

Lenguajes Formales

impartido por: Fernando Torre Cervigón en la E.U.I.T.I.O. [<http://www.euitio.uniovi.es>]

redactado por: Miguel Herrero Obeso [<http://petra.euitio.uniovi.es/~i2140099/>]

actualizado el: 11/06/2005

Nota: Estos apuntes pretenden ser un reflejo de lo explicado en clase y están pensados para compaginarlos con las transparencias/apuntes de la asignatura. No me responsabilizo de los suspensos que originen estos apuntes. Si consideras que alguna parte debería ser corregida, házmelo saber enviándome un correo-e a través de mi página web.

Índice

Índice	1
Conceptos Básicos	2
Monoide libre	3
Operaciones sobre palabras	3
Operaciones con lenguajes	5
Contexto válido	7
Relación	8
Derivaciones	9
Relación de Thue	10
Gramáticas	11
Formalizar una gramática	12
Gramática Formal	13
Notación de Backus-Naur	14
Lenguaje asociado a una gramática	14
Recursividad	15
Tipos de gramática	15
Gramáticas de tipo 0	15
Gramáticas de tipo 1	16
Gramáticas de tipo 2	16
Gramáticas de tipo 3	17
Árbol de derivación	19
Gramáticas independientes de contexto (GIC)	24
Símbolos inútiles	25
Reglas de producción λ o producciones lambda	25
Reglas de producción unitarias (o producciones unitarias)	27
Forma normal de Chomsky (CNF)	27
Forma normal de Greibach (GNF)	28
Máquina de Turing	30
Acciones posibles	31
Máquinas de Turing restringidas	34
Máquina de Turing universal	34
Composición de máquinas de Turing	34
Tipos de máquinas de Turing	36
Descripción instantánea	36
Lenguajes aceptados por una máquina de Turing	38
Máquinas de Turing no deterministas	39
Máquina de Turing de k cintas	40
Complejidad	42
Complejidad de los procesos realizados por las máquinas de Turing	42
Probabilidad	43
Tasas de crecimiento	44
Complejidad temporal de los problemas de reconocimiento de lenguajes	45
Decisión de un lenguaje	46
Lenguajes decidibles	46

Conceptos Básicos

Alfabeto: conjunto de elementos no vacío (como mínimo tiene un elemento), con todos los elementos distintos entre sí y finito (no puede tener infinitos elementos).

Se representa entre llaves y separado por comas.

{ ..., ..., ..., ... }

Elementos del alfabeto: letras, números, letras y números, otros símbolos.

Vamos a designar el alfabeto con letras griegas mayúsculas (A,B,Γ,Δ,Σ)

Ejemplo:

A = {a, b, c, ..., z}

Γ = {0, 1, 2, ..., 9}

Σ = {#, |, //, =, ...}

Para expresar la pertenencia de un elemento a un conjunto, lo indicamos así:

a ∈ A

2 ∈ Γ

| ∈ Σ

Nota: el conjunto vacío (Φ) NO puede representar un alfabeto.

Palabra ó cadena: secuencia finita de símbolos de un alfabeto.

Ejemplo:

Σ₁ = {A, B, C, ..., Z}

Palabras: CASA, B, ZZX

Alfabeto Binario Σ₂ = {0, 1}

Palabras: 0, 1, 00, 11, 101101

Σ₃ = {/, |, \, -}

Palabras: /||\

Vamos a representar las palabras con letras latinas minúsculas.

Ejemplo:

x = CASA

y = 3

z = /||\

Longitud de una palabra: número de símbolos que componen dicha palabra. Se representa situando la letra que define la palabra entre dos líneas verticales.

Ejemplo:

|x| = 4

Palabra vacía: cadena formada por ningún elemento. Se representa por λ.

Nota: $|\lambda| = 0$

Lenguaje universal: es el conjunto de todas las palabras que se pueden construir con los símbolos de un alfabeto. Es un conjunto infinito.

Ejemplo:
 $\Sigma = \{A\}$
 $W(\Sigma) = \{A, AA, AAA, AAAA, \dots\}$

Nota: la palabra vacía pertenece a TODOS los lenguajes universales.

Ejemplo:
 $W(\Sigma) = \{a, b, zzx, \dots\}$
 $\Sigma = \{A, B, \dots, Z, a, b, \dots, z\}$

Monoide libre

Dado un alfabeto Σ , todos los elementos de Σ son palabras de longitud 1 del lenguaje universal $W(\Sigma)$. Se observa que el resto de palabras, excepto la palabra vacía, se pueden construir concatenando dichas palabras. Entonces podemos definir el alfabeto:

$\Sigma =$ conjunto generacional de $W(\Sigma) - \{\lambda\}$
 $W(\Sigma) - \{\lambda\}$ se le llama semigrupo libre generado por Σ

Por lo tanto:

Semigrupo libre + $\{\lambda\} = W(\Sigma)$

A este $W(\Sigma)$ le llamamos **monoide libre** generado por Σ .

Operaciones sobre palabras

Concatenación

Dado $\Sigma = \{A_1, A_2, A_3, \dots\}$ tenemos x e y tal que:

$x \in W(\Sigma)$
 $y \in W(\Sigma)$
 $x = A_{m1}, A_{m2}, \dots, A_{mi}$
 $y = A_{m1}, A_{m2}, \dots, A_{mj}$
 $|x| = i$
 $|y| = j$

Se define $w=xy$ (otra notación posible es $w=x.y$) tal que:

$w = A_{m1}, A_{m2}, \dots, A_{mi}, A_{m1}, A_{m2}, \dots, A_{mj}$
 $|w| = |x| + |y| = i + j$
 $w \in W(\Sigma)$

Propiedades

1. Es una **operación cerrada**, es decir, que:

$$\begin{aligned}x &\in W(\Sigma) \\ y &\in W(\Sigma) \\ xy &\in W(\Sigma)\end{aligned}$$

2. Posee la **propiedad asociativa**.

$$x \cdot (yz) = (xy) \cdot z$$

Nota: cuando un conjunto goza de estas dos propiedades se le llama semigrupo.

3. Existe un **elemento neutro**.

$$\begin{aligned}x &\in W(\Sigma) \\ \lambda &\in W(\Sigma) \\ x\lambda &= x\end{aligned}$$

4. **No es conmutativa**.

$$xy \neq yx$$

Cabeza y Cola

Si $z = x.y$, x es **cabeza** de z e y es **cola** de z .

x es **cabeza propia** de z si $y \neq \lambda$.

y es **cola propia** de z si $x \neq \lambda$

Ejemplo:

$$z = ABC$$

Son cabezas de z : λ , A, AB, ABC

Son cabezas propias de z : λ , A, AB

Son colas de z : λ , C, BC, ABC

Son colas propias de z : λ , C, BC

Potencia de una palabra

Se define la **potencia i-ésima**, o de orden i , a una nueva palabra resultante de la concatenación de x consigo misma i veces. Se representa así x^i .

$$x^i = xxx\dots x \quad (i \text{ veces}) \quad i > 0$$

Se cumple que $x^1 = x$

Ejemplo:

$$x = ABC$$

$$x^2 = ABCABC$$

$$x^3 = ABCABCABC$$

La longitud de la potencia de una palabra es $|x^i| = i \cdot |x|$

Reflexión de una palabra

Una reflexión de una palabra es una nueva palabra representada de la forma x^{-1} y se la denomina **palabra refleja** de x . Se define como la secuencia de elementos de la palabra original con orden invertido.

Ejemplo:

$$\begin{aligned}x &= ABC \\ x^{-1} &= CBA\end{aligned}$$

La longitud de la palabra inversa es la misma que la palabra original.

Lenguaje (sobre un alfabeto Σ): es un subconjunto del lenguaje universal $W(\Sigma)$. Es decir que:

$$L \subset W(\Sigma)$$

Conjunto vacío: $\Phi = \{ \}$. Dado un conjunto cualquiera A , $\Phi \subset A$. Existe un lenguaje vacío Φ , $\Phi \subset W(\Sigma)$.

Cardinalidad de un conjunto: dado un conjunto A , su cardinalidad $c(A)$ es el número de elementos del conjunto A .

Cardinalidad de un lenguaje: número de palabras del lenguaje.

Ejemplo:

$$\begin{aligned}c(\Phi) &= 0 \\ c(W(\Sigma)) &= \infty\end{aligned}$$

Lenguaje con únicamente la palabra vacía:

$$\begin{aligned}\{\lambda\} &\neq \Phi, c(\{\lambda\}) = 1 \\ \{\lambda\} &\subset W(\Sigma) \\ \Phi &\subset W(\Sigma) \\ \Sigma &\subset W(\Sigma)\end{aligned}$$

Operaciones con lenguajes

Dados dos lenguajes L_1 y L_2 , definidos sobre el mismo alfabeto Σ , tal que:

$$L_1 \subset W(\Sigma) \text{ y } L_2 \subset W(\Sigma)$$

Unión de dos lenguajes

$$\begin{aligned}L &= L_1 \cup L_2 = \{x / x \in L_1 \text{ OR } x \in L_2\} \\ L_1 \cup L_2 &\subset W(\Sigma)\end{aligned}$$

Propiedades

1. Es una operación cerrada.

$$L_1 \cup L_2 \subset W(\Sigma)$$

2. Cumple la propiedad asociativa.

Dados tres lenguajes, L_1, L_2 y $L_3 \subset W(\Sigma)$, se cumple que $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$

3. Existe un elemento neutro.

Dado un lenguaje cualquiera $L_1 \subset W(\Sigma)$ se cumple que $L \cup \Phi = \Phi \cup L = L$

Nota: las anteriores propiedades definen un monoide.

4. Cumple la propiedad conmutativa.

$L_1 \cup L_2 = L_2 \cup L_1$

Nota: al añadir la propiedad anterior a las tres primeras, se define un monoide abeliano.

5. Cumple la propiedad ídem-potente.

$L \cup L = L$

Concatenación de lenguajes

Dados dos lenguajes $L_1 \subset W(\Sigma)$ y $L_2 \subset W(\Sigma)$, construidos sobre el mismo alfabeto Σ , se llama **lenguaje concatenado** o lenguaje producto de L_1 y L_2 a un nuevo lenguaje L tal que:

$L = L_1L_2 = L_1.L_2 = L_1 \times L_2 = \{x.y \mid x \in L_1 \text{ AND } y \in L_2\}$

Siendo L_1 y $L_2 \neq \Phi$

También se cumple que:

$L\Phi = \Phi = \Phi L$

Propiedades

1. Es una operación cerrada.

$L_1L_2 \subset W(\Sigma)$

2. Cumple la propiedad asociativa.

Dados L_1, L_2 y $L_3 \subset W(\Sigma)$ se cumple que $L_1(L_2L_3) = (L_1L_2)L_3$

3. Existe un elemento neutro.

$L \subset W(\Sigma), L \{\lambda\} = L$

Nota: un lenguaje es BINOIDE, es decir, monoide con respecto a la unión o monoide con respecto a la concatenación. Un lenguaje es un BINOIDE LIBRE generado por el alfabeto Σ .

Potencia de un lenguaje

Dado un lenguaje L se llama **potencia i-ésima** o potencia de orden i, a un nuevo lenguaje $L^i = LLL\dots L$ (i veces)

Propiedades

1. $L^1 = L, L^0 = \{\lambda\}$
2. $L^{i+1} = L^i L = L L^i, i \geq 0$
3. $L^{i+j} = L^i L^j = L^j L^i, i \geq 0$

Clausura positiva (cierre positivo, cerradura positiva)

Dado un lenguaje L, se llama **clausura positiva** del lenguaje L a un nuevo lenguaje:

$$L^+ = L^1 \cup L^2 \cup L^3 \dots = \bigcup_{i=1}^{\infty} L^i$$

Si el lenguaje L, no contiene la palabra vacía, entonces $\lambda \notin L^+$.

