Tema 2
Gramáticas y Lenguajes Libres de Contexto

1. Definiciones Básicas
2. Gramáticas y Lenguajes Libres de Contexto
3. Forma Normal de Chomsky
4. Autómatas de Pila
5. Propiedades de los Lenguajes Libres de Contexto

1. Definiciones básicas

1. Introducción
2. Definición Formal de Gramática
3. Derivaciones
4. Sentencias, lenguaje generado por una gramática y gramáticas equivalentes

1. Definiciones básicas
1.1. Introducción

- **Gramática**
  - Permite definir un lenguaje mediante reglas que nos permiten generar o producir cadenas de un lenguaje.
  - Estas gramáticas son similares a las gramáticas de los lenguajes naturales, pero mucho más restrictivas y sencillas.
  - Un ejemplo de regla de una gramática:

    Oración → Sujeto predicado
  - Estas reglas se suelen llamar **reglas de reescritura**: el símbolo *Oración* se puede reescribir por el símbolo *Sujeto* seguido del símbolo *Predicado*.

1. Definiciones básicas
1.1. Introducción

Autómata:

Al igual que con los lenguajes regulares podemos definir un autómata como una máquina reconocedora de cadenas (palabras) de un determinado lenguaje.

Los autómatas con los que trabajaremos en este tema son algo más complejos que los AF.

1. Definiciones básicas
1.1. Introducción

- **UN EJEMPLO**

ORACIÓN → SUJETO PREDICADO | PREDICADO
SUJETO → ARTÍCULO NOMBRE
ARTICULO → el | la
NOMBRE → casa | niño
PREDICADO → VERBO COMPLEMENTO
VERBO → corre | es
COMPLEMENTO → bien | obediente | bonita

1. Definiciones básicas
1.2. Definición Formal de Gramática

- $G = (VN, VT, S, P)$ donde:
  - *VN (***vocabulario no terminal***):* conjunto finito de símbolos que permiten representar estados intermedios de la generación de las palabras del lenguaje
  - *VT* (**vocabulario terminal**): conjunto finito de los símbolos que forman las palabras del lenguaje.
  - $S \in VN$ *(***símbolo inicial** o **axioma***)*: a partir del que se aplican las reglas de la gramática para obtener las distintas palabras del lenguaje.
  - *P* es el conjunto de **reglas de producción** (**reglas de derivación o reglas de reescritura**) que permiten generar las palabras del lenguaje.

1. Definiciones básicas
1.2. Definición Formal de Gramática

Las reglas de reescritura son un par ordenado de cadenas, $\alpha, \beta \in (VN \cup VT)^*$, representado de la forma

$$\alpha \rightarrow \beta$$

- Esto significa que si α es una subcadena de una cadena p, entonces se puede sustituir α por β en p.

- En general, las reglas son de la forma: $\alpha \rightarrow \beta$, con

$$\alpha \in (VN \cup VT)^+$$
$$\beta \in (VN \cup VT)^*$$

1. Definiciones básicas
1.2. Definición Formal de Gramática

Un ejemplo para ciertas expresiones aritméticas

***G* = (*VN*, *VT*, *S*, *P*)**

- **VT={ +, -, \, *, (,), id }**
- **VN= { E}**
- **S=E**
- **P:**

1. E→E+E
2. E→E- E
3. E→(E)
4. E→ E * E
5. E→ E / E
6. E → id

Nota: También los podemos ponerlo como una sola producción con alternativas

E → E + E | E - E | E \ E | E * E | (E) | id

1. Definiciones básicas
1.3. Derivaciones

- **Derivación Directa.** Se representa mediante v⇒w y es la aplicación de una regla de producción $x \rightarrow y$ a una cadena *v* para convertirla en otra palabra *w*, de tal forma que si $v = \alpha x \gamma$ entonces $w = \alpha y \gamma$.
  - Se dice que la producción x→y se aplica a la palabra *v* para obtener la palabra *w* o que de *v* se deriva directamente *w*.
  - Ejemplo: E + E ⇒ id + E
- **Derivación.** Se representa mediante v ⇒* w y es la aplicación de una secuencia de producciones a una cadena.
  - Si $\alpha_1, \alpha_2 \ldots \alpha_n$ son palabras y $\alpha_1 \rightarrow \alpha_2$, $\alpha_2 \rightarrow \alpha_3$ … $\alpha_{n-1} \rightarrow \alpha_n$ son reglas de derivación entonces se dice que $\alpha_1 \Rightarrow^* \alpha_n$ o que de α_1 se deriva de α_n .
  - Ejemplo: E + E * E ⇒* id + id * id

1. Definiciones básicas
1.3. Derivaciones

- **Derivación Más a la Izquierda.** Cuando se aplica en cada derivación directa la producción al símbolo no terminal que está más a la izquierda de la cadena.

  Ejemplo: $E \Rightarrow E + E \Rightarrow (E) + E \Rightarrow (E * E) + E \Rightarrow (id *E) + E \Rightarrow$
  $(id * id) + E \Rightarrow (id * id) + id$

- **Derivación Más a la Derecha.** Cuando se aplica al símbolo que está más a la derecha de la cadena.

  Ejemplo: $E \Rightarrow E + E \Rightarrow E + id \Rightarrow (E) + id \Rightarrow (E * E) + id \Rightarrow$
  $(E * id) + id \Rightarrow (id * id) + id$

1. Definiciones básicas
1.4. Sentencias, Lenguaje generado por una Gramática y Gramáticas Equivalentes

- **Sentencia.** *x* es una **sentencia** si $S \Rightarrow^* x$ y $x \in VT^*$.

- **Forma sentencial.** *x* es una **forma sentencial** si $S \Rightarrow^* x$ siendo $x \in (VT \cup VN)^*$.

- **Lenguaje generado por una Gramática.** Es el conjunto de todas las cadenas que se pueden derivar a partir del axioma

$$L(G)=\{w / w \in VT^* \text{ y } S \Rightarrow^* w\}$$

- **Gramáticas Equivalentes.** Dos gramáticas G_1 y G_2 son equivalentes si $L(G_1) = L(G_2)$.

2. Gramáticas y lenguajes libres de contexto

1. Introducción
2. Definición de Gramática Libre de Contexto (GLC) y de Lenguaje Libre de Contexto (LLC)
3. Árboles de Derivación

2. Gramáticas y lenguajes libres de contexto
2.1. Introducción

- **Lenguajes Regulares.** Los LR se describen mediante expresiones regulares y se reconocen mediante AFs.

- **Lema de Pumping. P**ermite demostrar que no todos los lenguajes son regulares. Por ejemplo, la expresión $\{a^n b^n, n > 0\}$ no denota a un lenguaje regular.

