Capítulo 10.

Gramáticas independientes del contexto

10.1. Conceptos generales

Definición, Motivación.

10.2. Simplificación de GIC

Reglas innecesarias, Símbolos inaccesibles, Símbolos superfluos, Reglas no generativas, Reglas unitarias.

10.3. Formas normales

Forma normal de Chomsky, Forma normal de Greibach.

10.1. Conceptos generales

Chomsky: $G_3 \subset \mathbf{G_2} \subset G_1 \subset G_0$

<u>Definición</u>

Σ_N - un conjunto de símbolos no terminales (variables)
Σ_T - un conjunto de símbolos terminales
P - un conjunto de reglas de producción
$S \in \Sigma_N$ un símbolo no terminal de Σ_N

La cuaterna (Σ_T, Σ_N , S, P) se llama *gramática independiente del contexto* (*Gramáticas de tipo 2*, según la clasificación de Chomsky) si todas las reglas de producción de P tienen la forma:

$$A ::= \beta, \text{ siendo } A \in \Sigma_N \text{ y } \beta \in \Sigma^*$$

Todo lenguaje generado por una GIC G (se denota por L(G)) se llama *Lenguaje independiente del contexto.*

Ejemplos:
G_1={{a,b},{S},S,{S::=aSb | ab}}
G_2={{a,b},{S},S,{S::=aSbb | abb}}
G_3={{a,b},{S},S,{S::=a | bS }}
G_4={{a,b},{S},S,{S::=aSb | SS | λ}}
G_5={{a,b},{S},S,{S::=aS | Sb | a | b}}

Motivación

Representación de la sintaxis de lenguajes de programación; descripción de la estructura de lenguajes de marcado (DTD en XML, …); generadores de compiladores (Yacc, …), etc.

Java Language Specification.

http://java.sun.com/docs/books/jls/html/index.html

ForStatement:
 for (*ForInit*$_{\text{opt}}$; *Expression*$_{\text{opt}}$; *ForUpdate*$_{opt}$) *Statement*

ForStatement:
 for (; ;) *Statement*
 for (; ; *ForUpdate*) *Statement*
 for (; *Expression* ;) *Statement*
 for (; *Expression* ; *ForUpdate*) *Statement*
 for (*ForInit* ; ;) Statement
 for (*ForInit* ; ; *ForUpdate*) *Statement*
 for (*ForInit* ; *Expression* ;) *Statement*
 for (*ForInit* ; *Expression* ; *ForUpdate*) *Statement*

```
for (z = 0; z < edges[k].length; ++z)
       if (edges[k][z] == i)
           break search;
```

Ada95 Reference Manual

http://www.adahome.com/rm95/

```
if_statement ::=
   if condition then
      sequence_of_statements
  {elsif condition then
      sequence_of_statements}
  [else
      sequence_of_statements]
   end if;
```

```
if Line_Too_Short then
    raise Layout_Error;
elsif Line_Full then
    New_Line;
    Put(Item);
elsif Line_Empty then
    ….
else
    Put(Item);
end if;
```

10.2. Simplificación de Gramáticas independientes del contexto

Justificación

Sea la siguiente GIC G definida por sus reglas:

S::= Aa | B | D
B ::= b
A ::= A | Aa | bA | B | cE
C ::= abd
E ::= λ
D::=Db

En G se pueden observar las siguientes redundancias:

1. La regla A::=A es innecesaria.
2. Del símbolo D no se pueden derivar sentencias (símbolo superfluo).
3. Del símbolo C se puede derivar la sentencia abd, pero no es accesible desde S (símbolo inaccesible).
4. Las reglas E ::= λ y A ::= cE podrían ser reemplazadas por la regla A ::= c (regla no generadora).
5. El símbolo B podría ser eliminado, y la regla B::= b podría ser reemplazada por las reglas S ::= b y A ::= b (regla de redenominación).

Si eliminamos estas redundancias en la gramática G, obtendremos una GIC G' tal que L(G) = L(G'):

S::= Aa | b
A ::= Aa | bA | b | c

Reglas innecesarias

Una regla de la forma A::=A es innecesaria y puede ser eliminada.

Ejemplo A::=A

Símbolos superfluos (o no generadores)

Un símbolo superfluo es un símbolo no terminal A tal que no existe una derivación $A \rightarrow^{*} w$, donde $w \in \Sigma_T^{*}$.

