Gramáticas: Ambigüedad. Formas normales.

Teoría Computacional - ESCOM - IPN - Prof. Rafael Aguilar

Contenido

- Ambigüedad
  - En sentencias
  - En gramáticas
  - En lenguajes
  - Ejemplo
- Métodos de transformación de gramáticas
  - Conceptos.
  - Forma normal de Chomsky
  - Forma normal de Greibach

Ambigüedad

- El concepto de ambigüedad en teoría de lenguajes y gramáticas es similar al empleado en el lenguaje coloquial: **existe más de una forma de generar una palabra** a partir del axioma de una gramática.
- La ambigüedad puede surgir en varios niveles: en **sentencias**, **lenguajes**, y **gramáticas**.
- Es importante que no exista ambigüedad en el uso de lenguajes y gramáticas pues provoca que **el análisis de las sentencias no sea determinista**.

Ambigüedad

- **Sentencia**: una sentencia es ambigua si tiene más de una derivación o árbol de derivación.

  Ejemplo: $G_1 = \{ \{1\}, \{A, B\}, A, \{(A ::= 1B), (A ::=11), (B ::=1)\} \}$

  La sentencia 11 puede ser obtenida por dos derivaciones:

  $A \rightarrow 1B \rightarrow 11$

  $A \rightarrow 11$

- **Gramática**: una gramática es ambigua si tiene al menos una sentencia ambigua.

  Ejemplo: La gramática G_1 es ambigua, ya que 11 es una sentencia ambigua

Ambigüedad

- **Lenguaje**

- Un lenguaje es ambiguo si existe una gramática ambigua que lo genera.

  Ejemplo: El lenguaje L = {11} es ambiguo ya que la gramática

  G_1 lo genera y es ambigua.

- Si todas las gramáticas que generan un lenguaje son ambiguas, el **lenguaje es inherentemente ambiguo**.

  Ejemplo: El lenguaje L1 no es inherentemente ambiguo ya que

  la siguiente gramática lo genera y no es ambigua.

  $G_2 = \{ \{1\}, \{A\}, A, \{(A ::= 11)\})$

Ambigüedad

- Una gramática es ambigua cuando es posible construir **dos o más árboles de derivación diferentes** para una misma palabra.
- El problema de la ambigüedad es complejo debido a que **no existe ningún algoritmo que permita reconocer si una gramática es o no ambigua**, y en el caso de que lo sea, tampoco existe ningún algoritmo que permita eliminar dicha ambigüedad.

Ambigüedad. Ejemplo

- Un ejemplo clásico de gramática ambigua se presenta en la definición de las **expresiones aritméticas** que aparecen comúnmente en los lenguajes de programación.
- El siguiente ejemplo simplificado permite definir expresiones en las que intervienen las cuatro operaciones aritméticas básicas con operandos que pueden ser identificadores (***id***) o constantes (***cte***). Llamaremos a esta gramática: **G_Exp_0**.

G_Exp_0:

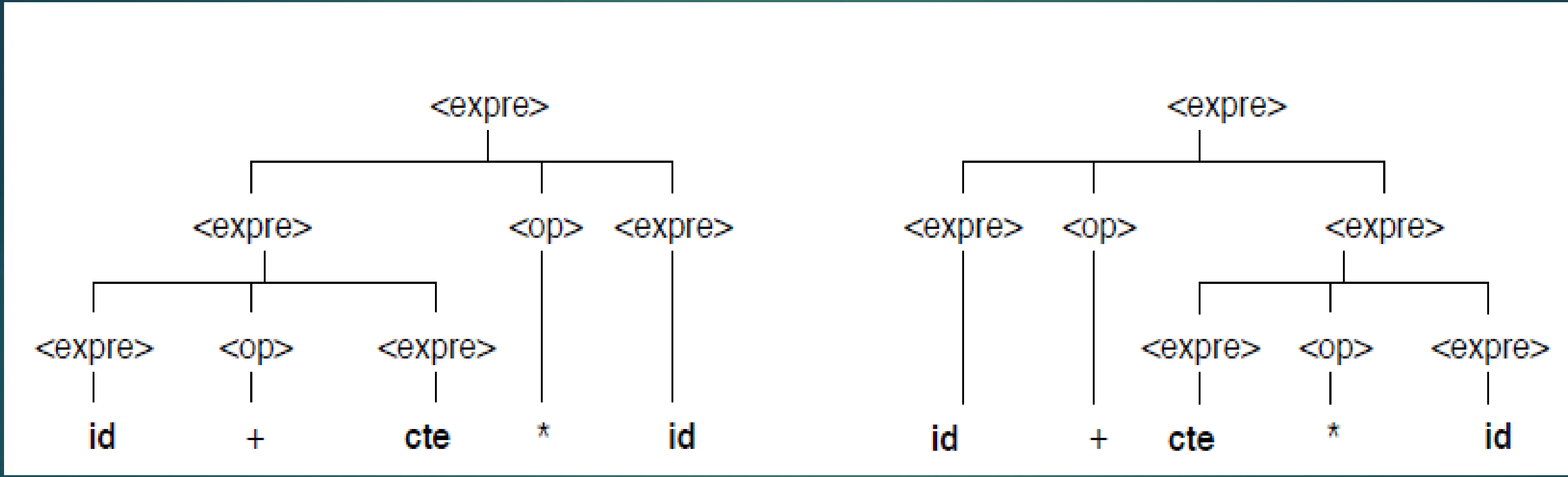
$$\Sigma_T = \{id, cte, (,), +, -, *, /\}$$