2. Gramáticas y lenguajes libres de contexto
2.1. Introducción

- **Lenguajes Libres de Contexto**. Los LLC se describen mediante las Gramáticas Libres de Contexto (GLC).
  - Todos los LR son LLC, pero no todos los LLC son LR.
  - Los LLC (que no sean LR) no pueden denotarse mediante expresiones regulares ni pueden ser reconocidos mediante AF.
  - Los LLC se utilizan para especificar la mayoría de los lenguajes de programación

2. Gramáticas y lenguajes libres de contexto
2.2. Definición de GLC y LLC

- **Gramáticas Libres de Contexto (GLC).** Sus producciones responden al modelo: $A \rightarrow \alpha$, donde $A \in VN$ y $\alpha \in (VN \cup VT)^*$

- **Notación BNF (Backus-Naur-Form).** Esta notación se utiliza para representar la gramática libre de contexto de un lenguaje de programación.
  - Símbolos no terminales entre <>
  - Sustitución de $\rightarrow$ por ::=
  - Ejemplo: <sentenciafor>::=for<condicion><sentencia>

2. Gramáticas y lenguajes libres de contexto
2.2. Definición de GLC y LLC

- **Lenguaje Libre de Contexto (LLC)**. Es el lenguaje generado por una gramática GLC.

$$L(G)=\{w / w \in VT^* \ y \ S \Rightarrow^* w\}$$

es decir $w \in L(G)$ si w está formado cadenas de cero o más terminales y puede ser derivada desde el axioma.

2. Gramáticas y lenguajes libres de contexto
2.2. Definición de GLC y LLC

- **Ejemplos.**

1. G = (VN, VT, E, P) con VT= {id, num,+,*,(,)} VN = {E}
   P: 1. E→E+E 4. E→id
   2. E→E*E 5. E→num
   3. E→(E)
   Derivar la palabra id * (id+num) * id

2. Hallar gramática que genera el lenguaje regular denotado por la expresión regular a*b*

3. Hallar gramática que genera el lenguaje $a^n b^n$, n >= 0

4. Hallar gramática que describe el lenguaje de los palíndromos formados por a's y b's

2. Gramáticas y lenguajes libres de contexto
2.3. Árboles de Derivación

1. Árboles de derivación
   - Definición de árbol de derivación
   - Relación entre derivaciones y árboles
2. Ambigüedad
   - Cadena ambigua
   - Gramática ambigua
   - Lenguaje ambiguo

2. Gramáticas y lenguajes libres de contexto

2.3.1. Definición de árbol de derivación

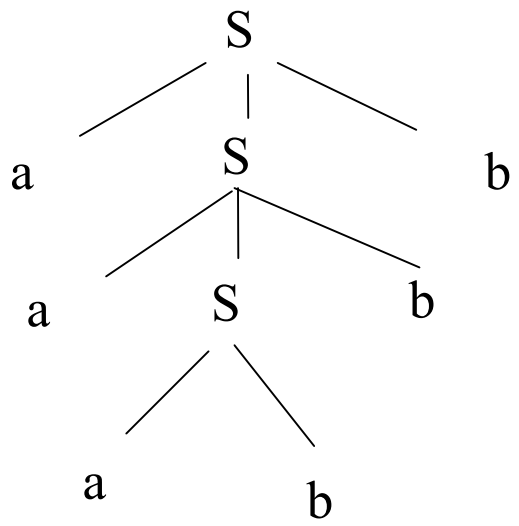
- **Árbol de derivación**
  - Sea *G=(VN,VT,S,P)* una GLC. Un árbol es un **árbol de derivación** para *G* si:
    1. Todo vértice tiene una etiqueta tomada de $VT \cup VN \cup \{\lambda\}$
    2. La etiqueta de la raíz es el símbolo inicial *S*
    3. Los vértices interiores tienen etiquetas de *VN*
    4. Si un nodo n tiene etiqueta *A* y $n_1 n_2 \ldots n_k$ respectivamente son hijos del vértice *n*, ordenados de izquierda a derecha, con etiquetas $x_1, x_2 .. x_k$ respectivamente, entonces:

       $$A \rightarrow x_1 x_2 \ldots x_k$$

       debe ser una producción en *P*
    5. Si el vértice *n* tiene etiqueta λ, entonces *n* es una hoja y es el único hijo de su padre.

2. Gramáticas y lenguajes libres de contexto
2.3.1. Definición de árbol de derivación

- **Árbol de derivación. Ejemplo**
  - Sea *G=(VN,VT,S,P)* una GLC con P: $S \rightarrow ab|aSb$
  - La derivación de la cadena *aaabbb* será: $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$ y el árbol de derivación:



2. Gramáticas y lenguajes libres de contexto

2.3.1. Relación entre derivaciones y árboles

- Si leemos las etiquetas de las hojas de izquierda a derecha tenemos una sentencia. Llamamos a esta cadena la **producción del árbol de derivación**.

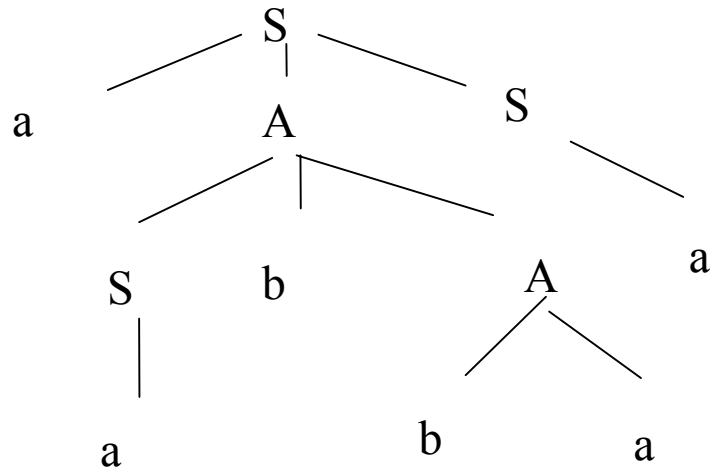
- **Teorema**. Sea *G=(VN,VT,S,P)* una GLC. Entonces $S \Rightarrow^* \alpha$ (de *S* se deriva α) si y sólo si hay un árbol de derivación en la gramática G con la producción α.

- Si w es una cadena de *L(G)* para la gramática libre de contexto *G*, entonces *w* tiene al menos un árbol de derivación. Referido a un árbol de derivación particular, *w* tendrá una única derivación a la izquierda y otra única a la derecha.

2. Gramáticas y lenguajes libres de contexto

2.3.1. Relación entre derivaciones y árboles

- Ejemplo.



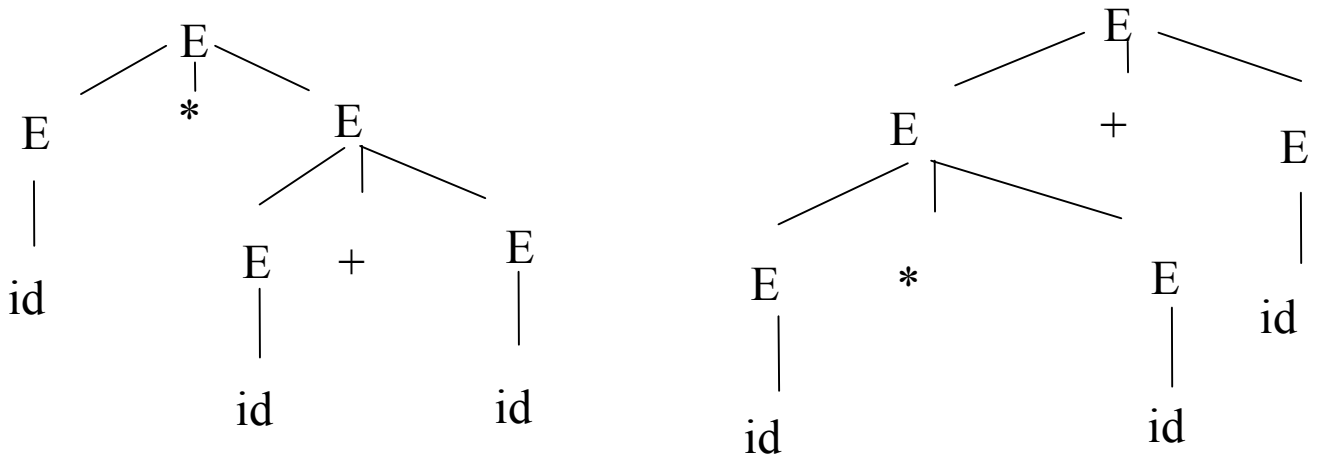
- Derivación a la izquierda:
  - $S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$
- Derivación a la derecha:
  - $S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa$