Ejemplo D (D::=Db)

Algoritmo de eliminación de símbolos superfluos

Sea la GIC $G = (\Sigma_T, \Sigma_N, S, P)$. Transformaremos G en $G' = (\Sigma_T, \Sigma'_N, S, P')$ de forma que L(G) = L(G'). Construimos iterativamente el nuevo Σ'_N como sigue:

Inicializar Σ'_N a $\emptyset$

Repetir

Añadir a Σ'_N todo no terminal A para el cual existe

$A ::= w \in P$ y $w \in (\Sigma_T \cup \Sigma'_N)^*$.

Hasta que no se puedan añadir más símbolos a Σ'_N.

3. Asignar a P' todas las reglas $p \in P$ cuyos símbolos pertenezcan a $\Sigma_T \cup \Sigma'_N$

4. Si $S \notin \Sigma'_N$, añadir S a Σ'_N

Ejemplo:

S::= Aa | B | D
B ::= b
A ::= Aa | bA | B | cE
C ::= abd
E ::= λ
D::=Db

G'={{a,b,c,d},Σ'_N,S,P'}

1. Inicializar:

$\Sigma'_N=\emptyset$; P'=$\emptyset$

2. Añadir símbolos no terminales:

Σ'_N={B}	(por B::=b)
Σ'_N={B,C}	(por C::=abd)
Σ'_N={B,C,E}	(por E::=λ)
Σ'_N={B,C,E,S}	(por S::=B)
Σ'_N={B,C,E,S,A}	(por A::=B)

3. Añadir reglas a P':

P'={ S::= Aa | B
B ::= b
A ::= Aa | bA | B | cE
C ::= abd
E ::= λ }

Símbolos inaccesibles

Un símbolo X (terminal o no terminal) será inaccesible si no existe ninguna derivación $S \to^* \alpha X \beta$ tal que $\alpha, \beta \in (\Sigma_T \cup \Sigma_N)^*$.

Ejemplo: C (C::=abd)

Algoritmo de eliminación de símbolos inaccesibles

Sea la GIC $G = (\Sigma_T, \Sigma_N, S, P)$. Transformaremos G en $G' = (\Sigma'_T, \Sigma'_N, S, P')$ de forma que $L(G) = L(G')$. Construimos iterativamente los nuevos Σ'_T, Σ'_N y P' como sigue :

1. Inicializar Σ'_N de forma que contenga el axioma S, e inicializar P' y Σ'_T a $\emptyset$.

2. Repetir

    Para $A \in \Sigma'_N$, y reglas $A ::= w \in P$:

    2.1. Introducir A::=w en P'.
    2.2. Para todo no terminal B de w, introducir B en Σ'_N.
    2.3. Para todo terminal a de w, introducir a en Σ'_T.

Hasta que no se puedan añadir nuevas reglas a P'.

Ejemplo:

$S ::= Aa \mid B$
$B ::= b$
$A ::= Aa \mid bA \mid B \mid cE$
$C ::= abd$
$E ::= \lambda$

$G' = \{\Sigma'_T, \Sigma'_N, S, P'\}$

1. Inicializar:

$\Sigma'_N = \{S\}$; $P' = \varnothing$; $\Sigma'_T = \varnothing$

2. Añadir reglas y símbolos:

$P' = \{S ::= Aa|B\}$; $\Sigma'_N = \{S,A,B\}$; $\Sigma'_T = \{a\}$
$P' = \{S ::= Aa|B,\ A ::= Aa|bA|B|cE\}$; $\Sigma'_N = \{S,A,B,E\}$;
$\Sigma'_T = \{a,b,c\}$
$P' = \{S ::= Aa|B,\ A ::= Aa|bA|B|cE,\ B ::= b\}$; $\Sigma'_N = \{S,A,B,E\}$;
$\Sigma'_T = \{a,b,c\}$

$\rightarrow$ $P' = \{S ::= Aa|B,$
$A ::= Aa|bA|B|cE,$
$B ::= b,\ E ::= \lambda\}$;
$\Sigma'_N = \{S,A,B,E\}$; $\Sigma'_T = \{a,b,c\}$

Los dos algoritmos vistos hasta el momento deben ser aplicados en el orden en que han sido expuestos, ya que si no, los resultados pueden no ser los deseables:

$S ::= AB \mid a$; $A ::= a$

1. Inacc. $S ::= AB \mid a$; $A ::= a$	1. Superf.: $S ::= a$; $A ::= a$
2. Superfl. $S ::= a$; $A ::= a$	2. Inacc.: $S ::= a$

Reglas no generativas (reglas λ)

Dado una gramática $G = (\Sigma_T , \Sigma_N , S, P)$, se dice que una regla es no generativa si tiene la forma $A::=\lambda$, siendo $A \in \Sigma_N$. Los símbolos A, tales que $A \rightarrow^* \lambda$, se denominan *anulables*.