$$\Sigma_N = \{< expre >, < op >\}$$

$$S = < expre >$$

$$P = \left\{ \begin{array}{l} < expre > ::= < expre >< op >< expre > \\ \qquad\qquad\quad |(< expre >) \\ \qquad\qquad\quad |id \\ \qquad\qquad\quad |cte \\ < op > ::= + \\ \qquad\qquad\quad |- \\ \qquad\qquad\quad |* \\ \qquad\qquad\quad |/ \end{array} \right\}$$

Es fácil demostrar que ésta gramática es ambigua mediante la construcción de dos árboles diferentes para generar la misma expresión: **id + cte * id**



Aunque las expresiones de cada árbol son correctas, una vez realizada la operación tendremos resultados distintos.
Para resolver este **caso de ambigüedad** se requiere establecer una jerarquía en los operadores.

La multiplicación y la división tienen mayor prioridad que la suma y la resta; la asociatividad será de izquierda a derecha. Para definir la jerarquía introducimos nuevos símbolos no terminales:

<termino> y **<op-adt>** estarán asociados a los operadores aditivos suma y resta.
<factor> y **<op-mult>** estarán asociados a los operadores multiplicación y división.

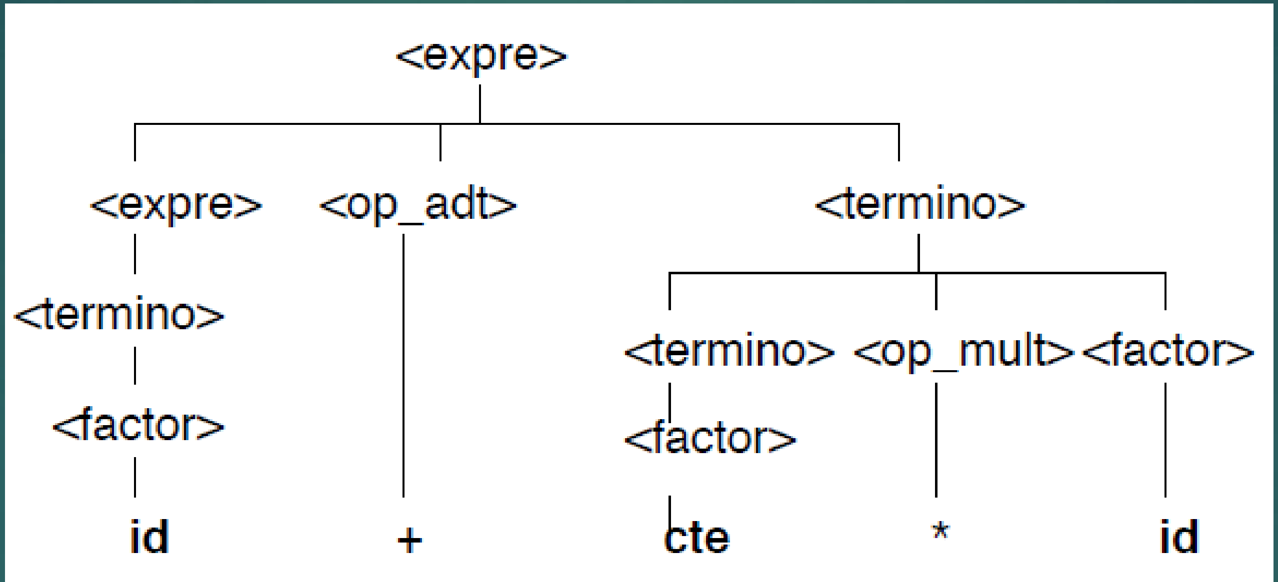
G_Exp_1:

$$\Sigma_N = \{< expre >, < termino >, < factor >, < op - adt >, < op - mult >\}$$

$$S = < expre >$$

$$P = \left\{ \begin{array}{l} < expre > ::= < expre >< op - adt >< termino > \\ \qquad\qquad\qquad | < termino > \\ < termino > ::= < termino >< op - mult >< factor > \\ \qquad\qquad\qquad | < factor > \\ < factor > ::= (< expre >) \\ \qquad\qquad\qquad | id \\ \qquad\qquad\qquad | cte \\ < op - adt > ::= + \\ \qquad\qquad\qquad | - \\ < op - mult > ::= * \\ \qquad\qquad\qquad | / \end{array} \right\}$$

Con esta nueva gramática tendremos ahora un único árbol de derivación:



Reconocedores

- Un **reconocedor** es un algoritmo que recibe como entrada una pabra $\mathbf{w} \, \varepsilon \, \Sigma_T^{+}$, examina sus símbolos de izquierda a derecha e intenta construir un árbol de derivación para dicha palabra.
- Con el proceso de construcción del árbol se obtiene además la estructura de la palabra, las producciones gramaticales que han de aplicarse y el orden en el que deben utilizarse.
- Un **reconocedor descendente** o **analizador sintáctico descendente** es un método de reconocimiento de palabras de un Lenguaje Independiente del Contexto (LIC) que se caracteriza porque **construye el árbol de derivación** de cada palabra de manera descendente: desde la raíz hasta las hojas.

Reconocedores

- Un **reconocedor ascendente** o **analizador sintáctico ascendente** es un método de reconocimiento de palabras de un Lenguaje Independiente del Contexto (LIC) que se caracteriza porque **construye el árbol de derivación** de cada palabra de manera ascendente: desde las hojas hasta la raíz.
- Un ejemplo de **reconocedor descendente** es el **LL(1)** o ***Left-Left(1)***. La primera **L** indica que la cadena se analiza de izquierda a derecha, la segunda **L** indica que en cada paso se construye la derivación por la izquierda, y el **1** indica que sólo es necesario un carácter para que el reconocedor decida qué producción debe utilizar en la construcción del árbol.
- Un ejemplo de **reconocedor ascendente** es el **LR(1)** o ***Left-Right(1)***. La primera **L** indica que la cadena se analiza de izquierda a derecha, la **R** indica que en cada paso se construye la derivación por la derecha, y el **1** indica que sólo es necesario un carácter para que el reconocedor decida qué producción debe utilizar en la construcción del árbol

Reconocedores

- Para el correcto funcionamiento de un reconocedor, es importante detectar y evitar determinados aspectos (defectos) de una GIC: **ambigüedad**, prefijos comunes, producciones inútiles, etc.
- Al proceso de transformar una gramática (con defectos) GIC en otra gramática GIC equivalente de forma que las producciones cumplan ciertos requisitos que faciliten la construcción de un reconocedor para dicha gramática, le denominamos **transformación de gramáticas**.

Métodos de transformación de gramáticas

- Los lenguajes generados por las **GIC** pueden ser reconocidos por los **autómatas de pila**. Tanto las GIC como los autómatas de pila permiten la descripción de la mayoría de los lenguajes de programación, por lo que **permiten la construcción de compiladores**.
- En este tema estudiaremos las técnicas para preparar una gramática a efectos de ser tratada eficientemente por un autómata que reconozca el lenguaje generado por la gramática.
- Abordaremos los conceptos de: **gramáticas limpias**, **bien formadas** y las **Formas Normales de Chomsky y Greibach**.

Métodos de transformación de gramáticas - Definiciones

- **Reglas innecesarias**

  Son las que tienen la forma **A ::= A**, **A ε Σ_N**.

  Estas reglas se deben eliminar de la gramática, ya que no generan derivaciones útiles.

- **Símbolos inaccesibles**

  Son aquellos símbolos no terminales **A ε Σ_N** que no pueden ser alcanzados por derivaciones desde el axioma de la gramática, **S**. Es decir, no hay una derivación **S** $\rightarrow^*$ **xAy**.

