uvwxy$ con $|vwx| \leq n$ y $v, x \neq \epsilon$.
Claramente vwx no puede contener a's y $c's$ al mismo tiempo, pues como tenemos que b^n está en medio, no se cumpliría la restricción. Dos casos:

1. Si vwx no contiene c's, entonces $vx = a^k b^m$, $k, m > 1$, por lo que $uwy = a^{n-k} b^{k-m} c^n \notin L$.
2. Si vwx no contiene a's, entonces $vx = b^k c^m$ y por tanto $a^n b^{n-k} c^{n-m} \notin L$.

En cualquier caso $uv^i wx^i$ no está en L para toda i, por lo que L no es libre de contexto.

Substituciones

Substitución

Sea Σ el alfabeto y $a \in \Sigma$ un símbolo del alfabeto. Una substitución es una función $s : \Sigma \to \mathcal{P}(\Sigma^*)$, que asigna un lenguaje a un símbolo:

$$s(a) = L_a$$

Homomorfismo en substituciones

Dada una cadena $w = a_1 a_2 \cdots a_n \in \Sigma^*$, la substitución sobre w se define como.

$$s(w) = s(a_1)s(a_2) \cdots s(a_n)$$

Sustitución de lenguajes

Substitución de lenguajes

Dado un lenguaje L sobre el alfabeto Σ, la substitución de L es el lenguaje:

$$s(L) = \bigcup_{w \in L} s(w)$$

Sea $\Sigma = \{0, 1\}$ y definamos las sustituciones de estos símblolos como sigue:

$$s(0) = \{a^n b^n : n \geq 1\} \qquad s(1) = \{aa, bb\}$$

Entonces $s(01) = s(0)s(1) = \{a^n b^n aa\} \cup \{a^n b^n bb\}$
Y si $L = [[0^*]]$ entonces tendremos que:

$$s(L) = \bigcup_{n=0} s(0)^n = \{a^{n_1} b^{n_1} a^{n_2} b^{n_2} \cdots a^{n_k} b^{n_k}\}$$

Resultados sobre lenguajes libres de contexto

Teorema

Sea L un lenguaje libre de contexto sobre Σ. Si s es una sustitución tal que para todo $a \in \Sigma$, $s(a)$ es libre de contexto, entonces $s(L)$ es libre de contexto.

Teorema

Los lenguajes libres de contexto son cerrados bajo unión.

Para mostrar esto, sean L_1, L_2 dos CFLs. Si tomamos $L = \{1, 2\}$ tal que $s(1) = L_1$ y $s(2) = L_2$, entonces $s(L) = L_1 \cup L_2$ es un lenguaje libre de contexto.
Para extenderlo, si tomamos $L = \{1, 2, ...\}$ (que puede ser infinito numerable, y asociamos $s(i) = L = i$, tenemos que:

$$\bigcup_{i=!} L_i$$

es libre de contexto.

Cerradura de CFLs

Teorema

Los lenguajes libres de contexto son cerrados bajo concatenación.

Teorema

Los lenguajes libres de contexto son cerrados bajo la operación de estrella de Kleene.

Proposición

Si L es un lenguaje libre de contexto, su reverso L^R también es libre de contexto.

Intersección de CFLs

Sea $L_1 = \{a^n b^n, c^m : n, m \geq 1\}$ y $L_2 = \{a^m b^n c^n : m, n \geq 1\}$ dos lenguajes libres de contexto. Su intersección

$$L_1 \cap L_2 = \{a^n b^n c^n\}$$

no es un lenguaje libre de contexto.

Teorema

Sea L un lenguaje libre de contexto y R en lenguaje regular, entonces:

1. $L \cap R$ es un lenguaje libre de contexto.
2. $L \setminus R$ es lenguaje libre de contexto.