Ejemplo: $E::= \lambda$ (regla no generativa), E (símbolo anulable)

Teorema:
Dado una gramática $G = (\Sigma_T , \Sigma_N , S, P)$, existe una gramática $G' = (\Sigma_T , \Sigma_N' , S, P')$ equivalente a G sin reglas no generativas excepto la regla $S::=\lambda$.

Existe un algoritmo para eliminar las reglas no generativas.

Idea:

$S::= Aa$	$\rightarrow$	$S::= Aa \mid a$
$A ::= Aa \mid b \mid \lambda$		$A ::= Aa \mid b \mid a$

Algoritmo para la eliminación de las reglas no generativas:

Sea la GIC $G = (\Sigma_T , \Sigma_N , S, P)$. Transformamos G en $G' = (\Sigma_T, \Sigma_N , S, P')$ de forma que $L(G) = L(G')$.

1. Obtención de los símbolos anulables en G (conjunto S_A).
   1.1. $S_A=\{ A| A\in\Sigma_N$ y $(A::= \lambda)\in P\}$
   1.2. Repetir para todas $B ::= w$ en P con $w\in\Sigma_N^*$:
      Si $w\in S_A^*$ (solo tiene símbolos anulables), entonces $S_A=S_A\cup\{B\}$.
   Hasta que no se añadan más símbolos no terminales a S_A.

2. Creación de G':
   2.1. $P'=\emptyset$
   2.2. Para cada regla $B ::= x_1 x_2 \ldots x_n$ de P ($x_1 x_2 \ldots x_n \in\Sigma^*$) se construyen todas las reglas posibles de la forma $B ::= y_1 y_2 \ldots y_n$ donde las y_i satisfagan:
      $y_i = x_i$ si x_i no es anulable ($x_i\notin S_A$).
      $y_i = x_i$ o λ si x_i es anulable ($x_i\in S_A$).
   2.3. De estas reglas se eliminan los que tienen la forma $B::= \lambda...\lambda$ y de las reglas restantes se eliminan los λ. Las reglas resultantes se incluyen en P'.

      Es decir, para $B::= x_1 x_2 x_3$ con $x_2, x_3\in SA$ se generan las reglas: $B::= x_1 x_2 \lambda \mid x_1 \lambda x_3 \mid x_1 \lambda \lambda \mid x_1 x_2 x_3$
      $\Rightarrow P'=P'\cup\{B::= x_1 x_2 \mid x_1 x_3 \mid x_1 \mid x_1 x_2 x_3\}$

   2.4. Si $S\in S_A$ entonces $P'=P'\cup\{S::= \lambda\}$

Ejemplo: Sea G una gramática definida por las siguientes reglas:

S::= Aa | B	B ::= bB | b | λ
A ::= Aa | bA | BEE	E ::= λ

1. Obtención de los símbolos anulables en G (conjunto SA).

SA={B,E} (por B ::= λ y E ::= λ)
SA={B,E,S} (por S ::= B)
SA={B,E,S,A} (por A ::= BEE)

2. Creación de G':

S::= Aa	⇒ S::= Aa | λa	⇒ S::= Aa | a
S::= B	⇒ S::= B | λ	⇒ S::= B
B ::= bB	⇒ B::= bB | bλ	⇒ B::= bB | b
B ::= b	⇒ B::= b	⇒ B::= b
B ::= λ	⇒ B ::= λ	⇒ (se elimina)
A ::= Aa	⇒ A::= Aa | λa	⇒ A::= Aa | a
A ::= bA	⇒ A::= bA | bλ	⇒ A::= bA | b
A ::= BEE	⇒ A::= BEE |BλE |BEλ |λEE |Bλλ |λλE |λEλ|λλλ	
		⇒ A::= BEE | BE | EE | B | E
E::= λ	⇒ E::= λ	⇒ (se elimina)

3. Se añade la regla S::= λ porque S∈SA

⇒ P'={S::=Aa | a | B |λ
B::=bB | b
A::=Aa | a | bA | b | BEE | BE | EE | B | E}

Como se ve fácilmente, ahora existe la regla A::=E y E es un símbolo superfluo.

⇒ Después de realizar el algoritmo **hay que eliminar símbolos superfluos**.

Reglas unitarias o de redenominación

Son reglas de la forma A ::= B, siendo A, B$\in\Sigma_N$.