Métodos de transformación de gramáticas - Definiciones

- Ejemplo:

G_1=({0,1, 2, 3}, { A, B, C, D, E}, A, P)
P= { A → D0 | E10 | λ
B → 1C3
C → C
D → 1A
E → 1E
}
Los símbolos **B** y **C** son inaccesibles, y la regla **C→C** es innecesaria. Por lo tanto, la gramática G_1' es equivalente a G_1 eliminando dichos símbolos (con sus reglas asociadas) y la regla innecesaria:

G_1' =({0,1, 2, 3}, { A, D, E }, A, P)
P= { A → D0 | E10 | λ,
D → 1A,
E → 1E }

Métodos de transformación de gramáticas - Definiciones

- **Símbolos superfluos**: su definición depende del conjunto al que pertenezca:
  - **Símbolo no terminal superfluo**, **A**: es aquel del que sólo se pueden derivar palabras en las que existe al menos un símbolo no terminal, o, en notación: $\mathbf{A \nrightarrow^{*} x \ (x \ \varepsilon \ \Sigma^{*}_{T})}$.
  - **Símbolo terminal superfluo**, **a**: es aquel que no puede ser alcanzado por derivación desde el axioma; es decir, no existe ninguna producción $\mathbf{(A ::= \alpha\, a\beta) \ \varepsilon \ P}$.
  - Ejemplo: en la gramática G_1' los símbolos E, 2 y 3 son superfluos.

G_1' =({0,1, **2**, **3**}, { A, D, E }, A, P)

P= { A → D0 | E10 | λ,

D → 1A,

E → 1E }

Métodos de transformación de gramáticas - Definiciones

- **Procedimiento de eliminación de los símbolos superfluos de una gramática**, por ejemplo la gramática G_1' :
- **Primero**, se marcan los símbolos no terminales que estén en la parte izquierda de una producción y en cuya parte derecha sólo aparezcan símbolos terminales o λ. Por tanto, marcamos a **A**.

$G_1' = (\{0,1, 2, 3\}, \{ A, D, E \}, A, P)$

$P = \{ \mathbf{A} \rightarrow D0 \mid E10 \mid \boldsymbol{\lambda},$

$D \rightarrow 1A,$

$E \rightarrow 1E \}$

Métodos de transformación de gramáticas - Definiciones

- **Sucesivamente**, se van marcando los símbolos no terminales que aparezcan en la parte izquierda de producciones en las que sólo haya combinaciones de símbolos terminales, λ, o símbolos no terminales previamente marcados (**A**). Por lo tanto marcamos a **D**.

G_1' =({0,1, 2, 3}, { A, D, E }, A, P)

P= { **A** → D0 | E10 | **λ**,

D → 1**A**,

E → 1E }

- Como ya no se puede marcar ningún otro símbolo no terminal, los únicos **símbolos no terminales no superfluos** son **A** y **D**.

Métodos de transformación de gramáticas - Definiciones

- A continuación, se eliminan el resto (solo **E**), y por último, se eliminan los símbolos terminales que no aparezcan a la derecha de ninguna regla (**2** y **3**).

$G_1' = (\{0,1, \mathbf{2}, \mathbf{3}\}, \{ A, D, \mathbf{E} \}, A, P)$

$P = \{ \mathbf{A} \rightarrow D0 \mid \mathbf{E10} \mid \boldsymbol{\lambda},$

$\mathbf{D} \rightarrow 1\mathbf{A},$

$\mathbf{E} \rightarrow \mathbf{1E} \}$

- La nueva gramática equivalente será:

$G_1'' = (\{0, 1\}, \{A, D\}, A, \{ (A::=D0), (A::= \lambda), (D::=1A) \})$

Algoritmo:

Función Eliminar-Símbolos-Superfluos (G): G'

G: entrada a la función, gramática definida por $(\Sigma_T, \Sigma_N, S, \mathcal{P})$
G': salida de la función, gramática sin símbolos superfluos