Cerradura positiva de un alfabeto Σ :

$\Sigma =$ Alfabeto = Lenguaje

$\Sigma^+ = \Sigma \cup \Sigma \Sigma \cup \Sigma \Sigma \Sigma \cup \dots$

$\lambda \notin \Sigma, \Sigma^+ = W(\Sigma) - \{\lambda\}$

Dado un lenguaje $L \subset W(\Sigma)$, definimos L^* como:

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \dots = \bigcup_{i=0}^{\infty} L^i = \{\lambda\} \cup L^+$$

$$L^+ = L L^* = L^* L$$

También es aplicable en alfabetos: $\Sigma^* = W(\Sigma)$.

Reflexión de un lenguaje

Dado un lenguaje $L \subset W(\Sigma)$, se llama lenguaje reflejo o inverso de L a un nuevo lenguaje $L^{-1} = \{x^{-1} \mid x \in L\}$

Contexto válido

Dado un alfabeto Σ y un lenguaje $L \subset \Sigma^*$, sea $x \in \Sigma^*$ y $x \in L$. Se dice que dos palabras (u, v) pertenecientes a Σ^* son un **contexto válido** de x en el lenguaje L, si la concatenación uxv pertenece a L.

Casos particulares:

1. Si (u, λ) es un contexto válido de x en L, entonces u es un **prefijo** válido de x en L.
2. Si (λ , v) es un contexto válido de x en L, entonces v es un **sufijo** válido de x en L.

Ejemplo:

Dado un alfabeto $\Sigma = \{0,1\}$ tal que $L = \{u \mid |u|=4\}$
 $L = \{0000, 0001, 0010, 0011, \dots, 1111\}$

Consideramos la palabra $x=01$, x no pertenece a L, $L \subset \Sigma^*$
 [no pertenece porque su longitud no es 4]

$x \in \Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, \dots\}$

Contextos válidos de x en L son $(0,0), (0,1), (1,0), (1,1), (\lambda, 00), (\lambda, 01), (\lambda, 10), (\lambda, 11),$ [sufijos]
 $(00, \lambda), (01, \lambda), (10, \lambda), (11, \lambda)$ [prefijos]

Ejemplo:

$X = 1011$
 Un contexto válido es (λ, λ) .

Ejemplo:

$X = 101101$

No existe un (u,v) tal que $uxv \subset L$.
 $|x| > 4$ luego x no pertenece a L .

Relación

Una relación R sobre un conjunto S , es un subconjunto de pares de elementos del conjunto S , si el par de elementos $(a,b) \in R$, se dice que existe la relación R entre los elementos, a y b del conjunto R . Se expresa de las siguientes formas:

$$(a,b) \in R \text{ ó } aRb$$

Se dice que:

- aRb : R es verdadera.
- $a \not R b$: R es falsa.

Propiedades

Dada una relación R sobre S , R puede ser:

- **Reflexiva:** $aRa, \forall a \in S, (a,a)$.
- **Transitiva:** Si aRb y bRc , entonces si se cumple aRc , R es transitiva.
- **Simétrica:** Si aRb y ello implica bRa , R es simétrica.

Ejemplo:

$N = \{0, 1, 2, 3, \dots\}$
 R es $<$
 $R = \{(0,1), (0,2), \dots, (1,2), (1,3), \dots, (2,3), (2,4), \dots\}$

R no es reflexiva. R es transitiva. R no es simétrica.

Nota: la asimetría implica siempre la irreflexividad.

Relación de equivalencia

Una relación R se dice que es una relación de equivalencia si posee las tres propiedades anteriores simultáneamente.

Ejemplo:

$$N = \{0, 1, 2, \dots\}$$

$$R = \{(0, 0), (1, 1), (2, 2), \dots\}$$

Dada una relación de equivalencia R sobre un conjunto S, R divide al conjunto S en S_i clases de equivalencia, y todas las S_i clases son conjuntos no vacíos y disjuntos.

$\forall x \in S$, podemos definir:

$$\{x\} = \{y \in S \mid (x,y) \in R\} = \text{clase de equivalencia.}$$

$$S = S_1 \cup S_2 \cup S_3 \cup \dots \cup S_n$$

$$S_i \neq \Phi$$

$$S_i \cap S_j = \Phi$$

$$i \neq j$$

Venn definió que:

$$\forall a \in S_i, \forall b \in S_j$$

Si $i=j$, entonces:

$$aRb, (a,b) \in R$$

aRb , R es verdadera

Si $i \neq j$:

$$a \not R b, (a,b) \notin R, R \text{ es falsa}$$

Ejemplo:

$$N = \{0, 1, 2, \dots\}$$

$$R = \text{"Tener la misma paridad"}$$

Ejemplo:

$$R = \text{"Tener el mismo resto que la división por 3"}$$

$$A = \{(3 \bmod x)=0\} = \{0, 3, 6, 9, \dots\}$$

$$B = \{(3 \bmod x)=1\} = \{1, 4, 7, \dots\}$$

$$C = \{(3 \bmod x)=2\} = \{2, 5, 8, \dots\}$$

Relación de equivalencia $P_L, S_L, L \in \Sigma^*$

Se dice que existe una relación de equivalencia P_L entre los elementos x e $y \in \Sigma^*$, y se escribe $xP_L y$, si las palabras x e y tienen el mismo conjunto de prefijos válidos en el lenguaje L .

Se dice que existe una relación de equivalencia S_L entre los elementos x e $y \in \Sigma^*$, y se escribe $xS_L y$, si las palabras x e y tienen un único conjunto de sufijos válidos en L .

Derivaciones

Producción (regla o norma de escritura, regla o norma de derivación): dado un alfabeto Σ , una producción es un par ordenado (x,y) en donde $x,y \in \Sigma^*$ y se representa por $x:=y$.

Dado un alfabeto Σ y $x:=y$ una producción sobre las palabras de Σ^* y, sean v y $w \in \Sigma^*$, se dice que w es una **derivación directa** de v , y se expresa como $v \rightarrow w$, si dos palabras $z,u \in \Sigma^*$, tales que $v=zxu$ y $w=zyu$.

v se produce directamente de w.
w es reducción directa de v.

$\Sigma = \{A, B, C, \dots, Z\}$
 $ME ::= BA$

$V = \text{CAMELLO} \rightarrow w = \text{CABALLO}$
 $Z = CA$
 $U = LLO$

CA, LLO son un contexto válido de ME y BA en L.

Dado un alfabeto Σ y un conjunto P de sobre las palabras construidas con símbolos del alfabeto y sean v y w dos palabras del lenguaje universal Σ^* , se dice que w es **derivación** de v y se expresa $v \rightarrow^+ w$ si existe una secuencia finita de palabras u_0, u_1, \dots, u_n , tales que:

$v = u_0, u_0 \rightarrow u_1, \dots, u_{n-1} \rightarrow u_n = w$

La **longitud de la derivación** es el número de pasos necesarios en la realización de la misma.

Si $v \rightarrow^+ w$ es de longitud 1, entonces $v \rightarrow w$ (derivación directa).

Relación de Thue

- **Simétricas:** relación de equivalencia ($v \leftrightarrow w$).
- **No simétricas:** relación de semithue.

Dado un alfabeto Σ , un conjunto P de producciones, y sean v y w dos palabras que pertenecen al lenguaje universal Σ^* ($v, w \in \Sigma^*$). Se dice que existe una relación de Thue entre v y w (y se expresa como $v \rightarrow^* w$) si se verifica que la palabra w se deriva de v o bien $w = v$.

Propiedades

1. **Reflexiva:** $\forall x \in \Sigma^* \quad x \rightarrow^* x$.
2. **Transitiva:** Dadas tres palabras $u, v, w \in \Sigma^*$, si $u \rightarrow^* v$ y $v \rightarrow^* w$, entonces $u \rightarrow^* w$.
3. **Simétrica:** Dadas dos palabras v y w, $v, w \in \Sigma^*$, si $v \rightarrow^* w$ entonces $w \rightarrow^* v$ (puede no tener esta propiedad).

Palabra irreductible (de una clase de equivalencia): aquella palabra de la cual no puede derivarse ninguna otra que sea de menor longitud, aplicando las reglas de producción del conjunto P.

Tenemos las palabras v y w pertenecientes a la misma clase de equivalencia. Tenemos que probar que v y w están en relación de THUE simétrica.

$v \leftrightarrow P$. Irreductible y $w \leftrightarrow P$. Irreductible entonces $v \leftrightarrow w$.

Ejemplo:

$\Sigma = \{0, 1\}$
 $P = \{00 ::= \lambda, 11 ::= \lambda, \lambda ::= 00, \lambda ::= 11\}$
 [las reglas van de P_1 a P_4]

```
v = 10111100101
v ∈ Σ*
w = 111000101
w ∈ Σ*

v = 10111100101 [P2] 101100101 [P2] 1000101 [P1] 10101
10101 es irreductible
W = 111000101 [P2] 1000101 [P1] 10101

v ↔ w
```

```
Ejemplo:

Σ={0,1}
P = {000 ::= 00, 010 ::= 00, 101 ::= 11, 111 ::= 11, ...}

v = 01010 [P2] 0010[P2] 000 [P1] 00

Y de otra forma:

V = 01010 [P3] 0110

Ambas palabras son irreductibles y pertenecen a la misma
clase de equivalencia.
```

Gramáticas

Es un conjunto de normas o reglas que nos definen la estructura de las palabras y frases de un lenguaje.

Formalizar el lenguaje natural

- R₁: Una frase tiene forma de sujeto seguido de verbo.
- R₂: Un sujeto es un pronombre.
- R₃: Un pronombre es "EL" o "ELLA".
- R₄: Un verbo es "DUERME" o "ESCUCHA".

R₁, R₂, R₃ y R₄ forman una gramática. A FRASE, SUJETO y VERBO se les llama variables, categorías o **símbolos no terminales**, pues no forman parte de la frase final, si no que son sustituidos. Se identifican entre <...>.

EL, ELLA, DUERME y ESCUCHA son **símbolos terminales**.

Las reglas que forman una gramática se llaman **reglas de producción gramaticales**. Hay dos tipos de reglas:

- Sintácticas.
- Morfológicas.

Gramática castellana

```
<Frase> ::= <Frase_Nominal><Frase_Verbal>
<Frase_Nominal> ::= <Frase_Nominal><Adjetivo>
<Frase_Nominal> ::= <Artículo><Frase_Nominal>
<Frase_Nominal> ::= <Nombre>
<Frase_Verbal> ::= <Verbo><Complementos>
```

Vamos a llamar a las reglas anteriores, **reglas sintácticas** porque utilizan símbolos no terminales.

- <Nombre> ::= 'niño'
- <Nombre> ::= 'roca'
- <Adjetivo> ::= 'pequeño'
- <Artículo> ::= 'el'
- <Artículo> ::= 'ella'
- <Verbo> ::= 'corre'
- <Complementos> ::= λ

Estas reglas se denominan **reglas morfológicas** porque emplean símbolos terminales.

```

Ejemplo:

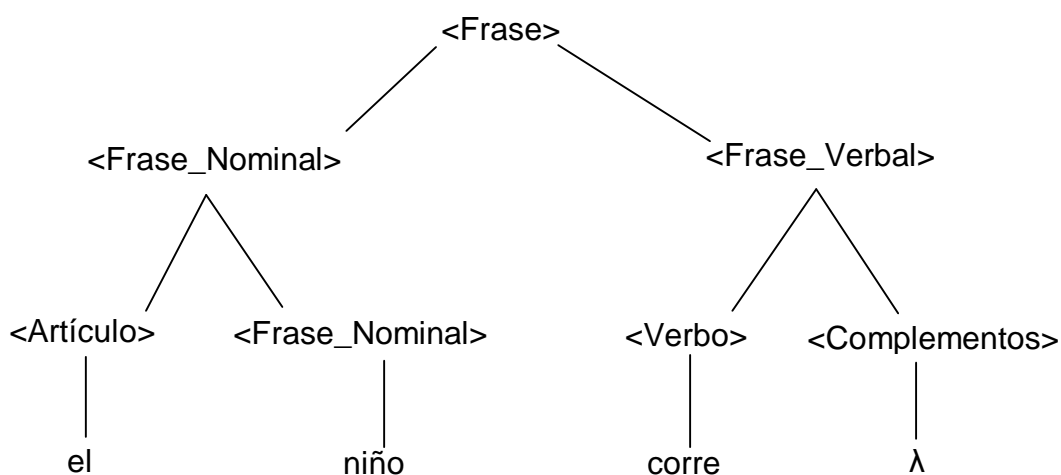
<Frase>
<Frase_Nominal><Frase_Verbal>
<Artículo><Frase_Nominal><Frase_Verbal>
el <Frase_Nominal><Frase_Verbal>
el <Nombre> <Frase_Verbal>
el niño <Frase_Verbal>
el niño <Verbo><Complementos>
el niño corre <Complementos>
el niño corre.
    
```

Sintaxis gramatical: se refiere a la construcción correcta de las frases.