Ejemplo: S::=B o A::=B

Teorema:

Dado una gramática G = (Σ_T , Σ_N , S, P), existe una gramática G'= (Σ_T , Σ_N' , S, P') equivalente a G sin reglas unitarias.

Ejemplo: Sea G una gramática definida por las siguientes reglas:

$\Rightarrow$ S::=Aa | a | C | E | CE   C::=B
B::=bB | b   A::=Aa | a | bA | b | B
E::=c | λ

Algoritmo (simple) para eliminar reglas unitarias

Repetir:
Para cada regla unitaria A::=B:
Sean B::= w_1 | w_2 | ... | w_n todas las reglas de B.
Substituye A::=B por las reglas A::=w_1 | w_2 | ... | w_n
Hasta que no haya más reglas unitarias.

(**Nota:** Este algoritmo solo funciona si no hay derivaciones unitarias cíclicas de la forma: A$\rightarrow$B$\rightarrow$ ... $\rightarrow$ A)

Resultado del ejemplo:

$\Rightarrow$ S::=Aa | a | bB | b | c | λ | CE   C::=bB | b
B::=bB | b   A::=Aa | a | bA | b | bB
E::=c | λ

Algoritmo (general) para eliminar reglas unitarias

(funciona para todos los casos)

Sea la GIC $G = (\Sigma_T , \Sigma_N , S, P)$. Transformamos G en $G' = (\Sigma_T, \Sigma'_N , S, P')$ de forma que $L(G) = L(G')$.

Para cada $A \in \Sigma_N$, se define el conjunto:
Unitario(A) = $\{B \in \Sigma_N \mid A \to^* B$ usando sólo reglas unitarias$\}$

1. Inicializar $P' = \varnothing$.
2. Para cada variable A y cada $B \in$ Unitario (A):
   Para cada regla no unitaria B ::= w de P,
   añadir A ::= w a P'.

Nota: Este algoritmo también elimina reglas innecesarias.

¿Cómo encontrar el conjunto unitario?

Ejemplo: Sea G una gramática definida por las siguientes reglas:
$\Rightarrow$ S::=Aa | a | C | E | BE; A::=Aa | a; C::=B;
B::=bB | b | S; E::=c | λ

S $\to$ C $\to$ B $\to$ S (ya en la lista)
S $\to$ E

Ejemplo: Sea G una gramática definida por las siguientes reglas:

$\Rightarrow$ S::=Aa | a | C | E | BE; A::=Aa | a; C::=B;
B::=bB | b | S; E::=c | λ

Unitario(S)={S, C, E, B}
Unitario(A)={A}
Unitario(B)={B, S, C, E}
Unitario(C)={C, B, S, E}
Unitario(E)={E}

Crear reglas de P':

S:	S::=Aa | a | BE	(por S)
	nada por C	
	S::=c | λ	(por E)
	S::=bB | b	(por B)
B:	B::=bB | b	(por B)
	B::=Aa | a | BE	(por S)
	nada por C	
	B::=c | λ	(por E)
C:	nada por C	
	C::=bB | b	(por B)
	C::=Aa | a | BE	(por S)
	C::=c | λ	(por E)
E:	E::=c | λ	(por E)
A:	A::=Aa | a	(por A)

Resultado: $\Rightarrow$
S::= Aa | a | BE | c | λ | bB | b
A::= Aa | a
B::= Aa | a | BE | c | λ | bB | b
C::= Aa | a | BE | c | λ | bB | b
E::= c | λ

NOTAS:

- Se ve fácilmente que C es un símbolo inaccesible.
- Para evitar nuevas reglas λ, se debe aplicar el algoritmo después de eliminar las reglas no generativas. Si la regla S::= λ está en P, se aplica el algoritmo sin esta regla y al final se vuelve a añadir a P'.

Gramática limpia

Una gramática G = (Σ_T , Σ_N , S, P) se dice *limpia* si no contiene símbolos inaccesibles, símbolos superfluos, ni reglas innecesarias.

Gramática bien formada

Una gramática G = (Σ_T , Σ_N , S, P) se dice *bien formada* si:

1. es limpia,
2. no contiene reglas no generativas (reglas λ) salvo en el axioma
3. no contiene reglas unitarias (o reglas de redenominación)
4. en caso de que contenga la regla S::= λ, también la gramática G'= (Σ_T , Σ_N , S, P') con P'=P – { S::= λ} está bien formada.

Ejemplo:

G={{a,b},{S},S,{S::=SS | aSb | λ} no está bien formada.