2. Gramáticas y lenguajes libres de contexto

2.3.2. Ambigüedad

- Es posible que exista más de un árbol de derivación para una única cadena.
- **Ejemplo.** G = (VN, VT, E, P) con VT= {id, num, +, *, (,)} VN = {E}

  P: E→E+E | E*E | (E) | id | num



- La cadena *id*id+id* tiene los dos árboles de derivación anteriores

2. Gramáticas y lenguajes libres de contexto
2.3.2. Ambigüedad

- **Ejemplo.** G = (VN, VT, E, P) con VT= {id , + , - , \ , *, (,) }

  VN = {E, T, F}

  P: E → E + T | E - T | T

  T → T * F | T / F | F

  F → (E) | id

estudiar los posibles árboles de derivación para la cadena

id*(id+id)/id - id

2. Gramáticas y lenguajes libres de contexto

2.3.2. Ambigüedad

- Una gramática libre de contexto *G*, se dice que es **ambigua**, si existe alguna palabra perteneciente al lenguaje que genera que tenga más de un árbol de derivación.

- Un lenguaje es **intrínsecamente ambiguo** cuando no existe ninguna gramática libre de contexto que lo genere que no sea ambigua.

3. Forma Normal de Chomsky

1. Eliminación de símbolos inútiles
2. Eliminación de λ-producciones
3. Eliminación de producciones unidad
4. Forma Normal de Chomsky

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Limpieza de GLC.** Restringir el formato de las producciones de la gramática sin reducir su poder generativo de palabras de un LLC.
  - Cada símbolo no terminal y cada símbolo terminal aparece en la derivación de alguna palabra del LLC generado por la GLC.
  - Se pretende que todos los símbolos sean útiles.

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Símbolos Muertos.** Símbolos **no terminales** de los que no se deriva ninguna cadena de símbolos terminales.

- **Símbolos Inaccesibles.** Símbolos **terminales** o **no terminales** que no puedes ser generados a partir del axioma

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Lema para eliminar símbolos muertos**

  - Dada una GLC $G = (VN, VT, S, P)$ con L(G) ≠ Ø, se puede encontrar otra GLC $G' = (VN', VT, S, P')$ **equivalente** a G tal que $\forall A \in VN'$ existe al menos una palabra $w \in VT^*$ tal que $A \Rightarrow^* w$.

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

Algoritmo para eliminar Símbolos Muertos.

- Paso 1. Cálculo de *VN'*
  - Si una producción es de la forma $A \rightarrow w$ con $w \in VT^*$ entonces se añade el símbolo *A* en *VN'*
  - Si una producción es de la forma $A \rightarrow x_1 x_2 \ldots x_n$, siendo cada x_i un símbolo terminal o un no-terminal que ya han sido incluido en *VN'*, entonces se añade el símbolo *A* en *VN'*

- Paso 2. Cálculo de *P'*
  - *P'* será el conjunto de todas las producciones de P que contengan sólo símbolos pertenecientes a *(VN' U VT)*

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Eliminar símbolos muertos.**
  - Ejemplo.

    G = ({S,A, B, C, D, E}, {a, b, c}, S, P) y

    P: S →aBB |CED

    A →BS

    B →A |b |λ

    C →bEa |BB

    D →AB |λ | cc

    E →EB |DE

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Lema para eliminar símbolos inaccesibles**
  - Dada una GLC $G = (VN, VT, S, P)$, se puede encontrar otra GLC $G' = (VN', VT', S, P')$ **equivalente** a G tal que $\forall X \in (VN' \cup VT') \exists \alpha, \beta \in (VN' \cup VT')^*$ tal que $S \Rightarrow^* \alpha X \beta$.

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Algoritmo para eliminar Símbolos Inaccesibles.**
  - Paso 1. Cálculo de *VN'* y *VT'*
    - Se coloca el símbolo inicial S en *VN'*
    - Si $A \in VN'$ y la producción $A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n \in P$, se añade a *VN'* todos los símbolos no terminales que aparecen en cada α_i *y a VT'* todos los símbolos terminales que aparecen en cada α_i
  - Paso 2. Cálculo de P'
    - *P'* será el conjunto de todas las producciones de P que contengan sólo símbolos pertenecientes a *(VN' U VT')*

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Eliminar símbolos inútiles**
  - **Es importante primero eliminar los símbolos muertos y luego los inaccesibles**
  - **Teorema.** Todo LLC no vacío es generado por alguna GLC sin símbolo inútiles.

3. Forma Normal de Chomsky

3.1. Eliminación de símbolos inútiles

- **Eliminar símbolos inaccesibles.**
  - Ejemplo.

    G = ({S,A, B, C, D, E}, {a, b, c}, S, P) y

    P: S →aBB

    A →BS | CC

    B →A |b

    C →bEa |BB

    D →AB |λ

    E →EB |DE

3. Forma Normal de Chomsky

3.2. Eliminación de λ-producciones

- **Eliminar λ-producciones o producciones vacías**
  - **Si** $\lambda \in L(G)$ entonces **no** se pueden eliminar **todas** las λ-producciones de *G*. Por el contrario, si $\lambda \notin L(G)$ entonces **sí** se pueden eliminar **todas** las λ-producciones de *G*.
  - **Símbolos anulables.** Un símbolo no terminal *A* es **anulable** $\Leftrightarrow A \Rightarrow^* \lambda$
    - Si $A \rightarrow \lambda$ es una producción, entonces *A* es anulable.
    - Si $A \rightarrow \alpha$ es una producción y todos los símbolos de α son anulables, entonces *A* es anulable.

3. Forma Normal de Chomsky

3.2. Eliminación de λ-producciones

- **Eliminar λ-producciones o producciones vacías**
  - **Ejemplo.** Dadas las producciones

    $S \rightarrow AbC \qquad A \rightarrow a \mid \lambda \qquad C \rightarrow c \mid \lambda,$

    entonces *A* y *C* son anulables.

  - **Teorema.** Si *L(G)* es generada por una GLC *G*, entonces *L(G)-{λ}* es generada por una GLC *G'* sin símbolos inútiles ni λ-producciones.