Semántica gramatical: nos indica el significado de las frases sintácticamente correctas.

Formalizar una gramática

Árbol de derivación: estructura arbórea que representa la sucesiva aplicación de las reglas de producción en la construcción de una frase.



Los nodos no hoja son símbolos no terminales, y las hojas son símbolos terminales. Entre dos nodos consecutivos hay una relación directa (representando una regla de producción).

Lenguaje artificial: lenguaje creado por el ser humano que nos permite construir frases sintácticamente y semánticamente correctas, que no s sirve en la resolución de problemas.

Los lenguajes de programación son lenguajes artificiales.

FORTRAN

$y = x^2 + z$, en Fortran lo escribiríamos así: $Y = X**2 + Z$

Reglas sintácticas

```

<Instrucción> ::= <Asignación>
<Asignación> ::= <Identificador = <Expresión>
<Expresión> ::= <Sumando> + <Expresión>
<Sumando> ::= <Factor>
<Sumando> ::= <Factor> * <Sumando>
<Factor> ::= <Identificador>
<Factor> ::= <Número>

```

Reglas morfológicas

```

<Identificador> ::= X
<Identificador> ::= Y
<Identificador> ::= Z
<Identificador> ::= λ
<Número> ::= 0
<Número> ::= 1
<Número> ::= 2

```

Ejemplo:

```

<Instrucción>
<Asignación>
<Identificador> = <Expresión>
Y = <Expresión>
Y = <Sumando> + <Expresión>
Y = <Factor> * <Sumando> + <Expresión>
Y = X * <Factor> * <Sumando> + <Expresión>
Y = X * <Identificador> * <Sumando> + <Expresión>
Y = X * λ * <Sumando> + <Expresión>
Y = X * λ * <Factor> + <Expresión>
Y = X ** <Factor> + <Expresión>
Y = X ** <Número> + <Expresión>
Y = X ** 2 + <Sumando>
Y = X ** 2 + <Factor>
Y = X ** 2 + <Identificador>
Y = X ** 2 + Z

```

Gramática Formal

Es una **cuádrupla**. La vamos a denominar $G = \{ \Sigma_T, \Sigma_N, S, P \}$ donde:

Σ_T es el alfabeto de símbolos terminales (en minúsculas).

Σ_N es el alfabeto de símbolos no terminales (en mayúsculas).

S es el axioma, el símbolo inicial ($S \in \Sigma_N$).

P es el conjunto de reglas de producción.

$$\Sigma_T \cap \Sigma_N = \Phi$$

$$\Sigma_T \cup \Sigma_N = \Sigma$$

P se define de la forma $u ::= v$ donde

$$u \in \Sigma^+, \text{ por lo que } u \neq \lambda$$

$$v \in \Sigma^*$$

$$u = xAy \text{ (} x, y \in \Sigma^+ \text{ y } A \in \Sigma_N \text{)}$$

Ejemplo:

$$G = \{ \{a,b\}, \{S\}, S, P = \{ S ::= aSb, S ::= ab \}$$

$$\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \dots = \{a,b,S\} \cup \{a,b,S\}^2 \cup \{a,b,S\}^3 \cup \dots$$

$$= \{a,b,S,aa,ab,ba,bb,aS,Sa,bS,SS,aaa,aab,\dots\}$$

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots = \{\lambda\} \cup \Sigma^+ = \{\lambda, a,b,S,aa,\dots\}$$

Notación de Backus-Naur

$$P = \{ u ::= v, u ::= w, u ::= z, \dots \} = P = \{ u ::= v | w | z | \dots \}$$

Lenguaje asociado a una gramática

Dada una gramática G, se llama lenguaje asociado a una gramática G y se representa por $L(G) = \{ x | S \rightarrow^* x \text{ AND } x \in \Sigma_T^* \}$.

Forma sentencial X de una gramática G si se verifica que: $S \rightarrow^* x$, es decir, si existe una relación de Thue entre S.

Si la forma sentencial $x \in \Sigma_T^*$ x puede ser λ . Se dice que x es una **instrucción** o **sentencia** de la gramática G.

$$L(G) = \{ \text{Todas las sentencias (o instrucciones) de la gramática G} \}$$

$$= \{ x | S \rightarrow^* x \text{ AND } x \in \Sigma_T^* \}$$

Ejemplo:

$$G = (\{a,b\}, \{S\}, S, P = \{ s ::= aSb | ab \})$$

$$S \rightarrow ab$$

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaabbb$$

...

$$L(G) = \{ ab, aabb, aaabbb, \dots \} = \{ a^n b^n | n \geq 1 \} = \{ a^1 b^1, a^2 b^2, a^3 b^3, \dots \}$$

Ejemplo:

$$G = (\{a,b\}, \{S,A,B\}, S, P = \{ S ::= aB | bA, A ::= bAA | aS | a, B ::= aBB | bS | b \})$$

$$L(G) = \{ ab, ba, abba, abab, \dots \}$$

$$S \rightarrow aB \rightarrow ab$$

$$S \rightarrow bA \rightarrow abS \rightarrow abbA \rightarrow abba \rightarrow abaB \rightarrow abab$$

$$L(G) = \{ x \in \Sigma_T^* | N_a(x) = N_b(x) \}$$

N_a es el número de letras 'a'.

Dada una gramática G y una forma sentencial $v=xuv$, se dice que U es una frase de la forma sentencial v respecto de un símbolo no Terminal $u \in \Sigma_N$ si $S \rightarrow^* xUy$, y además $U \rightarrow^+ u$.

Si U es una forma sentencial de G , entonces todas las frases de U serán formas sentenciales de G . Una forma $v=xuy$ se llama frase simple si $S \rightarrow^* xUy$ y $U \rightarrow u$.

Recursividad

Una gramática G es recursiva en $U \in \Sigma_N$ si $U ::= xUy$.

Si $U \rightarrow +Uy$ la gramática es **recursiva a la izquierda**.

Si $U \rightarrow xU+$ la gramática es **recursiva a la derecha**.

Si $x=\lambda \rightarrow U ::= Uy$, es decir, la regla de producción es recursiva a la izquierda.

Si $y=\lambda \rightarrow U ::= xU$, es decir, la regla de producción es recursiva a la derecha.

Dada una gramática, el lenguaje descrito por ella, $L(G)$ puede ser finito o infinito. Para que un lenguaje descrito por G sea infinito, la gramática tiene que ser recursiva.

Tipos de gramática

Noam Comsky dividió las gramáticas en cuatro grupos:

De tipo 0 (G_0)

De tipo 1 (G_1)

De tipo 2 (G_2)

De tipo 3 (G_3)

$$G_3 \subset G_2 \subset G_1 \subset G_0$$

Gramáticas de tipo 0

Sus reglas de producción son de la forma $u ::= v$, siendo $u \in \Sigma^*$ y $v \in \Sigma^*$.

$$u = xAy$$

$$x, y \in \Sigma^*, A \in \Sigma_N$$

x e y pueden ser λ .

$L(G_0)$ = lenguaje sin restricciones.

Nota: los lenguajes de programación no suelen estar generados por gramáticas de tipo 0.

Otro tipo de gramática más restringida, llamada **gramática estructurada por frases**, en la que sus reglas de producción son del tipo: $xAy ::= xvy$.

$|xAy| > |xy|$ si $v = \lambda$. A este tipo de reglas de producción se les denomina **compresoras**.

Ejemplo:

$$G_0 = (\{a, b\}, \{A, B, C\}, A, P)$$

$$P = \{A ::= aABC \mid abC, CB ::= BC, bB ::= bb, bC ::= b\}$$

G_0 no es una gramática estructurada por frases porque no todas las reglas son estructuradas (las dos primeras no lo son).

Vamos a crear una gramática que sea estructurada y genere el mismo lenguaje que G_0 .

Añadimos x e y a Σ_N .

$CB ::= BC$ sustituimos por $CB ::= XB$

$CB ::= BC$ sustituimos por $XB ::= XY$
 $CB ::= BC$ sustituimos por $XY ::= BY$
 $CB ::= BC$ sustituimos por $BY ::= BC$

Ahora sí tenemos una gramática G_0' estructurada por frases.
 $L(G_0) = L(G_0')$

Una gramática es compresora si tiene al menos una regla de producción de tipo compresora (su parte izquierda tiene mayor longitud que la derecha).

Gramáticas de tipo 1

Son aquellas cuyas reglas de producción tienen la forma:

$xAy ::= xvy$
 $A \in \Sigma_N$
 $x, y \in \Sigma^*$, luego x e y pueden ser λ .
 $\Sigma = \Sigma_T \cup \Sigma_N$

Este tipo de gramática **no tiene reglas de producción compresoras**. Sin embargo, se admite una excepción que permite que una de las reglas de producción sea del tipo $S ::= \lambda$.

Si la gramática no contiene esta regla, no puede generar un lenguaje que contenga la palabra vacía:

$\lambda \notin L(G_1)$, si la regla de producción $S ::= \lambda \notin P$.

$G_0, L(G_0) \rightarrow$ Lenguaje sin restricciones.

$G_1, L(G_1) \rightarrow$ Lenguajes dependientes del contexto (context sensitive).

Se les denomina dependientes del contexto, porque dado

$xAy ::= xvy$
 A y v tienen el mismo contexto (x, y) .

$G_1 \subset G_0$

Ejemplo:

$G_1 = (\{a, b, c\}, \{S, B, C\}, S, P)$
 $P = \{S ::= aSBC \mid aBC, bB ::= bb, bC ::= bc, cC ::= cc, CB ::= BC, aB ::= ab\}$
 $L(G_1) = \{a^n b^n c^n \mid n > 0\}$

Gramáticas de tipo 2

Sus producciones son de los forma $A ::= v$

$A \in \Sigma_N$ (un único símbolo)

$v \in \Sigma^* = (\Sigma_T \cup \Sigma_N)^*$

v puede ser λ .

$\lambda \in L(G_2)$ si v fuese λ

Este tipo de gramática son **independientes del contexto** (context free).

Los lenguajes de programación generalmente pertenecen a este tipo.

Dada una G_2 que genere un lenguaje $L(G_2)$ en la que $\lambda \in L(G_2)$, siempre existirá otra gramática G_2' equivalente (distinta a G_2 pero que genera el mismo lenguaje), que **no** tiene reglas de producción de la forma $A ::= \lambda$ (reglas Lambda).

Nota: G_2 es equivalente a G_2' si $L(G_2)=L(G_2')$

Ejemplo:

$$G_2 = (\{a, b\}, \{S\}, S, \{S ::= aSb \mid ab\})$$

$$S \rightarrow ab$$

$$S \rightarrow aSb \rightarrow aabb$$

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaabbb$$

$$L(G_2) = \{a^n b^n \mid n > 0\} = L(G_0)$$

Un lenguaje puede estar generado por varias gramáticas. Sin embargo, una gramática solo puede generar un lenguaje.