$\Rightarrow$ Aplicar los algoritmos.

Algoritmo para obtener una gramática bien formada

Aplicar los algoritmos para:

1. (Eliminar reglas innecesarias)
2. Eliminar reglas no generativas (reglas λ)
3. Eliminar reglas unitarias

   Si la gramática del paso 2 contiene la regla S::= λ, entonces aplicar el algoritmo sin esta regla y añadirla al final del proceso.
4. Eliminar símbolos superfluos
5. Eliminar símbolos inaccesibles

10.3. Formas normales

Forma Normal de Chomsky

Toda GIC se puede transformar en una nueva GIC G' equivalente a G, expresada en Forma Normal de Chomsky (FNC). En esta forma, las reglas pueden tener las siguientes formas:

1. A::= BC
2. S ::= λ
3. A ::= a

Donde A, B, C $\in \Sigma_N$, a $\in \Sigma_T$ y S es el axioma.

Los árboles de derivación de una GIC en FNC serán árboles binarios, lo cual facilitará la implementación de los analizadores sintácticos.

Ejemplo:

S::= AB | aSb | aAB | aB | aA | Bb | a | b | λ
A::= aAB | aB | aA | a
B::= Bb | b

⇒ (dos nuevos símbolos no terminales)

S::= AB | CSD | CAB | CB | CA | BD | a | b | λ
A::= CAB | CB | CA | a
B::= BD | b C::= a D::= b

⇒ (nuevos símbolos para casos como CSD)

S::= AB | CE | CF | CB | CA | BD | a | b | λ
A::= CF | CB | CA | a
B::= BD | b C::= a D::= b
E::= SD F::= AB

Algoritmo para obtener la FNC

Dada una GIC $G_0 = (\Sigma_{T0}, \Sigma_{N0}, S, P_0)$, vamos a obtener una GIC $G' = (\Sigma'_T, \Sigma'_N, S, P')$ en FNC equivalente a G.

1. Convertimos G_0 en GIC bien formada $G = (\Sigma_T, \Sigma_N, S, P)$.
2. $\Sigma'_N=\Sigma_N$, $P'=P$
3. Sustitución de símbolos terminales
   - 3.1. Para cada símbolo $a\in \Sigma_T$ distinto que aparezca en el cuerpo de una regla de P' se añade una regla C::=a a P', siendo $C\notin\Sigma'_N$ una variable nueva que se añade a Σ'_N.
   - 3.2. En cada regla A ::= x de P' con $x\in\Sigma^*$ y $|x|>1$ se sustituyen todos los símbolos terminales en x por la variable correspondiente del paso anterior.
4. Sustitución de reglas del tipo A::=x con $x\in\Sigma_N^*$ y $|x|> 2$:

   Repetir:

   Para cada regla A::=x en P' con $x\in\Sigma_N^*$ y $|x|> 2$:

   Sea $x=x_1x_2y$ con $x_1,x_2\in\Sigma_N$ e $y\in\Sigma_N^+$
   - Añade una regla $D::= x_1x_2$ a P' siendo $D\notin\Sigma'_N$ una variable nueva que se añade a Σ'_N
   - Sustituye la regla A::=x en P' por la regla A::=Dy

   Hasta que no hay más reglas A::=x en P' con $x\in\Sigma_N^*$ y $|x|>2$

Si en algún momento del algoritmo, se desea añadir una regla D::= u con $D\notin\Sigma_N$ y $u\in\Sigma^*$, y ya existe una regla de la forma B ::= u con $B\in\Sigma'_N$, y no existe ninguna regla más en P' cuya parte izquierda sea B, entonces no es necesario añadir el nuevo símbolo D ni tampoco la nueva regla D ::= u.

Ejemplo:

Sea la siguiente gramática: G = ({a,b,c} , {S,A,B} , S, P) con
P={ S::= Aba | a | BbA | λ
A::= aab | Acbc
B::=Ac }

1. G es una gramática bien formada
2. $\Sigma'_N=\Sigma_N$, P'=P
3. P'={ S::= ACD | a | BCA | λ;
A::= DDC | AECE;
B::=AE; C::=b; D::=a; E::=c }
Σ'_N={S,A,B,C,D,E}
4. (primer ciclo)
S::=ACD $\Rightarrow$ S::=FD y F::=AC
S::=BCA $\Rightarrow$ S::=GA y G::=BC
A::=DDC $\Rightarrow$ A::=HC y H::=DD
A::=AECE $\Rightarrow$ A::=BCE (ya existe B::=AE)
$\Rightarrow$P'={ S::= FD | a | GA | λ;
A::= HC | BCE;
B::=AE; C::=b; D::=a; E::=c;
F::=AC; G::=BC; H::=DD}
Σ'_N={S,A,B,C,D,E,F,G,H}