3. Forma Normal de Chomsky

3.2. Eliminación de λ-producciones

- **Eliminar λ-producciones: Algoritmo**
  1. Determinar los símbolos anulables.
  2. Incluir en *P'* todas las producciones de *P* menos las λ-producciones
  3. Si $A \rightarrow \alpha B \beta$ está en *P' y B* es anulable, entonces se añade a *P'* la regla $A \rightarrow \alpha\beta$ *(* excepto si $A \rightarrow \alpha\beta$ es una λ-producción).
  4. Repetir 3. hasta que no se puedan añadir más producciones

3. Forma Normal de Chomsky

3.2. Eliminación de λ-producciones

- **Eliminar λ-producciones: Ejemplos**
  - Encontrar la gramática equivalente sin λ-producciones.

    $S \rightarrow 0A$

    $A \rightarrow 10A \mid \lambda$

  - Encontrar la gramática equivalente sin λ-producciones.

    $S \rightarrow 0A$

    $A \rightarrow 10AB \mid \lambda$

    $B \rightarrow BAB \mid 1$

3. Forma Normal de Chomsky

3.3. Eliminación de producciones unidad

Producciones Unidad:

Las **producciones unidad** o **reglas de redenominación** son producciones de la forma

$$A \rightarrow B$$

es decir, la parte derecha es un único símbolo no terminal.

3. Forma Normal de Chomsky

3.3. Eliminación de producciones unidad

Algoritmo para eliminar Producciones Unidad.

1. Eliminar λ-producciones

2. Para toda producción unidad $A \rightarrow B$, *con* $B \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$ sustituir $A \rightarrow B$ *por* $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$

3. Repetir 2. mientras queden producciones unidad

3. Forma Normal de Chomsky

3.3. Eliminación de producciones unidad

- **Eliminar producciones unidad: Ejemplos**
  - Encontrar la gramática equivalente sin producciones unidad.

$S \rightarrow Cbh \mid D$

$A \rightarrow aaC$

$B \rightarrow Sf \mid ggg$

$C \rightarrow cA \mid d \mid C$

$D \rightarrow E \mid SABC$

$E \rightarrow be$

3. Forma Normal de Chomsky

3.3. Eliminación de producciones unidad

Teorema.

Todo LLC sin λ está definido por una gramática sin

- símbolos inútiles,
- ni λ-producciones,
- ni producciones unidad.

3. Forma Normal de Chomsky

3.3. Forma Normal de Chomsky

- Cualquier LLC sin λ es generado por una gramática en la que todas las producciones son de la forma $A \rightarrow a$ o $A \rightarrow BC$ donde $A, B, C \in VN$ y $a \in VT$.

- Una gramática que tiene todas las producciones en la forma indicada anteriormente se dice que están en **Forma Normal de Chomsky (FNC)**.

3. Forma Normal de Chomsky

3.3. Forma Normal de Chomsky

Suponemos *G* una GLC de forma que *L(G)* no contiene λ.

PASO PREVIO: Eliminar λ-producciones, producciones unidad y símbolos inútiles.

- En este momento, cualquier producción de la forma $A \rightarrow x$ ya está en FNC, porque *x* tiene que ser un terminal (no hay prod. unidad)

3.3. Forma Normal de Chomsky

- Nos ocuparemos de producciones de la forma

  $A \rightarrow x_1 x_2 \ldots x_m$

  con $m \geq 2$
- Si x_i es un símbolo terminal, a, introducimos un nuevo símbolo no terminal C_a y una producción

  $$C_a \rightarrow a$$

  y reemplazamos x_i por C_a.
- Repetimos el proceso para cada símbolo terminal y para cada producción.

3.3. Forma Normal de Chomsky

- Tendremos producciones de la forma

  $$A \rightarrow B_1 B_2 \ldots B_m$$

  con $m \geq 2$, siendo cada $B_i \in VN$
- Si $m = 2$ ya está en FNC
- Si m>2 creamos símbolos no terminales D_1, D_2, ..., D_{m-2} y reemplazamos la producción $A \rightarrow B_1 B_2 \ldots B_m$ por el conjunto de producciones:

  $\{ A \rightarrow B_1 D_1,\ D_1 \rightarrow B_2 D_2, \ldots\ D_{m-3} \rightarrow B_{m-2} D_{m-2},\ D_{m-2} \rightarrow B_{m-1} B_m \}$

3.3. Forma Normal de Chomsky

- Buscar una gramática en FNC equivalente a $G = (\{S, A, B\}, \{a, b\}, P, S)$ siendo P:

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

- Buscar una gramática en FNC que genere $a^n b^n$, con $n > 0$.

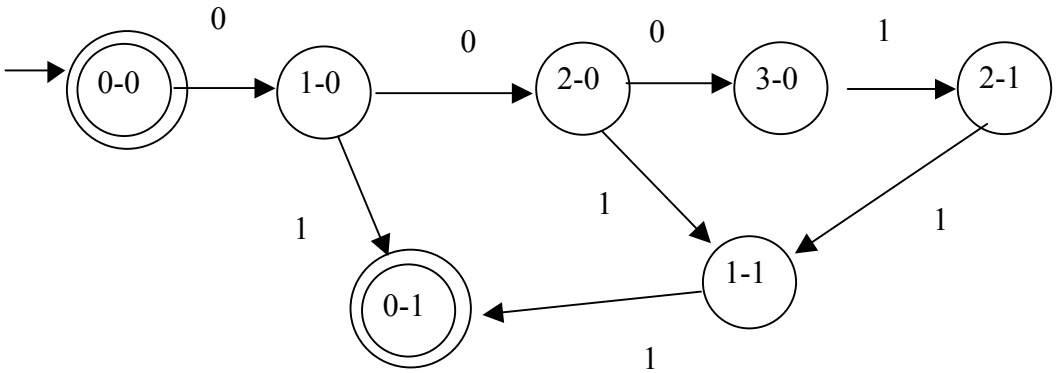
4. Autómatas de Pila

1. Introducción
2. Definición Formal de un AP
3. AP Deterministas y No Deterministas
4. Construcción de un AP a partir de una gramática

4. Autómatas de Pila

4.1 Introducción

¿Podríamos encontrar un AF que reconociera $L=\{0^n1^n \mid n \geq 0\}$?

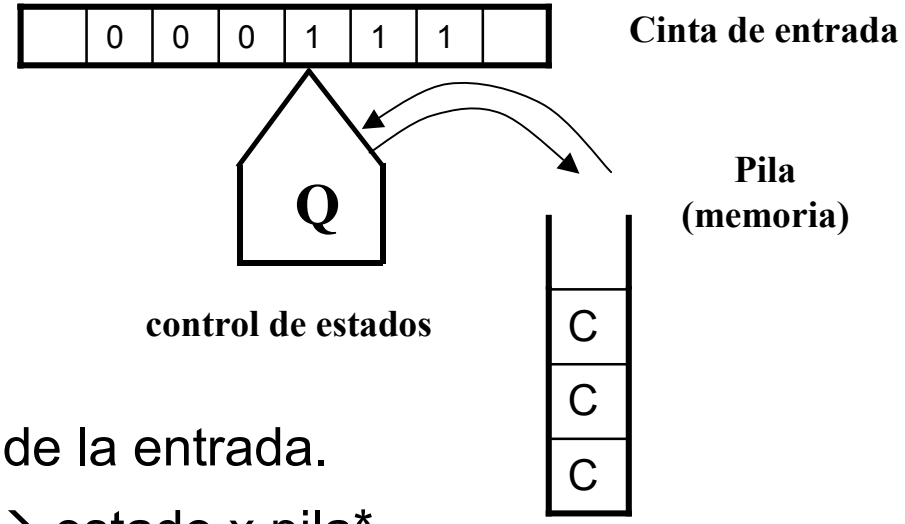


- La longitud de la cadena puede ser infinita y no podemos construir infinitos estados
- El lema de Pumping nos dice que no
- Idea: No tenemos forma de memorizar el número de 0's que han llegado para compararlo con el número de 1's

4. Autómatas de Pila

4.1 Introducción

- Un Autómata de Pila es aquella máquina que reconoce lenguajes generados por gramáticas libres de contexto (GLC).