Gramáticas de tipo 3

Existen dos clases:

1. Lineales por a izquierda.
2. Lineales por la derecha.

Las lineales por la izquierda tienen las siguientes reglas de producción:

$$A ::= a \quad A \in \Sigma_N$$

$$A ::= Va \quad a \in \Sigma_T$$

$$A ::= \lambda \quad V \in \Sigma \in N$$

Las lineales por la derecha tienen las siguientes reglas de producción:

$$A ::= a \quad A \in \Sigma_N$$

$$A ::= aV \quad a \in \Sigma_T$$

$$S ::= \lambda \quad V \in \Sigma \in N$$

Los lenguajes generado por gramáticas de tipo3 se llaman **lenguajes regulares**.

Ejemplo:

$$G_3 = (\{0, 1\}, \{A, B\}, A, \{A ::= B1 \mid 1, B ::= A0\})$$

G_3 es una gramática lineal por la izquierda.

$$L(G_3) = \{1, 101, 10101, \dots\} = \{1 \cdot (01)^n \mid n=0, 1, 2, \dots\}$$

$L(G_3)$ es un lenguaje regular.

$$G_3' = (\{0, 1\}, \{A, B\}, A, \{A ::= 1B \mid 1, B ::= 0A\})$$

$$L(G_3') = \{1 \cdot (01)^n \mid n \geq 0\}$$

$L(G_3')$ es un lenguaje regular.

$$L(G_3) = L(G_3')$$

Dada una gramática lineal por la derecha, siempre existe otra lineal por la derecha equivalente, pero sin reglas de producción de tipo $A ::= aS$, en donde S es el axioma, $A \in \Sigma_N$ y $a \in \Sigma_T$.

Para construir G_3 , añadimos un símbolo no Terminal B al alfabeto Σ_N de G_3 . Por cada regla de producción de forma $S ::= x$, siendo $x \in \Sigma^+$, añadimos otra $B ::= x$ y, finalmente, transformamos todas las reglas $A ::= aS$ en $A ::= aB$

Ejemplo:

Dada la gramática $G_3 = (\{a,b\}, \{S,A\}, S, \{S ::= bA, A ::= aS | a\})$
 Lineal por la derecha con producción $A ::= aS$.

$L(G_3) = \{ba, baba, bababa, \dots\} = \{(ba)^n | n > 0\}$

$G_3' = (\{a,b\}, \{S,A,B\}, \{S ::= bA, A ::= aB | a, B ::= bA\})$
 Esta también es lineal por la derecha, pero no tiene producciones de la forma $A ::= aS$.

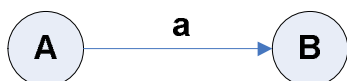
$L(G_3) = L(G_3')$

Este teorema también es aplicable a las gramáticas lineales por la izquierda.

Dada una gramática G_3 lineal por la derecha, siempre existe otra gramática equivalente G_3' lineal por la izquierda (como se muestra en el primer ejemplo).

Si G_3 es lineal por la derecha y tuviese reglas de producción del tipo $A ::= aS$ construiríamos otra gramática G_3'' lineal por la derecha equivalente pero sin reglas $A ::= aS$. A partir de G_3'' vamos a construir un **grafo**.

1. El grafo tendrá tantos nodos como símbolos no terminales haya en el alfabeto Σ_{N+1} . Cada nodo se etiqueta con los símbolos no terminales. El nodo adicional se etiqueta con λ .
2. Cada regla de producción de la forma $A ::= aB$ se indica así:



Cada regla de producción de la forma $A ::= a$ se indica así:



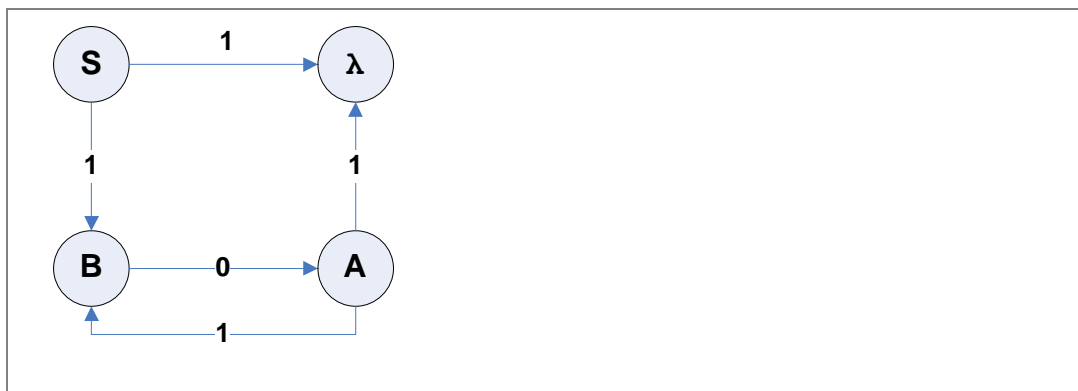
Cada regla de producción de la forma $S ::= \lambda$ se indica así:



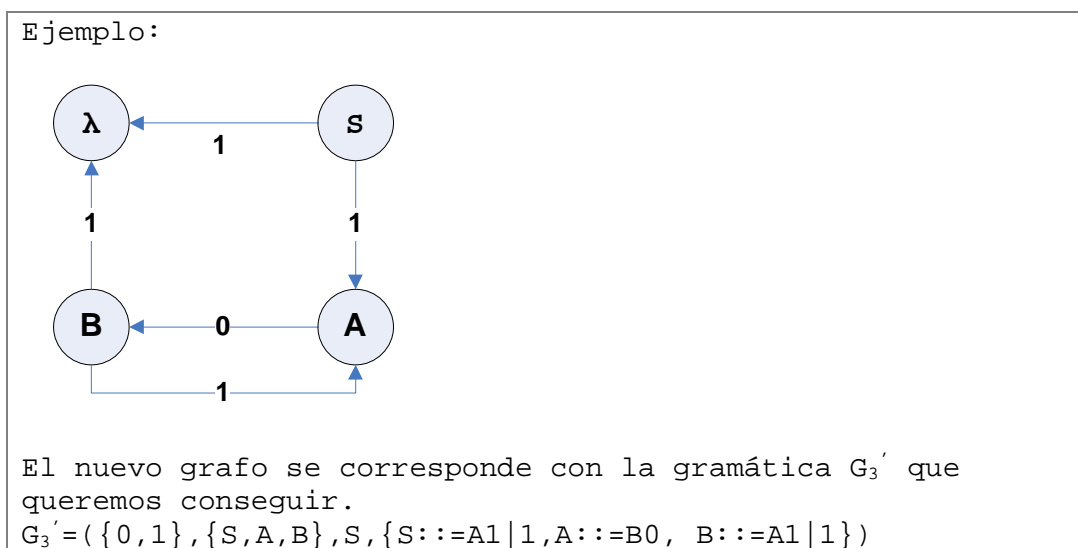
Ejemplo:

$G_3 = (\{0,1\}, \{S,B\}, S, \{S ::= 1B | 1, B ::= 0S\})$
 Lineal por la derecha con producciones del tipo $B ::= 0S$

$G_3'' = (\{0,1\}, \{S,B,A\}, S, \{S ::= 1B | 1, B ::= 0A, A ::= 1B | 1\})$
 Lineal por la derecha pero sin producciones del tipo $B ::= 0S$



Las palabras se forman recorriendo el grafo a partir del axioma y terminando en λ. Ahora para construir la gramática lineal por la izquierda, intercambiamos las etiquetas de los nodos S y λ e invertimos las direcciones de las flechas.

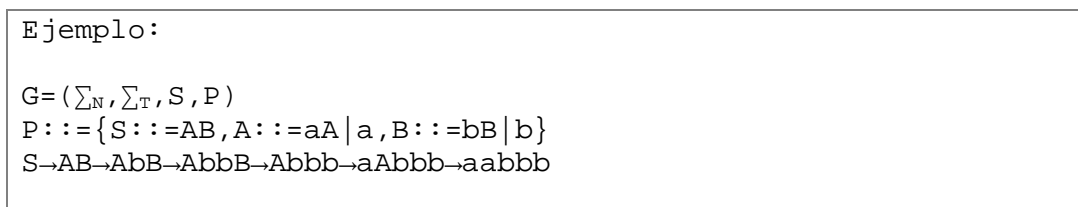


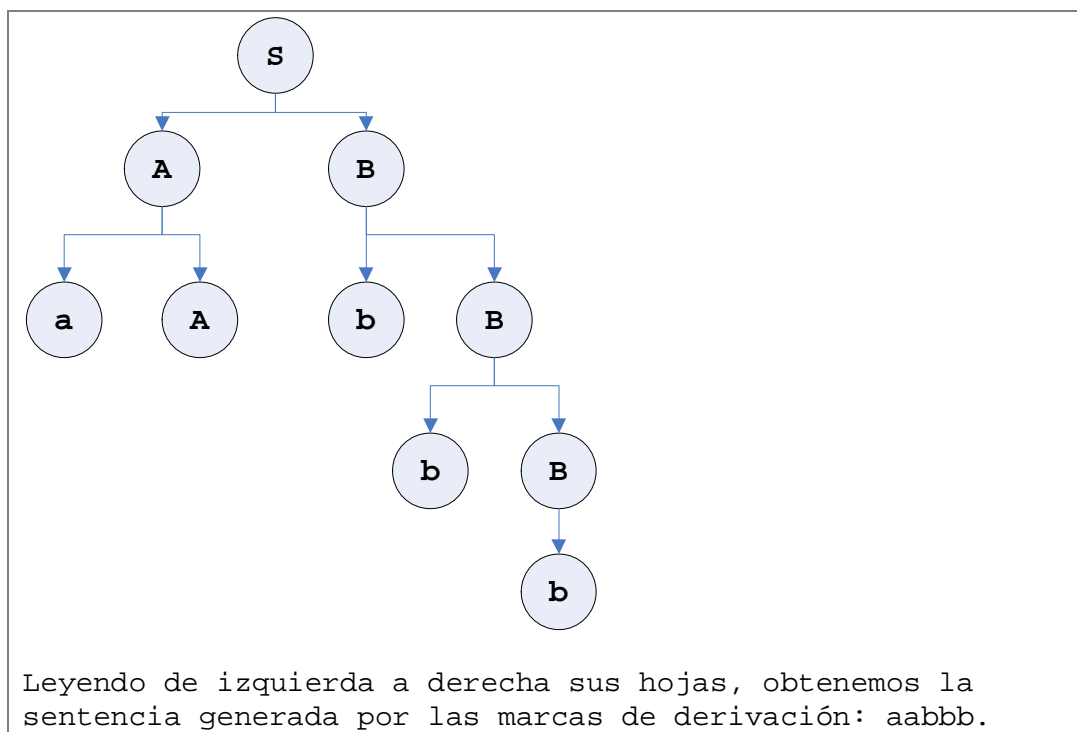
Árbol de derivación

Constan de **nodos** y **ramas**. Los nodos se etiquetan con símbolos del alfabeto Terminal, no Terminal, o con la cadena vacía λ. Existen unos nodos finales llamados **hojas** que pertenecen al alfabeto terminal.

Las ramas que van de un nodo a una hoja se llaman **ramas terminales**. El nodo raíz es único y se le etiqueta con el axioma. Las hojas leídas de izquierda a derecha nos muestran la sentencia o palabra derivada del axioma y perteneciente al lenguaje.

Derivación directa es una parte del árbol de derivación que parte de un nodo padre y termina en un nodo hijo.





Se pueden construir distintos árboles de derivación para una misma palabra.

$S \rightarrow AB \rightarrow aAB \rightarrow aaB \rightarrow aabB \rightarrow aabbB \rightarrow aabbb$

Esta derivación (distinta a la anterior) produce la misma palabra y genera el mismo árbol de derivación.