(segundo ciclo)
A::=BCE $\Rightarrow$ A::=GE (ya existe G::=BC)

$\Rightarrow$P'={ S::= FD | a | GA | λ;
A::= HC | GE;
B::=AE; C::=b; D::=a; E::=c;
F::=AC; G::=BC; H::=DD}
Σ'_N={S,A,B,C,D,E,F,G,H}

Forma Normal de Greibach

Toda GIC G se puede transformar en una nueva GIC G' equivalente a G, expresada en Forma Normal de Greibach (FNG). En esta forma, las reglas pueden tener las siguientes formas:

1. A::= aX
2. S ::= λ

Donde $A \in \Sigma_N$, $a \in \Sigma_T$, $X \in \Sigma_N^*$ y S es el axioma.

Como se verá más delante, esta representación será útil para construir el Autómata a Pila asociado a una GIC.

Ejemplo:

S::= AB | λ
A::= aA | bB | b
B::= b

$\Rightarrow$ Sustitución de las reglas con A en S::=AB

S::= aAB | bBB | bB | λ
A::= aA | bB | b
B::= b

¿Y que pasa con reglas de tipo A::=Abc | a?

Regla recursiva a izquierdas

Se llama regla recursiva a izquierdas a la que tiene la forma A::= Ax, donde $x \in \Sigma^*$.

Lema

“Toda gramática independiente del contexto puede reducirse a otra equivalente sin reglas recursivas a izquierdas”

Ejemplo:

A::=Ab | ACD | bDC | a |

Posible derivación desde A:
$A \rightarrow ACD \rightarrow AbCD \rightarrow AbbCD \rightarrow ACDbbCD \rightarrow bDCCDbbCD$

Cada derivación podría comenzar con: bDC | a y seguir con la repetición 0 o n veces de: b | CD.

La parte correspondiente a la repetición de b | CD se podría obtener con (reglas no recursivas a izquierdas):

B::= bB | CDB | b | CD

y juntando con A se obtiene la parte del comienzo de las derivaciones de A:

A::=bDCB | aB | bDC | a

El conjunto de las reglas para A y B obtiene las mismas derivaciones que la gramática inicial.

Método de demostración:

Sea G = (Σ_T , Σ_N , S, P) una GIC con reglas recursivas a izquierdas, donde P contiene reglas de la forma:

$$A ::= Ax_1 \mid Ax_2 \mid \dots \mid Ax_n \mid y_1 \mid y_2 \mid \dots\dots \mid y_m$$

x_i, $y_i \in \Sigma^*$, A no es el primer símbolo de ningún y_i

Se construye una G' = (Σ_T , $\Sigma_N \cup \{B\}$, S, P'), donde P' = P sin reglas de la forma A := Ax_i y con las siguientes reglas:

$$A ::= y_1B \mid y_2B \mid \dots\dots \mid y_mB \mid y_1 \mid y_2 \mid \dots\dots \mid y_m$$
$$B ::= x_1B \mid x_2B \mid \dots \mid x_nB \mid x_1 \mid x_2 \mid \dots \mid x_n$$

Demostraremos que L(G)=L(G').
Cualquier derivación de G que parta de A tendrá la siguiente forma:

$$A \to Ax_{i1} \to Ax_{i2}x_{i1} \dots \to y_kx_{in} \dots x_{i2}x_{i1}$$

y se puede obtener con G':

$$A \to y_kB \to y_kx_{in}B \to y_kx_{in}x_{in-1}B \dots \to y_kx_{in} \dots x_{i2}x_{i1}$$

Lo mismo es cierto para cualquier derivación de G' respecto a G. Como G y G' se diferencian solo en las producciones que afectan a A se sigue que L(G')=L(G).

Obtención de la FNG

Tipos de reglas:

Tipo 0:	A::=aAB | b	→ ya esta en FNG
Tipo 1:	A::=aBc | bc	→ A::=aBC | bC, C::=c
Tipo 2:	A::=Aa | c	→ A::=cE | c, E::=aE | a
Tipo 3:	A::=Ba, B::=bD | aB	→ A::=bDa | aBa, B::=bD | aB (algunos tipo 1)
Tipo 4:	A::=Ba, B::=Aa|c	→ A::=Aaa | ca , B::=Baa | c (algunos tipo 2 o 1)

Tipo 5: (más complicado)

P={A::=Ba | a, B::=Cb , C::=Ac} →

A::=Cba | a, B::=Acb , C::=Bac | ac →

A::=Bacba | acba | a, B::=Cbacb | acb, C::=Acbac | ac

(No se consigue por esta vía.)