Aceptación de las cadenas

- Por vaciado de la pila
- Por estados finales

Funcionamiento dependiente de la entrada.

f: estado x pila x cinta → estado x pila*

Funcionamiento independiente de la entrada.

f: estado x pila x {λ} → estado x pila*

(*) De la pila se lee un solo símbolo pero puede escribir una cadena

4. Autómatas de Pila

4.2. Definición Formal de un AP

Un AP es una séptupla $(\Sigma, \Gamma, Q, Z, q_0, f, F)$ donde:

Σ es el alfabeto de entrada

Γ es el alfabeto de la pila

Q es el conjunto finito de estados

Z es el símbolo inicial de la pila

q_0 es el estado inicial

f es una aplicación $f : Q \times \Sigma \cup \{\lambda\} \times \Gamma \rightarrow P(Q \times \Gamma^*)$

F es un subconjunto de Q que contiene los estados finales.

4. Autómatas de Pila

4.2. Definición Formal de un AP

Descripción Instantánea (configuración).

$$(q, w, \alpha)$$

- q representa al estado actual del autómata
- w es la cadena de entrada que falta por analizar, $w \in \Sigma^*$

  si $w = \lambda$, toda la cadena de entrada ha sido leída.
- α es el contenido de la pila en el instante actual

  si $\alpha = \lambda$ la pila está vacía, $\alpha \in \Gamma^*$

4. Autómatas de Pila

4.2. Definición Formal de un AP

Movimiento.

Transición entre dos descripciones instantáneas

$$(q, aw, Z\alpha) \vdash (q', w, \gamma\alpha)$$

donde la función **f** para esta entrada toma el valor:

$$(q', \gamma) \in f(q, a, Z)$$

NOTA: $\vdash^*$ equivale a cero o más movimientos

4. Autómatas de Pila

4.2. Definición Formal de un AP

Configuración inicial:

(q_0, w, Z) , $w \in \Sigma^*$

Configuración final:

AP por estados finales:

(p, λ, X) con $p \in F$ y $X \in \Gamma^*$

AP por vaciado de pila:

(p, λ, λ), con $p \in Q$, $\alpha \in \Gamma^*$

4. Autómatas de Pila

4.2. Definición Formal de un AP

Definición. **Lenguaje aceptado por un AP.**

- **Lenguaje aceptado por estados finales** por el AP = $(\Sigma, \Gamma, Q, Z, q_0, f, F)$:

$$\{w \in \Sigma^* \mid (q_0, w, Z) \vdash^* (p, \lambda, \mathbf{X}), \textbf{ con } \mathbf{p \in F \text{ y } X \in \Gamma^*}\}$$

- **Lenguaje aceptado por vaciado de pila** por el AP = $(\Sigma, \Gamma, \mathbf{Q, Z, q_0, f, \varnothing})$ **al conjunto:**

$$\mathbf{\{w \in \Sigma^* \mid (q_0, w, Z) \vdash^* (p, \lambda, \lambda), \textbf{ con } p \in Q\}}$$

4. Autómatas de Pila

4.2. Definición Formal de un AP

EJEMPLO: AP que reconozca el LLC $L=\{0^n1^n \mid n \geq 1\}$

1) Alfabeto de entrada : $\Sigma = \{0, 1\}$
2) Alfabeto de la pila Γ
   - i) Llegada de un 0 $\Rightarrow$ Metemos una C en la pila
   - ii) Llegada de un 1 $\Rightarrow$ Quito una C de la pila
   - iii) El estado inicial de la pila $\Rightarrow$ Z

   Luego, $\Gamma = \{C, Z\}$
3) Conjunto de estados Q

   Estado p: leyendo ceros ('apilando')

   Estado q: comprobando unos ('desapilando')

   Luego, $Q = \{p, q\}$

4. Autómatas de Pila

4.2. Definición Formal de un AP

4) Empezaremos a funcionar en el estado p: $q_0 = p$
5) El AP va a funcionar por vaciado de pila: $F = \varnothing$
6) El símbolo inicial de la pila: Z
7) Y por último la función se definirá como (*)

$f(p, 0, Z) = \{(p, CZ)\}$
$f(p, 0, C) = \{(p, CC)\}$
$f(p, 1, C) = \{(q, \lambda)\}$

$f(q, 1, C) = \{(q, \lambda)\}$
$f(q, \lambda, Z) = \{(q, \lambda)\}$

(*) con f definimos aquellos movimientos que hacen que las cadenas del lenguaje sean aceptadas, no definimos aquellas transiciones que no nos conduzcan a una aceptación

4. Autómatas de Pila

4.3. AP Deterministas y No Deterministas

Se dice que un AP es determinista cuando cumple que:

1. $\forall q \in Q$ y $\forall X \in \Gamma$, **si** $f(q, \lambda, X)) \neq \varnothing$ **entonces** $f(q, a, X) = \varnothing \quad \forall a \in \Sigma$

   Es decir, no se podrá optar por un funcionamiento independiente de la entrada u otro dependiente de la entrada. Ejemplo:

   $f(q, b, A) = \{(q, AA)\}$
   $f(q, \lambda, A) = \{(q, S)\}$

2. $\forall q \in Q$, $\forall A \in \Gamma$ y $\forall a \in (\Sigma \cup \{\lambda\})$ debe cumplir que Cardinal $(f(q, a, A)) \leq 1$

   No habrá para ningún estado más de un posible movimiento dado un determinado símbolo de entrada y uno en la cima de la pila. Ejemplo:

   $f(q, a, S) = \{(q, A), (q, AS)\}$

4.3. AP Deterministas y No Deterministas

AP que reconoce el lenguaje $\{z2z^{-1} \mid z \in (0|1)^*\}$

El AP será $(\{0,1,2\}, \{Z,C,U\}, \{p,q\}, Z, p, f, \varnothing\}$ y f será:

$f(p, 0, Z) = \{(p, CZ)\}$ $f(p, 0, C) = \{(p, CC)\}$
$f(p, 0, U) = \{(p, CU)\}$ $f(p, 1, Z) = \{(p, UZ)\}$
$f(p, 1, C) = \{(p, UC)\}$ $f(p, 1, U) = \{(p, UU)\}$
$f(p, 2, Z) = \{(q, Z)\}$ $f(p, 2, C) = \{(q, C)\}$
$f(p, 2, U) = \{(q, U)\}$ $f(q, 0, C) = \{(q, \lambda)\}$
$f(q, 1, U) = \{(q, \lambda)\}$ $f(q, \lambda, Z) = \{(q, \lambda)\}$