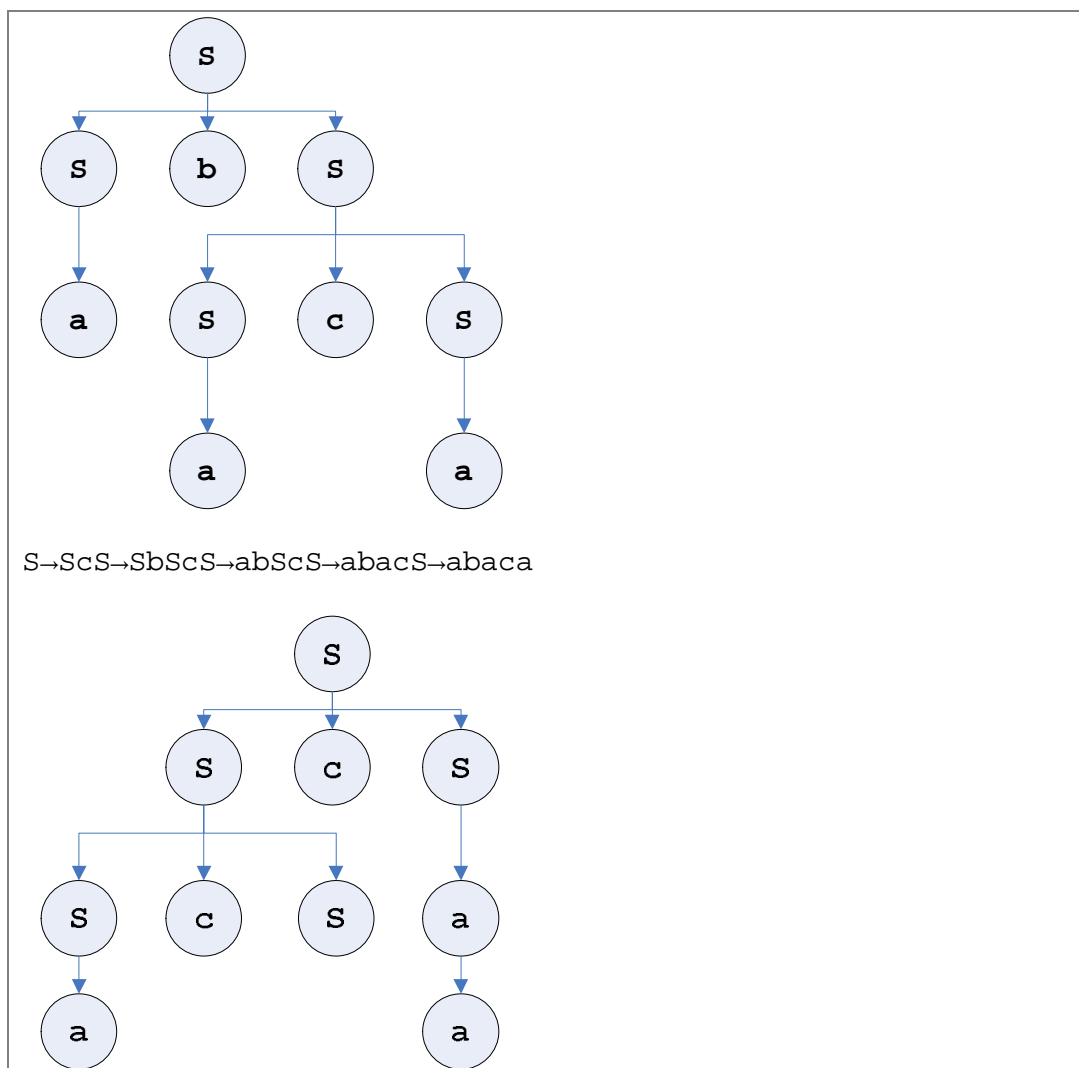
Dado un árbol, podemos deducir distintas derivaciones. Dada una derivación, solo podemos generar un árbol a partir de ella.

Una gramática se dice que es **ambigua** si para una misma sentencia genera distintos árboles de derivación.

Ejemplo:

$G = (\{a, b, c\}, \{S\}, S, P)$
 $P = \{S ::= SbS \mid SaS \mid a\}$

$S \rightarrow SbS \rightarrow SbSaS \rightarrow SbSca \rightarrow Sbaca \rightarrow abaca$



Luego 'abaca' del ejemplo anterior es una **sentencia ambigua**, porque se puede representar con árboles de derivación distintos. Basta que una gramática genere una sentencia ambigua para que sea una gramática ambigua.

Toda gramática ambigua puede transformarse en otra no ambigua equivalente.

Ejemplo:

$$G = (\{a, b, c, =, +, *, (,)\}, \{A, E, I\}, A, P)$$

$$P = \{A ::= I = E, I ::= a | b | c, E ::= E + E | E * E | (E) | I\}$$

$A \rightarrow I = E \rightarrow a = E \rightarrow a = E + e \rightarrow a = I + E \rightarrow a = b + E \rightarrow a = b + E * E \rightarrow a = b + I * E \rightarrow$
 $a = b + c * E \rightarrow a = b + c * I \rightarrow a = b + c * a$

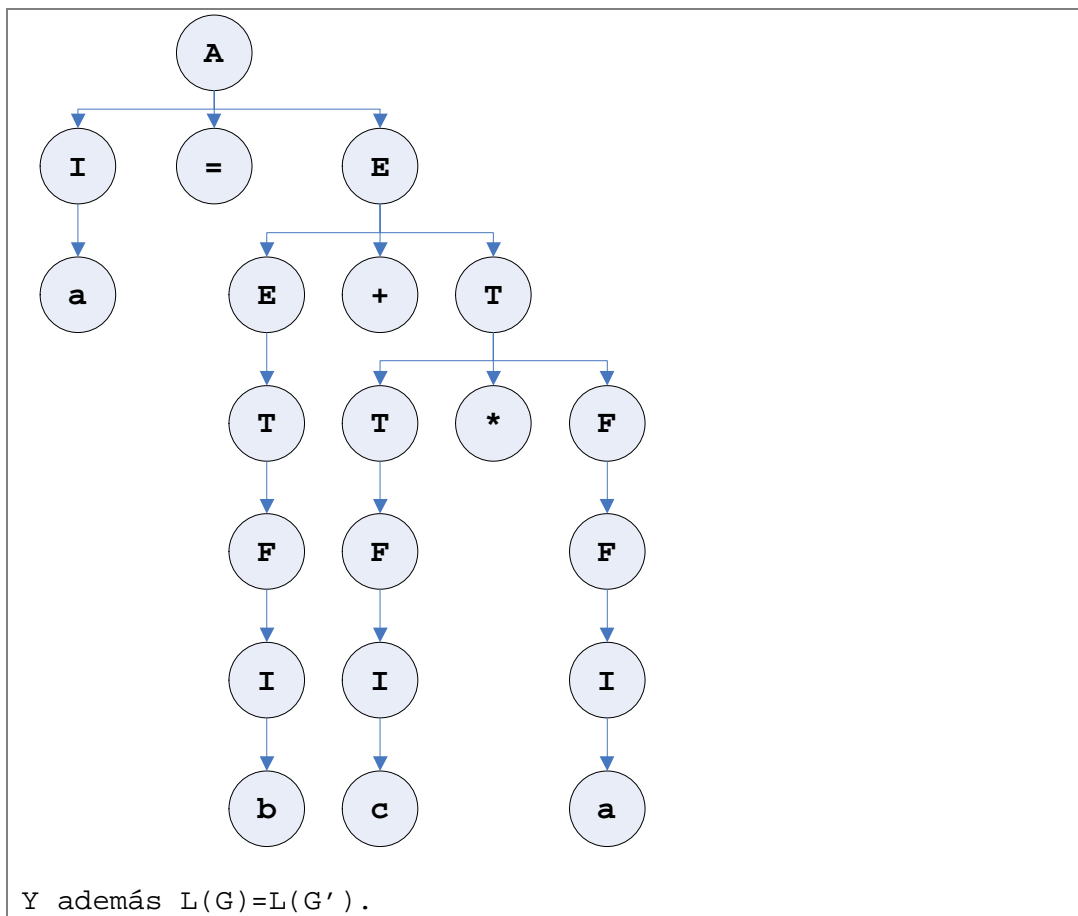
Si intentamos generar la misma sentencia variando las reglas de producción, generaremos otro árbol, luego G es una gramática ambigua.

Para eliminar la ambigüedad, vamos a introducir símbolos no terminales.

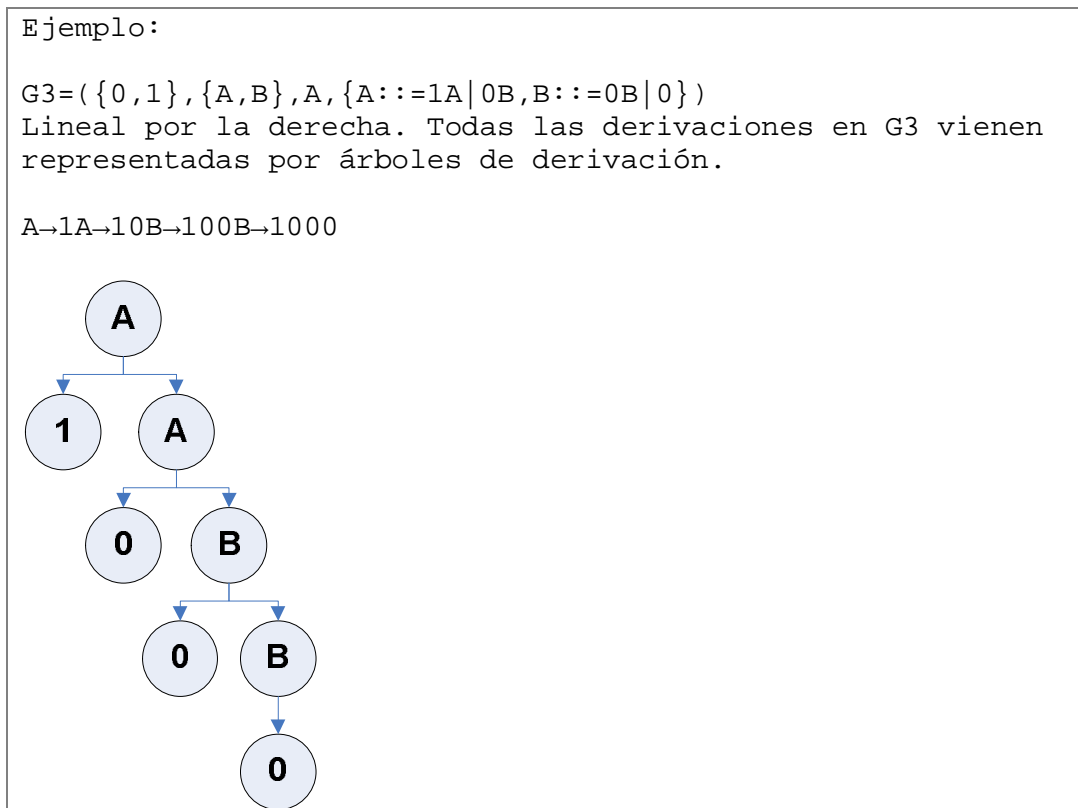
$$G' = (\{a, b, c, =, +, *, (,)\}, \{A, E, I, T, F\}, A, P')$$

$$P' = \{A ::= I = E, I ::= a | b | c, E ::= T | E + T, T ::= F | T * F, F ::= (F) | I\}$$

Ahora G' no es ambigua:



Si todas las gramáticas que generan un lenguaje son ambiguas, el lenguaje se dice que es **intrínsecamente (o inherentemente) ambiguo**.



Ejemplo:

$$G = (\{a, b, c\}, \{A, B, C, E, F\}, A, P)$$

$$P = \{A ::= BF, B ::= EC, E ::= a, C ::= b, F ::= c\}$$

Este es un diagrama de las derivaciones posibles. NO es un árbol de derivación.

$A \rightarrow BF \rightarrow ECF \rightarrow aCF \rightarrow abF \rightarrow abc$

Hemos aplicado producciones al símbolo que está más a la izquierda. Esta derivación se denomina **extrema izquierda**.

$A \rightarrow BF \rightarrow BC \rightarrow ECc \rightarrow Ebc \rightarrow abc$

Hemos aplicado producciones al símbolo Terminal más a la derecha. Esta es una derivación **extrema derecha**.