Solución: (establer orden A<B<C)

1. tratar reglas X::=Yw con X>Y:

A::=Ba | a, B::=Cb, C::=Ac → (C>A)

... C::=Bac | ac → (C>B)

... C::=Cbac | ac →

... C::=acF | ac, F::=bacF | bac (tipo 1)

2. tratar reglas X::=Yw con X<Y: (de mayor a menor X)

A::=Ba | a, B::=Cb, C::=acF | ac, F::=bacF | bac

2.1. tratar B: (B>A)

... B::=Cb → ... B::=acFb | acb (tipo 1)

2.2. tratar A:

... A::=Ba|a → ... A::=acFba | acba | a (tipo 1)

→ Resultado:

A::=acFba | acba | a B::=acFb | acb

C::=acF | ac F::=bacF | bac

Sustitución

Sea $G=(\Sigma_T, \Sigma_N, S, P)$ una gramática y sea la regla $R=(A::= yBx) \in P$ con $A,B \in \Sigma_N$ y $x,y \in \Sigma^*$. Sean $B::= w_1|...|w_m$ todas las reglas de P cuya parte izquierda es B.
Se llama *sustitución del símbolo B en la regla R* a la acción de eliminar la regla A::= yBx de P y de incluir en este conjunto las reglas $A ::= yw_1x \mid ... \mid yw_mx$.

Ejemplo:
Sea G una gramática definida por las siguientes reglas:

A::=Ba (axioma) B::=CA | a
C::=AB | b

Sustitución de B en A::=Ba
$\Rightarrow$ A::= CAa | aa (axioma) B::=CA | a
C::=AB | b

Lema: Dada una GIC $G=(\Sigma_T, \Sigma_N, S, P)$, la gramática $G'=(\Sigma_T, \Sigma_N, S, P')$ que se obtiene de la sustitución de cualquier símbolo $B \in \Sigma_N$ en cualquier regla que contiene B en su cuerpo, es equivalente a G.

Método de demostración:

Sea $G=(\Sigma_T, \Sigma_N, S, P)$ una GIC y sea $B\in\Sigma_N$. Sea $A::= yBx$ con $A\in\Sigma_N$ y $x,y\in\Sigma^*$ una regla en P.

Caso 1: No existen reglas $B::= w_1|...|w_m$

$\Rightarrow P'=P - \{A::= yBx\}$

En este caso B es un símbolo superfluo y, por tanto, la regla A::= yBx puede ser eliminada sin que cambie el lenguaje representado.

Caso 2: Existen reglas $B::= w_1|...|w_m$

$\Rightarrow P'=(P - \{A::= yBx\})\cup \{A ::= yw_1x \mid ... \mid yw_mx\}$

Cualquier derivación de G que parte de A utilizando la regla A::= yBx tiene la forma:

$$A\to yBx\to^* vBu\to vw_ku$$

y se puede obtener con G':

$$A\to yw_kx \to^* vw_ku$$

Por otra parte, cualquier derivación de G' que parte de A y utiliza una de las reglas nuevas tiene la forma:

$$A\to yw_kx$$

y se puede obtener con G:

$$A\to yBx\to yw_kx$$

Como G y G' se diferencian sólo en las producciones que afectan a A y B se sigue que L(G')=L(G).

Algoritmo para obtener la FNG

Dada una GIC $G_0 = (\Sigma_{T0}, \Sigma_{N0}, S, P_0)$, vamos a obtener una GIC $G' = (\Sigma'_T, \Sigma'_N, S, P')$ en FNG.

1. Convertimos G_0 en una GIC bien formada $G=(\Sigma_T, \Sigma_N, S, P)$.
2. Eliminamos $S::=\lambda$ de G.
3. Eliminamos la recursividad a izquierdas en G.
4. Establecemos una ordenación entre todos los símbolos no terminales de Σ_N: $A_1 < A_2 < \ldots < A_n$
5. Clasificamos las reglas de P en tres grupos:
   a) Reglas de la forma $A_i ::= ax$, donde $a \in \Sigma_T$, $x \in \Sigma^*$ (ya comienzan con un símbolo terminal)
   b) Reglas de la forma $A_i ::= A_j x$, donde $i < j$, $x \in \Sigma^*$
   c) Reglas de la forma $A_i ::= A_j x$, donde $i > j$, $x \in \Sigma^*$