$\Sigma_N$marcados:$=\emptyset$;
Repetir
    Si $A \in \Sigma_N$ y $A \notin \Sigma_N$marcados y
        $(A ::= x) \in \mathcal{P}$, $x \in (\Sigma_T^* \cup \Sigma_N^*$marcados$)$
    Entonces $\Sigma_N$marcados:$=\Sigma_N$marcados$\cup\{A\}$
Hasta que $(\Sigma_N = \Sigma_N$marcados$)$ ó
        no se marque ningún símbolo nuevo
Si todos los $A \in \Sigma_N$ están marcados (* $\Sigma_N = \Sigma_N$marcados *)
Entonces Devolver $G' = G$
Si no $\Sigma'_N := \Sigma_N$marcados; $\mathcal{P}' := \mathcal{P}$; $\Sigma'_T := \Sigma_T$;
    Para cada $A \notin \Sigma_N$marcados
        Borrar todas las produccíones de $\mathcal{P}'$ en las que aparezca A;
    Para cada $a \in \Sigma_T$
        Borrar a de Σ'_T si no existe en $\mathcal{P}'$ una regla $(A ::= \alpha a \beta)$
    Devolver $G' = (\Sigma'_T, \Sigma'_N, S, \mathcal{P}')$

Métodos de transformación de gramáticas - Definiciones

- **Gramática limpia**
  - Se dice que una gramática está limpia si no tiene reglas innecesarias, símbolos inaccesibles, ni símbolos superfluos.
  - Ejemplo: la gramática **G_1''** está limpia.
- **Reglas no generativas**
  - Una regla es no generativa cuando **$A ::= \lambda$, $A \neq S$**
  - Para eliminar dichas reglas podemos seguir el siguiente algoritmo:

Algoritmo: reglas no generativas

Función Eliminar-Reglas-no-Generativas (G): G'

G: entrada a la función, gramática definida por $(\Sigma_T, \Sigma_N, S, \mathcal{P})$
G': salida de la función, gramática sin reglas no generativas

---

$\mathcal{P}' := \mathcal{P};$
Repetir
    Para cada $P = (A ::= \lambda) \in \mathcal{P}'$ Y $(A \neq S)$
        $\mathcal{P}' := \mathcal{P}' - \{P\};$
        Para cada $P' = (B ::= xAy) \in \mathcal{P}'$ $(x, y \in \Sigma^*)$
            $\mathcal{P}' := \mathcal{P}' \cup \{(B ::= xy)\} - \{P'\}$
Hasta que todas las reglas sean generativas;
Devolver $G' = (\Sigma_T, \Sigma_N, S, \mathcal{P}')$

Ejemplo: dada la siguiente gramática

- G_2 =({0}, { A, B, C }, A, P)
  P= { A → C0B | λ
  B → BC | **λ**
  C → 0B | **λ** }

  Al aplicar el algoritmo tenemos las siguientes conversiones en P:
  Eliminación de la regla **B::= λ**
  P= { A → C0B | **C0** | λ
  B → BC | **C**
  C → 0B | **0** | λ }

  Eliminación de la regla **C::= λ**
  P= { A → C0B | **0B** | C0 | **0** | λ
  B → BC | C | **λ**
  C → 0B | 0 }

  En este paso, se habría generado la regla **B::=B** al reemplazar C en la regla B::= BC por λ. Esta regla al ser innecesaria se elimina.

Ejemplo: continuación

Eliminación de la regla **C::= λ**

P= { A → C0B | **0B** | C0 | **0** | λ
B → BC | C | **λ**
C → 0B | 0 }

Eliminación de la regla **B::= λ** que ha aparecido de nuevo:

P= { A → C0B | 0B | C0 | 0 | λ
B → BC | C
C → 0B | 0 }

La gramática quedaría como:

G_2' = ({0}, { A, B, C }, A, P)
P= { A → C0B | 0B | C0 | 0 | λ
B → BC | C
C → 0B | 0 }

Métodos de transformación de gramáticas - Definiciones

- **Reglas de redenominación**

- Una regla es de redenominación cuando **A ::= B** con **A**, **B ε** $\sum_N$. Para eliminarlas, se borra esa regla y se genera una nueva producción **A ::= x** por cada regla **B ::= x**, con **x ε** $\sum^*$

  Ejemplo: en la gramática ,
  $G_2' = (\{0\}, \{ A, B, C \}, A, P)$
  $P = \{ A \rightarrow C0B \mid 0B \mid C0 \mid 0 \mid \lambda$
  $B \rightarrow BC \mid C$
  $C \rightarrow 0B \mid 0 \}$

  La regla **B ::= C** es de redenominación. Al eliminar la regla y reemplazar C por las partes derechas de sus producciones se convierte la gramática en:
  $G_2'' = (\{0\}, \{ A, B, C \}, A, P)$
  $P = \{ A \rightarrow C0B \mid 0B \mid C0 \mid 0 \mid \lambda$
  $B \rightarrow BC \mid \mathbf{0B} \mid \mathbf{0}$
  $C \rightarrow 0B \mid 0 \}$

Métodos de transformación de gramáticas - Definiciones

- **Gramáticas bien formadas**

  Una gramática está bien formada si está limpia, no tiene reglas no generativas y no tiene reglas de redenominación.