Gramáticas independientes de contexto (GIC)

1. Cada símbolo que pertenece al alfabeto terminal, o bien al no terminal, deberá aparecer en, por lo menos, alguna derivación de alguna palabra.
2. No deberán existir reglas de producción de la forma $A ::= B$, en donde $A, B \in \Sigma_N$.
3. Si $\lambda \notin L(G)$, no deberán existir reglas de producción de tipo $A ::= \lambda$, $A \in \Sigma_N$.

Pueden existir GIC que produzcan símbolos que no se van a utilizar después, o que lleguen a la misma derivación de la cual se partió. Si una gramática cumple esto, se denomina **gramática sucia**.

Podemos eliminar estas redundancias *limpiando* la GIC, mediante estos pasos:

1. Comprobamos si la GIC no genera ninguna palabra (ni siquiera λ), es decir, si $L(GIC) = \Phi$.
2. Eliminamos los símbolos inútiles de la GIC.

Ejemplo:

$$GIC = (\Sigma_T, \Sigma_N, S, \{S ::= Aa \mid B \mid D, B ::= b, A ::= aA \mid bA \mid B, C ::= abd\})$$

Si hay reglas de producción de este tipo $P_i ::= \alpha C \beta$, $\alpha, \beta \in \Sigma^*$

P_i no aporta información y la eliminamos.

C es un símbolo inútil.

D es otro símbolo inútil, pues no puede derivarse ningún terminal a partir de él.

B es un símbolo redundante, que podría sustituirse por $A ::= b$.

Del alfabeto de terminales, el ' d ' nunca puede accederse, así que lo eliminamos.

Símbolos inútiles

Dada una GIC, el símbolo $X \in \Sigma_N$ es **útil** si aparece en alguna derivación

$S \rightarrow^* \alpha X \beta \rightarrow^* w$

$\alpha, \beta \in \Sigma^*$

$w \in \Sigma_T^*$

En caso contrario es inútil.

Reglas de producción λ o producciones lambda

$A ::= \lambda, A \in \Sigma_N$

Si $\lambda \notin L(GIC)$ entonces el conjunto P de la GIC, $A ::= \lambda \notin P$.

Si $\lambda \in L(GIC)$ entonces no podemos eliminar dichas producciones.

Una GIC se dice que es sin producciones lambda si:

- No hay ninguna regla λ , en el conjunto P .
- Si solo existe $S ::= \lambda$ y además S no aparece en la parte derecha de ninguna regla de producción del conjunto P

Ejemplo:

$GIC = (\Sigma_T, \Sigma_N, S, \{S ::= ABCBCD, A ::= CD, C ::= a \mid \lambda, B ::= cb, D ::= bD \mid \lambda\})$
 $\lambda \notin L(GIC)$

No podemos eliminar directamente las reglas de producción con λ , pues eliminamos funcionalidad. Debemos sustituirlas.

Ejemplo:

$GIC = (\Sigma_T, \Sigma_N, S, P)$

$P = \{S ::= ABCBCD, A ::= CD, C ::= a \mid \lambda, B ::= Cb, D ::= bD \mid \lambda\}$

Esta gramática tiene reglas lambda ($C ::= \lambda, D ::= \lambda$), pero λ no pertenece al lenguaje generado por la gramática.

$GIC_1 = (\Sigma_T, \Sigma_N, S, P_1)$ no tiene reglas lambda.

$L(GIC_1) = L(GIC)$

$P_1 = \{S ::= ABC_1BC_2D, \dots\}$

$\{C_1, C_2, D\}$ contienen reglas lambda. Vamos a formar combinaciones de estos elementos:

$\Phi, \{C_1\}, \{C_2\}, \{D\}, \{C_1, C_2\}, \{C_1, D\}, \{C_2, D\}, \{C_1, C_2, D\}$

Vamos a sustituir producciones por la original. Cada nueva producción es la producción original menos cada uno de los subconjuntos:

$$\begin{aligned} ABC_1BC_2D - \Phi &= ABC_1BC_2D \\ ABC_1BC_2D - C1 &= ABBC_2D \\ ABC_1BC_2D - C2 &= ABC_1BD \\ \dots \end{aligned}$$

Ahora podemos eliminar las reglas λ , realizando la misma operación para **todas** las producciones. Si aparecen reglas duplicadas, las eliminamos. También puede ser necesario renombrar símbolos.

Por ejemplo $A::=C_1C_2$ y $C::=\lambda \rightarrow A::=C$ (pues eliminamos subíndices). Se aparecen reglas del estilo $A::=A$, la eliminamos.

Decimos que en una GIC, un **símbolo es anulable** si se verifica que:

1. $A \in \Sigma_N$ y $A::=\lambda \in P$
2. $A \in \Sigma_N$ y existe $n \geq 1$ tal que $A::=B_1|B_2|\dots|B_n \in P$ siendo $B_i \in \Sigma_N$ y B_i símbolo anulable.

Más formalmente, un símbolo es anulable si A está en relación de THUE con λ , es decir, $A \rightarrow^* \lambda$.

Dada una $GIC=(\Sigma_T, \Sigma_N, S, P)$ con reglas lambda, queremos encontrar una GIC' equivalente sin dichas reglas, tal que $L(GIC)=L(GIC')$:

1. Hacemos $P'=P$.
2. Buscamos los símbolos anulables
3. Para cada producción $A::=\alpha \in P$ se añaden al conjunto P' todas las producciones que se obtienen borrando de la cadena α cualquier subconjunto de símbolos anulables.
4. Eliminamos las producciones con λ .

En el ejemplo anterior resultaría:

$$GIC'=(\Sigma_T, \Sigma_N, S, P')$$

$$P'=\{$$

$$S::=ABCBCD | ABBCD | ABCBD | ABCBC | BCBCD | ABBD | ABCD | ABBC | BBBCD | BCBD$$

$$| BCBC | ABB | BCD | BBC | BBD | BB,$$

$$A::=CD | C | D,$$

$$C::=a,$$

$$D::=bD | b$$

$$\}$$

Ejemplo:

$$GIC=(\{a, b, c\}, \{S, A, B\}, S, P)$$

$$P=\{S::=BAAB, A::=aAc | cAa | \lambda, B::=AB | bB | \lambda\}$$

Tiene dos reglas lambda. Todos los símbolos son anulables. Realizando todos los pasos anteriormente explicados, resulta:

$GIC' = (\{a, b, c\}, \{S, A, B\}, S, P')$
 $P' = \{$
 $S ::= BAAB \mid AAB \mid BAB \mid BAA \mid AB \mid BB \mid BA \mid AA \mid B \mid A,$
 $A ::= aAc \mid aAa \mid ac \mid ca,$
 $B ::= AB \mid bB \mid A \mid b$
 $\}$
 $L(GIC) = L(GIC') + \{\lambda\}$
 Para que el lenguaje coincida, incluimos λ al final de S , tal que $S ::= BAAB \mid \dots \mid \lambda$. Esta regla lambda no se considera como tal.
 Ahora ambos lenguajes realmente coinciden.

Reglas de producción unitarias (o producciones unitarias)

$A ::= B \quad A, B, C \in \Sigma_N$
 $A ::= C$
 $C ::= B \mid \lambda$

Podría simplificarse tal que:

$A ::= B \mid \lambda$

Ejemplo:

$GIC = (\Sigma_T, \Sigma_N, S, P)$
 $P = \{S ::= S+T \mid T, T ::= T * F \mid F, F ::= (S) \mid a$
 $S ::= T \text{ y } T ::= F \text{ son unitarias.}$
 $GIC' = (\Sigma_T, \Sigma_N, S, P')$
 $P' = \{S ::= S+T \mid T * F \mid (S) \mid a, T ::= T * F \mid (S) \mid a, F ::= (S) \mid a\}$
 Ya no tiene producciones unitarias.

Dada una gramática :

$GIC = (\{a\}, \{S, A, B\}, S, \{S ::= AB \mid a, A ::= a\})$
 Es una gramática sucia, pues hay símbolos redundantes.
 $GIC' = (\{a\}, \{S\}, S, \{S ::= a\})$
 Esta gramática está limpia y $L(GIC) = L(GIC')$

Dada una GIC *sucia*, siempre podemos obtener otra GIC' sin símbolos inútiles, sin reglas lambda y sin producciones unitarias, equivalente a la original.

Forma normal de Chomsky (CNF)

Dada una GIC se puede poner bajo forma normal de Chomsky que no genere la palabra vacía. A estas gramáticas las llamaremos CNF.

Las producciones de gramáticas en CNF, siempre tienen esta forma.

$A ::= BC$	$A, B, C \in \Sigma_N$
$A ::= a$	$a \in \Sigma_T$

Dada una GIC = $(\{a, b\}, \{S, A, B\}, S, P)$

$P = \{S ::= bA \mid aB, A ::= bAA \mid aS \mid a, B ::= aBB \mid bS \mid b\}$

Vamos a construir una gramática CNF equivalente.

Creamos producciones para que sean de la forma correcta según la CNF, por ejemplo:

Sustituimos $S ::= bA$ por $S ::= C_bA$ y $C_b ::= b$

La gramática resultante es:

CNF = $(\{a, b\}, \{S, A, B, C_a, C_b, D_1, D_2\}, S, P')$

$P' = \{$

$S ::= C_bA \mid C_aB,$

$A ::= C_aS \mid C_bD_1 \mid a,$

$B ::= C_bS, C_aD_2, b,$

$D_1 ::= AA,$

$D_2 ::= BB,$

$C_a ::= a,$

$C_b ::= b$

$\}$

Forma normal de Greibach (GNF)

Dada una GIC, la denominaremos GNF y cumple que:

1. GIC = $(\Sigma_T, \Sigma_N, S, P)$ si $A ::= \alpha_1 B \alpha_2 \in P$ y $A, B \in \Sigma_N$, $\alpha_1, \alpha_2 \in \Sigma^*$, y las demás producciones del símbolo B son de la forma $B ::= \beta_1 \mid \beta_2 \mid \dots \mid \beta_N$ en donde $\beta_i \in \Sigma^*$, en la que se puede eliminar la producción $A ::= \alpha_1 B \alpha_2$ simplemente añadiendo $A ::= \alpha_1 \beta_i \alpha_2$ al conjunto P'.
2. Dada una GIC = $(\Sigma_T, \Sigma_N, S, P)$ si $A ::= A \alpha_1 \mid A \alpha_2 \mid \dots \mid A \alpha_r \in P$ y son todas las producciones en que interviene el símbolo A (recursividad) y si $A ::= \beta_1 \mid \beta_2 \mid \dots \mid \beta_s$, $\beta_i \in \Sigma^*$, existe otra GIC' equivalente (no recursiva en A) sustituyendo las producciones A por las siguientes:

- a. Si $1 \leq i \leq s$ $A ::= \beta_i, A ::= \beta_i B$
- b. Si $1 \leq i \leq r$ $B ::= \alpha_i, B ::= \alpha_i B$

Ahora tenemos **recursividad por la derecha**:

$A ::= \beta_1 \mid \beta_2 \mid \dots \mid \beta_s \mid \beta_1 B \mid \beta_2 B \mid \dots \mid \beta_s B$

$B ::= \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_r \mid \alpha_1 B \mid \alpha_2 B \mid \dots \mid \alpha_r B$

GIC' es equivalente pero recursiva a la derecha.

Toda gramática GIC que genere un lenguaje $L(GIC)$ que no contenga la palabra vacía puede ponerse bajo la forma normal de Greibach, en la que las producciones son de la forma:

$A ::= a \alpha$ $A \in \Sigma_N, \alpha \in \Sigma_N^*, a \in \Sigma_T$

Y podemos **convertir una CNF en GNF**

$$\text{CNF} = (\{a, b\}, \{A_1, A_2, A_3\}, A_1, P)$$

$$P = \{A_1 ::= A_2 A_3, A_2 ::= A_3 A_1 \mid b, A_3 ::= A_1 A_2 \mid a\}$$

Para ello tenemos que seguir los siguientes pasos:

1. Poner cada parte derecha de las producciones A_1 , A_2 y A_3 , comenzando por terminales o no terminales, con el mayor subíndice.