Aceptación de la cadena w = 0112110

$\underline{(p, 0112110, Z)} \vdash (p, 112110, CZ) \vdash (p, 12110, UCZ)$
$\vdash (p, 2110, UUCZ) \vdash (q, 110, UUCZ) \vdash (q, 10, UCZ)$
$\vdash (q, 0, CZ) \vdash (q, \lambda, Z) \vdash \underline{(q, \lambda, \lambda)}$

4. Autómatas de Pila

4.3. AP Deterministas y No Deterministas

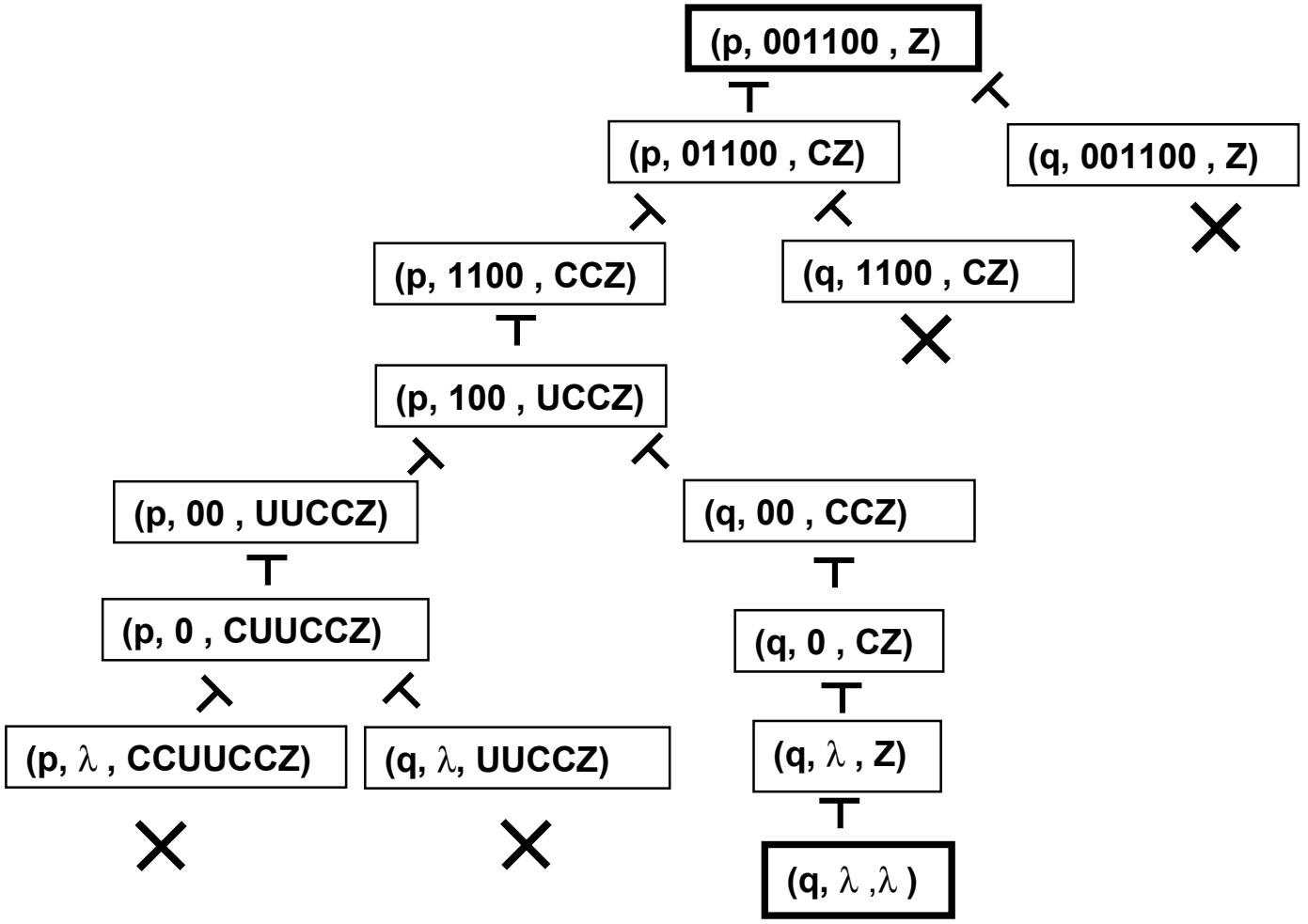
AP que reconoce el lenguaje $\{zz^{-1} \mid z \in (0|1)^*\}$
El AP será $(\{0,1\}, \{Z,C,U\}, \{p,q\}, Z, p, f, \varnothing\}$ y f será:

$f(p, 0, Z) = \{(p, CZ)\}$
$f(p, 0, U) = \{(p, CU)\}$
$f(p, 1, C) = \{(p, UC)\}$
$f(q, 0, C) = \{(q, \lambda)\}$
$f(q, \lambda, Z) = \{(q, \lambda)\}$

$f(p, 0, C) = \{(p, CC), (q, \lambda)\}$
$f(p, 1, Z) = \{(p, UZ)\}$
$f(p, 1, U) = \{(p, UU), (q, \lambda)\}$
$f(q, 1, U) = \{(q, \lambda)\}$
$f(p, \lambda Z) = \{(q, \lambda)\}$

4. Autómatas de Pila

4.3. AP Deterministas y No Deterministas



4. Autómatas de Pila

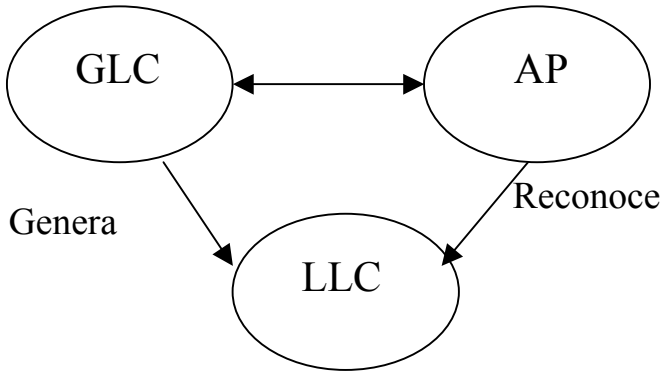
4.3. AP Deterministas y No Deterministas

Teorema. El conjunto de lenguajes aceptados por estado final es igual al conjunto de lenguajes aceptados por vaciado de pila.

Teorema. A toda gramática libre de contexto, le corresponde un AP que reconoce por vaciado de pila el lenguaje generado por ella y viceversa.

Corolario. El conjunto de lenguajes reconocidos por los AP son los Lenguajes Libres de Contexto y todo LLC tiene un AP que lo reconoce.

También se puede deducir que existe una correspondencia entre los LLC, las Gramáticas Libres de Contexto y los AP.