6. Repetir (Eliminación de las reglas del tercer grupo)
   6.1. Seleccionamos la regla $A_i ::= A_j x$ ($x \in \Sigma^*$) del tercer grupo tal que la posición de A_i en la ordenación de Σ_N es mínima y sustituimos A_j en esta regla.
   6.2. Si en las nuevas reglas existe alguna regla del tercer grupo, entonces a cada una de estas se le aplica de nuevo el paso 6.1.
   6.3. Si en las nuevas reglas existen reglas recursivas a izquierdas, las eliminamos. Cada nuevo símbolo no terminal será añadido a Σ_N y colocado al principio de la ordenación de Σ_N.

Se repite el paso 6 hasta que no quedan reglas del tercer grupo.

7. Eliminación de las reglas del segundo grupo:

   Para cada A_i de Σ_N (de mayor a menor según el orden definido):

   7.1. Para cada regla R de la forma $A_i ::= A_k x$ ($x \in \Sigma^*$) sustituimos A_k en R.

   7.2. Si en las nuevas reglas existe alguna regla del segundo grupo, entonces a cada una de estas reglas se le aplica de nuevo el paso 7.1.

8. Convertir las reglas del primer grupo en reglas en FNG:

   Para cada símbolo terminal $c \in \Sigma_T$:

   Si c aparece en el cuerpo de alguna regla (sin contar la primera posición):

   - Elige una variable nueva $B \notin \Sigma_N$, añade B a Σ_N y añade la regla B ::= c a P
   - Sustituye c en los cuerpos de todas las reglas de P (salvo en la primera posición)

9. Si en el paso 2. se ha eliminado la regla $S ::= \lambda$, entonces se vuelve a añadir esta regla a P.

10. G'=G

Importante:
Es importante que se realice los pasos 1 y 2 (la gramática debe ser bien formada y no debe tener ninguna regla λ).

Ejemplo:
A::=Ba (axioma) B::=CA | a
C::=AB | b

1. 2. y 3. ya esta bien formada y no tiene reglas recursivas a izquierdas

4. Orden A<B<C:

5. Grupo 1: B::= a , C::= b
   Grupo 2: A::=Ba , B::=CA
   Grupo 3: C::=AB

6. (eliminación de reglas del grupo 3):
   C::=AB $\Rightarrow$ C::=BaB (del grupo 3; otra sust.)
   C::=BaB $\Rightarrow$ C::=CAaB | aaB
   (eliminación de reglas recursivas a izquierdas)
   C::= CAaB | aaB | b $\Rightarrow$
   C::=aaBD | bD | aaB | b D::=AaBD | AaB
   (se añade D en la primera posición en el orden: D<A<B<C)

$\Rightarrow$ Gramática sin reglas del grupo 3 ni reglas recursivas a la izquierda:
A::=Ba
B::=CA | a
C::=aaBD | bD | aaB | b
D::=AaBD | AaB

7. (eliminación de reglas del grupo 2 desde mayor a menor)
C (ya esta)
B::=CA ⇒ B::= aaBDA | bDA | aaBA | bA
A::=Ba ⇒ A::= aaBDAa | bDAa | aaBAa | bAa | aa
D::=AaBD ⇒ D::= aaBDAaaBD | bDAaaBD | aaBAaaBD | bAaaBD | aaaBD
D::=AaB ⇒ D::= aaBDAaaB | bDAaaB | aaBAaaB | bAaaB | aaaB

⇒ Gramática sin reglas de los grupos 3 y 2 ni reglas recursivas a la izquierda:
A::= aaBDAa | bDAa | aaBAa | bAa | aa
B::= aaBDA | bDA | aaBA | bA | a
C::= aaBD | bD | aaB | b
D::= aaBDAaaBD | bDAaaBD | aaBAaaBD | bAaaBD
| aaaBD
| aaBDAaaB | bDAaaB | aaBAaaB | bAaaB | aaaB

8. Gramática en FNG (sustitución de símbolos terminales):
A::= aEBDAE | bDAE | aEBAE | bAE | aE
B::= aEBDA | bDA | aEBA | bA | a
C::= aEBD | bD | aEB | b
D::= aEBDAEEBD | bDAEEBD | aEBAEEBD | bAEEBD
| aEEBD
| aEBDAEEB | bDAEEB | aEBAEEB | bAEEB | aEEB
E::= a

9. G no tiene S::=λ

10. G’=G