En A_1 , el A_3 debe ir a la izquierda del A_2 .
 En A_3 , el A_2 debe ir a la izquierda del A_1 .

$A_3 ::= A_1 A_2$, lo sustituimos por $A_3 ::= A_2 A_3 A_2$, es decir, sustituimos A_1 por su producción. Al realizar este paso obtenemos recursividad.

Vamos a sustituir la cabeza de $A_3 ::= A_2 A_3 A_2$ por $A_3 A_2$ y b (sustituimos A_2 por sus producciones).

$$A_3 ::= A_3 A_1 A_3 A_2$$

$$A_3 ::= b A_3 A_2$$

$$A_3 ::= a$$

Tenemos que crear nuevos símbolos:

$$A_3 ::= A_3 \alpha_1$$

$$\alpha_1 ::= A_1 A_3 A_2$$

$$A_3 ::= b A_3 A_2 = \beta_1$$

$$A_3 ::= \beta_2$$

$$A_3 ::= b A_3 A_2 \beta_1$$

$$A_3 ::= b A_3 A_2 B_3 = \beta_1 B_3$$

$$A_3 ::= a B_3 = \beta_2 B_3$$

$$B_3 ::= A_1 A_3 A_2 = \alpha_1$$

$$B_3 ::= A_1 A_3 A_2 B_3 = \alpha_1 B_3$$

Al final obtenemos esto:

$$A_1 ::= A_2 A_3$$

$$A_2 ::= A_3 A_1 \mid b$$

$$A_3 ::= b A_3 A_2 B_3 \mid a B_3 \mid b A_3 A_2 \mid a$$

$$B_3 ::= A_1 A_3 A_2 \mid A_1 A_3 A_2 B_3$$

2. Todas las producciones con A_3 en la parte izquierda, tienen la parte derecha empezando con un símbolo terminal.

$A_2 ::= A_3 A_1$ la convertimos en las siguientes:

$$A_2 ::= b A_3 A_2 B_3 A_1$$

$$A_2 ::= b A_3 A_2 A_1$$

$$A_2 ::= a B_3 A_1$$

$$A_2 ::= a A_1$$

$$A_2 ::= b$$

$A_1 ::= A_2 A_3$ la convertimos en las siguientes:

```
A1 ::= bA3A2B3A1A3
A1 ::= bA3A2A1A3
A1 ::= aB3A1A3
A1 ::= aA1A3
A1 ::= bA3
```

3. Las dos producciones en B₃ dan lugar a otras 10, colocando en el lado derecho de las 5 producciones que tienen comienzan por A₁ lo siguiente:

Convertimos B₃::A₁A₃A₂ en las siguientes:

```
B3 ::= bA3A2B3A1A3A3A2
B3 ::= aB3A1A3A3A2
B3 ::= bA2A3A2
B3 ::= bA3A2A1A3A3A2
B3 ::= aA1A3A3A2
```

La gramática resultante es la siguiente:

```
GNF = ( { a, b } , { A1, A2, A3, B3 } , A, P' )
P' = {
A1 ::= bA3A2B3A1A3 | bA3A2A1A3 | aB3A1A3 | aA1A3 | bA3 ,
A2 ::= bA3A2B3A1 | bA3A2A1 | aB3A1 | aA1 | b ,
A3 ::= A3A2B3 | bA3A2 | aB3 | a ,
B3 ::= bA3A2B3A1A3A3A2B3 | bA3A2B3A1A3A3A2 | aB3A1A3A3A2B3 | bA3A2A1A3A3A2B3 |
bA3A2A1A3A3A2 | aA1A3A2B3 | aA1A3A3A2
}
```

Máquina de Turing

Consta de:

- Una **cinta infinita**. Contiene símbolos, pertenecientes a un alfabeto llamado alfabeto de entrada (Σ). Existe un símbolo especial llamado carácter de relleno o blanco.

Ejemplo:

```
... | A | B | C | b | ...
```

Nota: el blanco se denota por una 'b' minúscula.

- Una **cabeza móvil** de lectura/escritura que se puede desplazar hacia la izquierda o la derecha. Lee y escribe símbolos en la posición de la cinta sobre la que se sitúa. Dichos símbolos pertenecen a otro alfabeto llamado alfabeto de la cinta (Γ).
- Un **indicador de estados**. Los estados se representan como elementos de un conjunto finito $P = \{q_0, q_1, \dots, q_p\}$. Existe siempre un estado inicial, designado como q_0 . Existen uno o varios estados finales que pertenecen a un conjunto $F = \{q_p, \dots\}$. El número mínimo de estados de una máquina de Turing es de dos (uno inicial y otro final).

Lo anterior cumple que:

```
 $\Sigma \subset \Gamma$ 
b  $\notin \Sigma$ 
b  $\in \Gamma$ 
```

Acciones posibles

1. **Cambiar de estado** (pasar del estado actual a un nuevo estado).
2. **Escribir un símbolo** en una celda de la cinta en la cual se encuentra posicionada la cabeza de L/E (en la que se acaba de leer).
3. **Mover la cabeza** de L/E una posición hacia la derecha (D), hacia la izquierda (I) o quedarse inmóvil (P).

Las acciones vienen determinadas por una correspondencia entre dos conjuntos:

$$Q = \{q_0, q_1, \dots, q_n\}$$

$$\Gamma = \{A, B, C, \dots, b, \dots\}$$

$$M = \{I, D, P\}$$

$$Q \times \Gamma = \{(q_1, A), (q_2, B), \dots\}$$

$$Q \times \Gamma \times M = \{(q_0, A, I), (q_0, A, D), \dots, (q_1, A, P), \dots\}$$

$$Q \times \Gamma \rightarrow f \rightarrow Q \times \Gamma \times M$$

F: función de transición ó de transferencia

Ejemplo:

$$(q_0, B)$$

$$\dots | B | \dots$$

$$\quad \quad \quad \wedge$$

$$(q_n, A, D)$$

$$\dots | A | | \dots$$

$$\quad \quad \quad \wedge$$

Ejemplo:

$$Q = \{q_0, q_1, q_p\}$$

q_0 : estado inicial.
 q_p : estado final.

$$\Gamma = \{a, b\} \quad b \notin \Sigma$$

$$\Sigma = \{a\}$$

$$f(q_0, a) = (q_1, b, P)$$

$$f(q_0, b) = (q_p, b, P)$$

$$f(q_1, a) = (q_0, a, P)$$

$$f(q_1, b) = (q_0, b, D)$$

$Q \backslash \Gamma$	a	b
q_0	q_1, b, P	q_p, b, P
q_1	q_0, a, P	q_0, b, D
q_p		

Una máquina de Turing, puede ser definida por una **séptupla**:

$$M = (\Gamma, \Sigma, b, Q, q_0, F, f)$$

En la cual

- Γ: alfabeto de la máquina.
- Σ: alfabeto de entrada.
- b: carácter de relleno.
- Q: conjunto de todos los estados posibles que puede adoptar la máquina.
- q0: estado inicial. Debe pertenecer a Q.
- F: conjunto de todos los estados finales. Debe ser parte de Q.
- f: función de transición.

Ejemplo:
 $M = (\{1, 0, b\}, \{1\}, b, \{p, q, r, s\}, p, \{s\}, f)$

Q\Γ	1	0	b
p	q0D	p0I	rbD
q	q1D	q0D	p0I
r		r1D	sbP
s			

$\dots | b | b | \overset{\wedge_p}{1} | 1 | b | b | \dots \quad f(p, 1) = (q, 0, D)$
 $\dots | b | b | 0 | \overset{\wedge_q}{1} | b | b | \dots \quad f(q, 1) = (q, 1, D)$
 $\dots | b | b | 0 | 1 | \overset{\wedge_q}{b} | b | \dots \quad f(q, b) = (p, 0, I)$
 $\dots | b | b | 0 | \overset{\wedge_p}{1} | 0 | b | \dots \quad f(p, 1) = (q, 0, D)$
 $\dots | b | b | 0 | 0 | \overset{\wedge_q}{0} | b | \dots \quad f(q, 0) = (q, 0, D)$
 $\dots | b | b | 0 | 0 | 0 | \overset{\wedge_q}{b} | \dots \quad f(q, b) = (p, 0, I)$
 $\dots | b | b | 0 | 0 | 0 | 0 | \overset{\wedge_p}{\dots} \quad f(p, 0) = (p, 0, I)$

Tras unas cuantas iteraciones más, resulta lo siguiente:

$\dots | b | b | 1 | 1 | 1 | 1 | \overset{\wedge_p}{b} | \dots$

Y la máquina termina su ejecución.

Ejemplo:
 $M = (\{0, 1, b\}, \{0, 1\}, b, \{p, q, r\}, p, \{r\}, f)$

Q\Γ	0	1	b
p	p0D	q1D	r0P
q	q0D	p1D	r1P
r			

Esta máquina comprueba la paridad de una cadena.

Ejemplo:
 $M = (\{0, 1, *, A, B, b\}, \{0, 1, *\}, b, \{p, q, r, s, t\}, p, \{t\}, f)$

QΓ	0	1	*	A	B	B
p	q1D	rBD	p*I	pAI	pBI	sbD
q	rBD		q*D	qAD	qBD	pAI
r			r*D	rAD	pBD	pBI
S			s*D	S0D	s1D	tBP
t						

...|b|b|1|0|1|*|b|b|b|... $f(p, *) = (p, *, I)$
 $\wedge p$

...|b|b|1|0|1|*|b|... $f(p, 1) = (r, B, D)$
 $\wedge p$

...|b|1|0|B|*|b|... $f(r, *) = (r, *, D)$

...|b|1|0|B|*|b|... $f(r, b) = (p, B, I)$
 $\wedge r$

...|b|1|0|B|*|B|b|... $f(p, *) = (p, *, I)$
 $\wedge p$

Unas cuantas iteraciones después, llegamos a lo siguiente:

...|b|B|A|B|*|B|A|B|b
 $\wedge p$

Esta máquina copia trozos de la cinta en otros trozos, usando el asterisco como separador.

Ejemplo:
 $M = (\{ (,), *, |, |, B, M, b \}, \{ (,), |, | \}, b, \{ p, q, r, s, t \}, p, \{ t \}, f)$

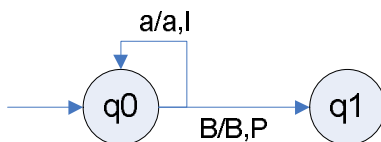
QΓ	()	*			b
p	p(D	q*I	p*D	p D	p I	
q	p*D		q*I	tMP		
r	s*I		r*I	tBP		
S	S*I		s*I	tMP		
t						

...|b|b| | (((|))) | | | b|b|...

Esta máquina comprueba si hay igual número de paréntesis de apertura que de cierres. Si termina con M el número es el mismo. En caso contrario, termina con B.

Podemos representar las transiciones en un grafo de la siguiente forma:

Q\Γ	a	b
q ₀	q ₀ a	q ₁ BP
q ₁		



Máquinas de Turing restringidas

Una máquina de Turing puede ser restringida en distintos aspectos, originando máquinas con características tales como:

- Únicamente usa el alfabeto binario.
- Con la cinta de entrada limitada en un sentido. La transformación de una máquina sin restricciones a otra con esta restricción es sencilla. Solo hemos de definir un nuevo alfabeto.

Ejemplo:

...|B|a₋₃|a₋₂|a₋₁|a₀|a₁|a₂|a₃|B|...

Se convertiría en la siguiente cinta:

a₀, \$ |a₁, a₋₁|a₂, a₋₂|a₃, a₋₃|B, B|...

- Que no pueda cambiar de estado y sobrescribir simultáneamente.
- Que no pueda mover la cabeza y sobrescribir simultáneamente.
- Que no pueda mover la cabeza y cambiar de estado simultáneamente.
- Que solo realice una operación en cada paso o transición.

Máquina de Turing universal

Se compone de una cinta que contiene, la cinta de la máquina de Turing a simular y la función de transferencia de la máquina a simular.

Es una máquina que **permite adoptar el comportamiento** de cualquier máquina de Turing.