4. Autómatas de Pila

4.4. Construcción de un AP a partir de una gramática

Sea la GLC G = (VN, VT, S, P) cualquiera y construimos el AP= (Σ, Γ, Q, Z, q_0, f, $\varnothing$) donde:

Σ= VT Γ= VT U VN

Q= {q} Z= S

q_0=q

definiendo f como:

$(q, \lambda) \in f(q, a, a) \quad \forall a \in VT$

$(q, \alpha) \in f(q, \lambda, A)$ sii $A \rightarrow \alpha \in P$, donde $A \in VN$ y $\alpha \in (VT \cup VN)^*$

4. Autómatas de Pila

4.4. Construcción de un AP a partir de una gramática

Ejemplo. Sea la gramática G = (VN, VT, S, P) con

VT = { id, +, *, [,] } P ={ $E \rightarrow E+T \mid T$

VN = {E, T, F} $T \rightarrow T*F \mid F$

S = E $F \rightarrow [E] \mid id$ }

Construimos el AP = (Σ, Γ, Q, Z, q_0, f, $\varnothing$) donde:

Σ= {id, +, *, [,]} Γ= VN U VT= {E, T, F,id, +, *, [,]}

Q= {q} Z= E q_0=q

f: f(q, id, id) = {(q, λ)} f(q, λ, E) = {(q, E+T), (q, T)}

f(q, +, +) = {(q, λ)} f(q, λ, T) = {(q, T*F), (q, F)}

f(q, *, *) = {(q, λ)} f(q, λ, F) = {(q, [E]), (q, id)}

f(q, [, [ ) = {(q, λ)}

f(q,],]) = {(q, λ)}

4. Autómatas de Pila

4.4. Construcción de un AP a partir de una gramática

f: f(q, id, id) = {(q, λ)} | f(q, +, +) = {(q, λ)} | f(q, λ, T) = {(q, T*F), (q, F)}
f(q, [, [ ) = {(q, λ)} | f(q,],]) = {(q, λ)} | f(q, λ, E) = {(q, E+T), (q, T)}
f(q, *, *) = {(q, λ)} | f(q, λ, F) = {(q, [E]), (q, id)}

(q, id*[id+id], E)	⊢	(E→T)
(q, id*[id+id], T)	⊢	(T→T*F)
(q, id*[id+id], T*F)	⊢	(T→F)
(q, id*[id+id], F*F)	⊢	(F→ id)
(q, id*[id+id], id*F)	⊢	**id**
(q, *[id+id], *F)	⊢	*
(q, [id+id], F)	⊢	(F→[E])
(q, [id+id], [E])	⊢	**[**

4. Autómatas de Pila

4.4. Construcción de un AP a partir de una gramática

f: $f(q, id, id) = \{(q, \lambda)\}$ | $f(q, +, +) = \{(q, \lambda)\}$ | $f(q, \lambda, T) = \{(q, T*F), (q, F)\}$

$f(q, [, [\,) = \{(q, \lambda)\}$ | $f(q,],]) = \{(q, \lambda)\}$ | $f(q, \lambda, E) = \{(q, E+T), (q, T)\}$

$f(q, *, *) = \{(q, \lambda)\}$ | | $f(q, \lambda, F) = \{(q, [\ E\ ]), (q, id)\}$

(q, id+id],	E])	⊢ (E→ E + T)
(q, id+id],	E + T])	⊢ (E→ T)
(q, id+id],	T + T])	⊢ (T→ F)
(q, id+id],	F + T])	⊢ (F→ id)
(q, id+id],	id + T])	⊢ **id**
(q, +id],	+ T])	⊢ **+**
(q, id],	T])	⊢ (T→ F)
(q, id],	F])	⊢ (F→ id)
(q, id],	id])	⊢ **id**
(q,],	])	⊢ **]**

(q, λ, λ)

5. Propiedades de los Lenguajes Libres de Contexto

- Propiedades de Clausura de los LLC
- Algoritmos de decisión para LLC

5. Propiedades de los LLC

5.1. Propiedades de clausura

Teorema.

Los LLC son cerrados bajo la unión, la concatenación y el cierre de Kleene.

Demostración.

Supongamos L_1 y L_2 LLC generados por GLC.

$G1 = (VN_1, VT_1, S_1, P_1)$ $G_2 = (VN_2, VT_2, S_2, P_2)$

Como podemos renombrar variables sin cambiar el lenguaje generado, asumiremos que VN_1 y VN_2 son disjuntos.

Asumiremos también que S_3, S_4 y S_5 no están ni en VN_1 ni en VN_2.

5. Propiedades de los LLC

5.1. Propiedades de clausura

Unión: $L_1 \cup L_2$

construimos la gramática $G_3 = (VN_1 \cup VN_2 \cup \{ S_3\}, VT_1 \cup VT_2, S_3, P_3)$, donde P_3 tiene $P_1 \cup P_2$ más las producciones:

$$S_3 \rightarrow S_1 \mid S_2$$

Demostración:

1) Si $w \in L_1$ ó $w \in L_2 \Rightarrow w \in L_3$

2) Si $w \in L_3 \Rightarrow w \in L_1$ ó $w \in L_2$

1) Si $w \in L_1$ (idem si $w \in L_2$) entonces $w \in L_3$, ya que la derivación

$$S_3 \Rightarrow S_1 \Rightarrow^* w$$

es una derivación en G_3 pues cada producción de G_1 también lo es de G_3. (Idem para $w \in L_2$.)

$$\text{Así, } L_1 \cup L_2 \subseteq L_3$$

5.1. Propiedades de clausura

Unión: $L_1 \cup L_2$ (cont)

2) Inversamente, supongamos $w \in L(G_3)$. Entonces, la derivación

$S_3 \Rightarrow^* w$ comienza con

$S_3 \Rightarrow S_1 \Rightarrow^* w$ ó

$S_3 \Rightarrow S_2 \Rightarrow^* w$

Si $S_1 \Rightarrow^* w$ entonces $w \in L_1$; ó si $S_2 \Rightarrow^* w$, entonces $w \in L_2$.

Por eso, $L(G_3) \subseteq L_1 \cup L_2$.

Y por tanto de 1) y 2) : $\mathbf{L_3 = L_1 \cup L_2}$

5. Propiedades de los LLC

5.1. Propiedades de clausura

Concatenación: $L_1 \cdot L_2$

Construimos la gramática $G_4 = (VN_1 \cup VN_2 \cup \{ S_4\}, VT_1 \cup VT_2, S_4, P_4)$, donde P_4 tiene $P_1 \cup P_2$ más la producción:

$$S_4 \rightarrow S_1 \, S_2$$

La demostración de que $L(G_4) = L(G_1)L(G_2)$ es similar a la demostración para la unión.

5. Propiedades de los LLC

5.1. Propiedades de clausura

Cierre estrella: L_1^*

Construimos la gramática $G_5 = (VN_1 \cup \{ S_5\}, VT_1 , S_5, P_5)$, donde P_5 tiene P_1 más las producciones:

$$S_5 \rightarrow S_1 S_5 \mid \lambda$$

La demostración de que $L(G_5) = L(G_1)^*$ es idéntica a las anteriores.

5. Propiedades de los LLC

5.2. Algoritmos de decisión

Hay algunas otras cuestiones que no tienen una respuesta algorítmica, como son:

- ¿Son dos GLC equivalentes?
- ¿El complementario de un LLC es también un LLC?
- ¿Es ambigua una GLC?

Pero para resolver cuestiones contar con algorítmos para ello:

- ¿Es un LLC vacío?
- ¿Es un LLC finito o infinito?
- ¿Una palabra dada pertenece al lenguaje?