  Ejemplo: la gramática **G_2''** está bien formada.

Formas normales de Gramáticas Independientes del Contexto.

- En algunas ocasiones es imprescindible que las gramáticas se hallen dispuestas de una forma especial. Es decir, se trata de obtener una gramática equivalente, que genera el mismo lenguaje, pero que debe cumplir ciertas especificaciones.
- Las dos formas normalizadas más frecuentes que se emplean en los lenguajes formales y sus aplicaciones son:
  - **Forma Normal de Chomsky (FNC)**
  - **Forma Normal de Greibach (FNG)**

Forma Normal de Chomsky (FNC)

- Las GIC se pueden transformar en gramáticas equivalentes expresadas en FNC
- En FNC, las producciones cumplen que:

P = { (A::= BC) ó (S::=λ) ó (A::=a) | A, B, C ε $\sum_N$, a ε $\sum_T$}

Forma Normal de Chomsky (FNC)

- Algoritmo para transformar una GIC a una gramática equivalente en FNC

  La idea principal consiste en sustituir las reglas de producción cuya parte derecha tiene más de dos símbolos, por varias reglas que tengan en la parte derecha dos símbolos no terminales o un símbolo terminal. Esto implica generar nuevos símbolos no terminales.

Ejemplo: Una GIC a FNC

G_3 = ({0, 1, 2}, { A, B, C }, A, P)
P= { A → CB2
A → 1B
A → λ
B → BC
B → 1
C → 2
}

Ejemplo: Una GIC a FNC

$P = \{ (A ::= BC) \text{ ó } (S ::= \lambda) \text{ ó } (A ::= a) \mid A, B, C \in \Sigma_N, a \in \Sigma_T \}$

$G_3 = (\{0, 1, 2\}, \{ A, B, C \}, A, P)$

$P = \{ A \rightarrow CB2$	No está en FNC
$A \rightarrow 1B$	No está en FNC
$A \rightarrow \lambda$	Está en FNC
$B \rightarrow BC$	Está en FNC
$B \rightarrow 1$	Está en FNC
$C \rightarrow 2$	Está en FNC
$\}$	

$A \rightarrow CB2$ } $D \rightarrow B2$ } $A \rightarrow CD$

$D \rightarrow B2$ } $D \rightarrow BC$

$C \rightarrow 2$

$A \rightarrow CD$ → Está en FNC

$D \rightarrow BC$ → Está en FNC

Ejemplo: Una GIC a FNC

P = { (A::= BC) ó (S::=λ) ó (A::=a) | A, B, C ε $\sum_N$, a ε $\sum_T$}

G_3 = ({0, 1, 2}, { A, B, C }, A, P)

P= { A → CB2 → No está en FNC
A → 1B → No está en FNC
A → λ → Está en FNC
B → BC → Está en FNC
B → 1 → Está en FNC
C → 2 → Está en FNC
}

A → 1B } E → 1 } A → EB }

E → 1 → Está en FNC

A → EB → Está en FNC

Ejemplo: Una GIC a FNC

P = { (A::= BC) ó (S::=λ) ó (A::=a) | A, B, C ε $\sum_N$, a ε $\sum_T$}

La gramática FNC resultante es:

G_3 = ({0, 1, 2}, { A, B, C, D, E }, A, P)

P= { A → CD → En FNC
A → EB → En FNC
A → λ → En FNC
B → BC → En FNC
B → 1 → En FNC
C → 2 → En FNC
D → BC → En FNC
E → 1 → En FNC
}

Ejercicio 1: pasar GIC a FNC

G_3 = ({a, b}, { S, U }, S, P)
P= { S → aUb
S → aab
S → b
U → aab
}

Ejercicio 2: pasar GIC a FNC

Sea la gramática G=(ΣN={S, A, B}, ΣT ={a, b}, P, S) cuyas producciones son:

$S \rightarrow bA \mid aB$

$A \rightarrow bAA \mid aS \mid a$

$B \rightarrow aBB \mid bS \mid b$

encontrar una gramática equivalente en FNC.