5.2. Algoritmos de decisión

¿Es un LLC vacío?

- Un LLC será vacío si a partir del axioma no generar ninguna cadenas de símbolos terminales (incluida λ).

- Es decir, basta determinar si el axioma es un símbolo muerto

5.2. Algoritmos de decisión

¿Es un infinito?

Para ver si L(G) es finito, buscamos una GLC G' en FNC y sin símbolos inútiles, que genere L(G) - {λ}. L(G') será finito si L(G) lo es.

Construimos un grafo dirigido con un vértice para cada no-terminal.

Si exite en P alguna producción de la forma $A \rightarrow BC$ o $A \rightarrow CB$ para cualquier C, entoces exixtirá un arco del nodo A al nodo B.

Entonces, el lenguaje creado es finito sii este grafo **no tiene ciclos**.

5.2. Algoritmos de decisión

¿Es un infinito?

Ejemplo:

$$S \rightarrow AB$$

$$A \rightarrow BC \mid a$$

$$B \rightarrow CC \mid b$$

$$C \rightarrow a$$

5. Propiedades de los LLC

5.2. Algoritmos de decisión:CYK

¿ Pertenece x a L(G) ?

- Dada un GLC G = (VN, VT, S, P) y una cadena $x \in VT^*$, existe un algorítmo denominado CYK (Cocke-Younger-Kasami) para determinar si $x \in L(G)$.

- Partimos de una GLC en Forma Normal de Comsky

- Se trata de reconstruir el árbol de derivación para x de abajo arriba.

5. Propiedades de los LLC

5.2. Algoritmos de decisión:CYK

¿ Pertenece x a L(G) ?

- Iremos obteniendo los VN que generan las distintas subcadenas de x,

- Estos conjuntos de VN se denominarán V_{ij} y contienen símbolos VN que derivan la **subcadena de x que comienza en la posición i y con longitud j**.

- El objetivo es obtener los VN del conjunto V_{1n} (toda la cadena x), si uno de ellos es el axioma, entonces **x**$\in$ **L(G).**

5. Propiedades de los LLC

5.2. Algoritmos de decisión:CYK

Formalmente, podemos expresar el algoritmo anterior como sigue:

begin

for i:=1 to n do

V_{i1} := { A | A $\rightarrow$ a es una producción de P y el i-ésimo símbolo de x es a }

for j := 2 to n do /* longitudes sucesivas*/

for i:=1 to n-j+1 do /* principio de la palabra*/

begin

V_{ij} := $\varnothing$;

for k:=1 to j-1 do

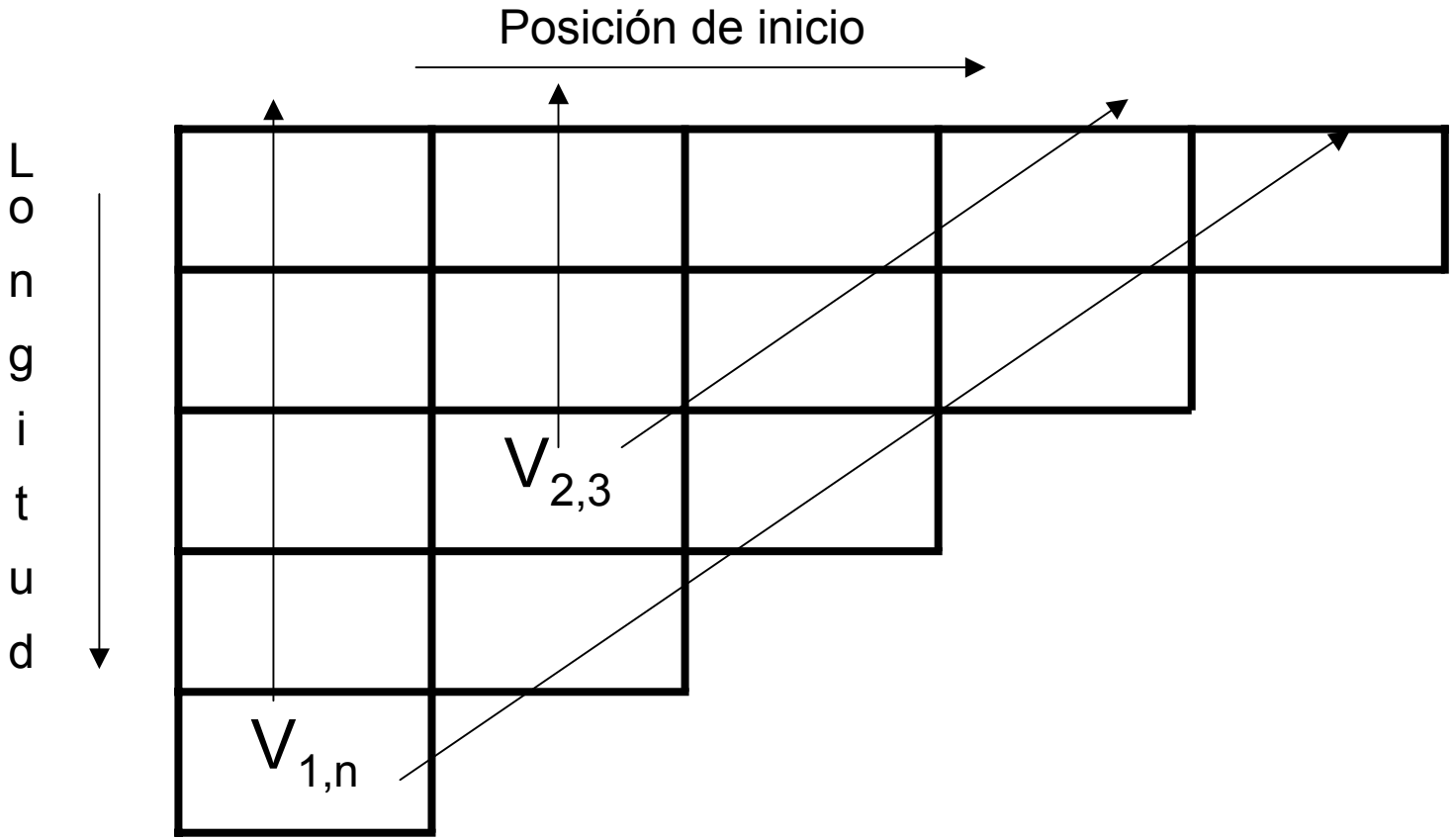
V_{ij} := $V_{ij} \cup$ { A | A $\rightarrow$ BC es una producción de P, B está en V_{ik} y C está en V_{i+kj-k} }

end

end

5.2. Algoritmos de decisión:CYK

El algoritmo va a ir rellenado la siguiente matriz



5.2. Algoritmos de decisión:CYK

Ejemplo:

$S \rightarrow AB \mid BC$ $\quad$ $A \rightarrow BA \mid a$

$B \rightarrow CC \mid b$ $\quad$ $C \rightarrow AB \mid a$

	b	a	a	b	a ← La cadena
1	B	A,C	A,C	B	A,C
2	A,S	B	S,C	S,A	
3	∅	B	B		
4	∅	A,C			
5	A,S,C				