Composición de máquinas de Turing

Componiendo máquinas de Turing, podemos crear máquinas con **comportamientos complejos**. Designamos las máquinas mediante letras mayúsculas.

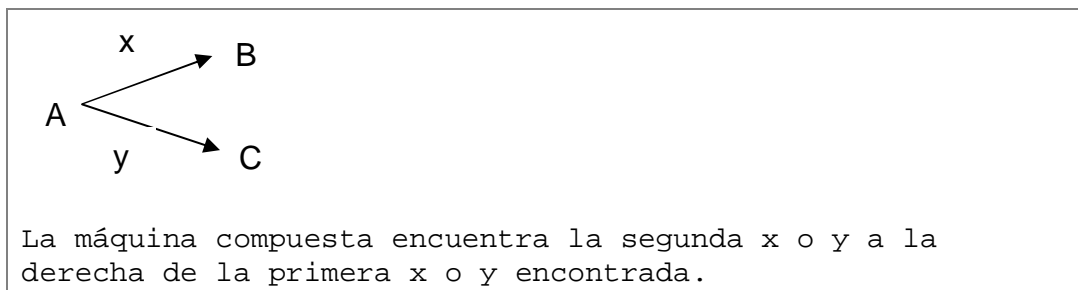
Ejemplo:

Dadas 3 máquinas, A, B y C...

A: mueve la cabeza una celda a la derecha.

B: busca un símbolo x hacia la derecha.

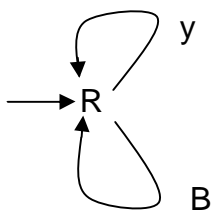
C: busca un símbolo y hacia la derecha.



Llamaremos **R** a la máquina que solo mueve la cabeza una celda a la derecha.
 Llamaremos **L** a la máquina que solo mueve la cabeza una celda a la izquierda.
 Llamaremos **W** a la máquina que escribe en la celda actual o sobre la que se encuentra posicionada la cabeza de L/E.

Máquina que busca un símbolo x

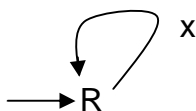
La siguiente máquina busca un símbolo x a la derecha:



Vamos a llamar a esta máquina R_x (o L_x en caso de buscar a la izquierda).

Máquina que desplaza la cabeza hasta encontrar un símbolo distinto a x

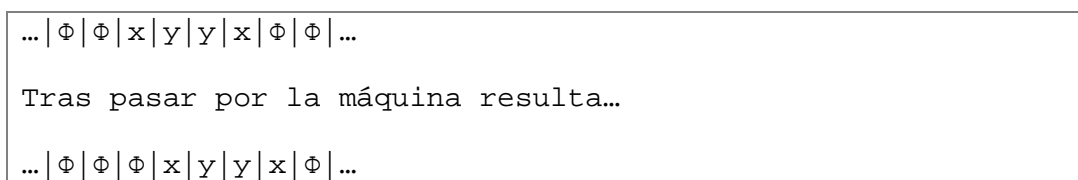
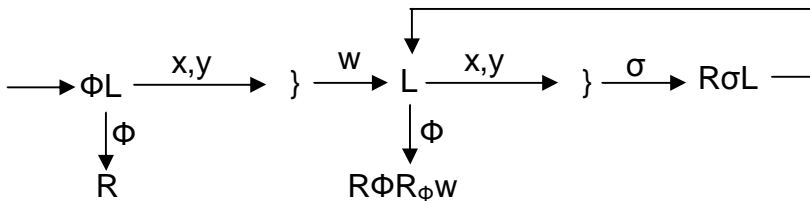
La siguiente máquina desplaza la cabeza hacia la derecha:



Vamos a llamar a esta máquina R_{-x} (L_{-x} en caso de desplazarse a la izquierda).

Máquina que mueve una cadena

Para desplazar la cadena a la derecha:



La vamos a llamar S_R (*shift right*), o S_L en caso de desplazar la cadena a la izquierda.

1. Se inserta el símbolo del estado antes del símbolo al cual apunta la cabeza de L/E.
2. Si la cabeza apunta a un carácter de relleno (Φ), dicho símbolo queda incluido en la descripción instantánea.
3. Eliminamos todas las celdas que contengan caracteres de relleno (Φ), situadas a la izquierda o derecha del contenido no de relleno de la cinta, y que no estén incluidas en el apartado anterior. Sustituimos los caracteres eliminados justo antes por paréntesis.

Ejemplo:

...| Φ | Φ | Φ |0|1|0| Φ | Φ | Φ |... se convierte en (101 Φ p Φ)

^
p

En resumen:

1. Movimiento de la cabeza a la derecha:

$f(p, a) = (q, b, D)$
 $(pa... \rightarrow (bq...$
 $(pa... \rightarrow (q... \text{ si } b = \Phi$
 $...pa) \rightarrow ...bq\Phi)$
 $...pa... \rightarrow ...qcb...$

De lo que obtenemos:

$pa \rightarrow bq$
 $(\Phi \rightarrow ($
 $pa \rightarrow bq\Phi)$

2. Movimiento de la cabeza a la izquierda:

$F(p, a) = (q, b, I)$
 $cpa \rightarrow (qIb...$
 $...cpa) \rightarrow ...qcb)$
 $...cpa) \rightarrow ...qc) \text{ si } b = \Phi$
 $...cpa... \rightarrow ...qcb...$

De lo que obtenemos:

$(pa \rightarrow (q\Phi b$
 $cpa \rightarrow qcb$
 $a\Phi) \rightarrow a) \text{ para todo } a \in \Sigma$

3. No hay movimiento de la cabeza:

$F(p, a) = (q, b, P)$
 $(pa... \rightarrow (qb...$
 $...pa) \rightarrow ...qb)$
 $...pa... \rightarrow ...qb...$

De lo que obtenemos:

$pa \rightarrow qb$

Lenguajes aceptados por una máquina de Turing

Se dice que una máquina de Turing acepta o reconoce un lenguaje si todas las palabras del lenguaje son borradas por la dicha máquina.

Ejemplo:

$$T = (\Sigma \cup \{ \vdash, \dashv, \Phi \}, \Sigma \cup \{ \vdash, \dashv \}, \Phi, Q, q_0, F, f)$$

$$w \in \Sigma^*$$

$$\dots | \Phi | \Phi | \vdash | \dots | \dashv | \Phi | \Phi | \dots$$

\wedge_{q_0}

Descripción instantánea inicial (q_0, \vdash, w, \dashv)

$$\dots | \Phi | \Phi | \Phi | \dots | \Phi | \Phi | \Phi | \dots$$

\wedge_{q_t}

Descripción instantánea final (q_f, Φ)

$$G = (\Sigma_T, \Sigma_N, S, P)$$

$$\Sigma = \Sigma_T \cup \Sigma_N = \{ Q \cup \{ S, (,), \vdash, \dashv, \Phi \} \} \quad S \notin Q$$

$$P = \{ S ::= (q, \Phi), bq ::= pa, (::= (\Phi, bq\Phi) ::= pa, (q\Phi b ::= (pa, qcb ::= cpa, a) ::= a\Phi), qb ::= pa, (q_0 \vdash a ::= a, a \dashv) ::= a \}$$

Para todo $a \in \Sigma$ y para todo $c \in \Sigma \cup \{ \vdash, \dashv \}$

Ejemplo:

$$T = (\{0, 1, \vdash, \dashv, \Phi\}, \{0, 1, \vdash, \dashv\}, \Phi, \{p, q, r, s, t, u, v, z\}, p, \{v\}, f)$$

f\T	0	1	\vdash	\dashv
p	p0I	p1I	qΦD	
q	p⊢D	s⊢D		vΦP
r	r0D	p1D		tΦI
s	s0D	s1D		uΦI
T	p⊢I	zΦP	vΦP	
u	zΦP	p⊢I	vΦP	
v				
z				

Palíndromo $w.w^{-1} \in L(G)$
 $w \in \{0, 1\}^+$ w no puede ser la palabra vacía

Tenemos que obtener la gramática de esta máquina de Turing.

$$G = (\{0, 1\}, \{p, q, r, s, t, u, v, z, S, (,), \vdash, \dashv, \Phi\}, S, P)$$

$$P = \{ S ::= (v\Phi), (p\Phi 0 ::= (p0, p00 ::= 0p0, p10 ::= 1p0, p\vdash 0 ::= \vdash p0, p\vdash 0 ::= \dashv p0, 0) ::= 0\Phi), 1) ::= 1\Phi), \vdash) ::= \vdash \Phi), \dashv) ::= \dashv \Phi) \}$$

Convertimos todas las transiciones en reglas de producción. Cuando tengamos todas, podremos derivar el axioma para obtener una palabra del lenguaje, generando palíndromos.

Máquinas de Turing no deterministas

La correspondencia entre Q (los estados posibles de la máquina con Γ (los elementos de la cinta) ya no es uniforme, es decir, a la función de transición le pueden corresponder varias tripletas.

Ejemplo:
 $T = (\{0, 1, \vdash, \dashv, \Phi\}, \{0, 1, \vdash, \dashv\}, \Phi, \{p, q, r, s\}, p, \{s\}, f)$

$Q\Gamma$	0	1	\vdash	\dashv	Φ
p	q0P	q Φ D	q Φ D	r Φ P	
q	p0P	q Φ D r Φ D		s Φ P	
r	p Φ D			p Φ P	
s					

Una máquina de Turing no determinista reconoce un lenguaje si existe al menos un camino que permita pasar de la descripción instantánea inicial ($p \vdash _$) a una final ($s\Phi$) en donde sef.

Probamos el ejemplo anterior:

- 1) $w=0 \rightarrow 0 \notin L$
- 2) $w=1 \rightarrow 1 \in L$
- 3) $w=101 \rightarrow 101 \in L$
- 4) $w=1001 \rightarrow 1001 \notin L$

$L(T) =$ Todas las palabras que comiencen y terminen por 1 en las que si hay dos ceros, deberán estar 'encerrados' por dos unos.

Dada una gramática G_0 , vamos a tratar de construir una máquina que acepte o reconozca el lenguaje.

$P = \{S ::= AB1 \mid ASB, A0 ::= 00, A1 ::= 11, B ::= 00 \mid \lambda\}$

$S \rightarrow AB1 \rightarrow A1 \rightarrow 11$
 $S \rightarrow AB1 \rightarrow A0001 \rightarrow 0001$
 $S \rightarrow ASB \rightarrow AAB1B \rightarrow AA1 \rightarrow A11 \rightarrow 111$

$L(G_0) = \{11, 0001, 111, \dots\}$

$S \rightarrow ASB \rightarrow AAAB1B \rightarrow AA001 \rightarrow A0001 \rightarrow 00001$

Como $B ::= \lambda$, podríamos tener palabras nuevas usando esa producción, agrandando el árbol de derivación.

Dada una máquina de Turing no determinista que reconoce el lenguaje L , se puede construir otra determinista equivalente.

Máquina de Turing de k cintas

Es una máquina corriente que trabaja con varias cintas a la vez. Cada cinta tiene una cabeza de lectura escritura.

$$F(p, a_1, a_2, \dots, a_k) = (q, b, j, M)$$

Ejemplo:
 $J=2, M=I$

Toda máquina de Turing de varias cintas se puede transformar en otra equivalente con una sola cinta.

```

... | x | y | x | x | ...
      ^
... |  $\Phi$  | x | y | y | y | ...
      ^
... |  $\Phi$  |  $\Phi$  |  $\Phi$  | x | x | ...
      ^
    
```

La transformamos en lo siguiente:

x	y	x	x	
		1		
	x	y	y	y
				1
			x	X
1				

Hemos creado una tabla con el doble de filas que de cintas. En las filas impares hemos colocado el contenido de las cintas. En las pares, hemos marcado con un uno la presencia de la cabeza de L/E.

Ahora vamos a agrupar las columnas, dando lugar a símbolos compuestos. Para facilitar las cosas, vamos a introducir un símbolo al principio de la cinta, por ejemplo #.

```

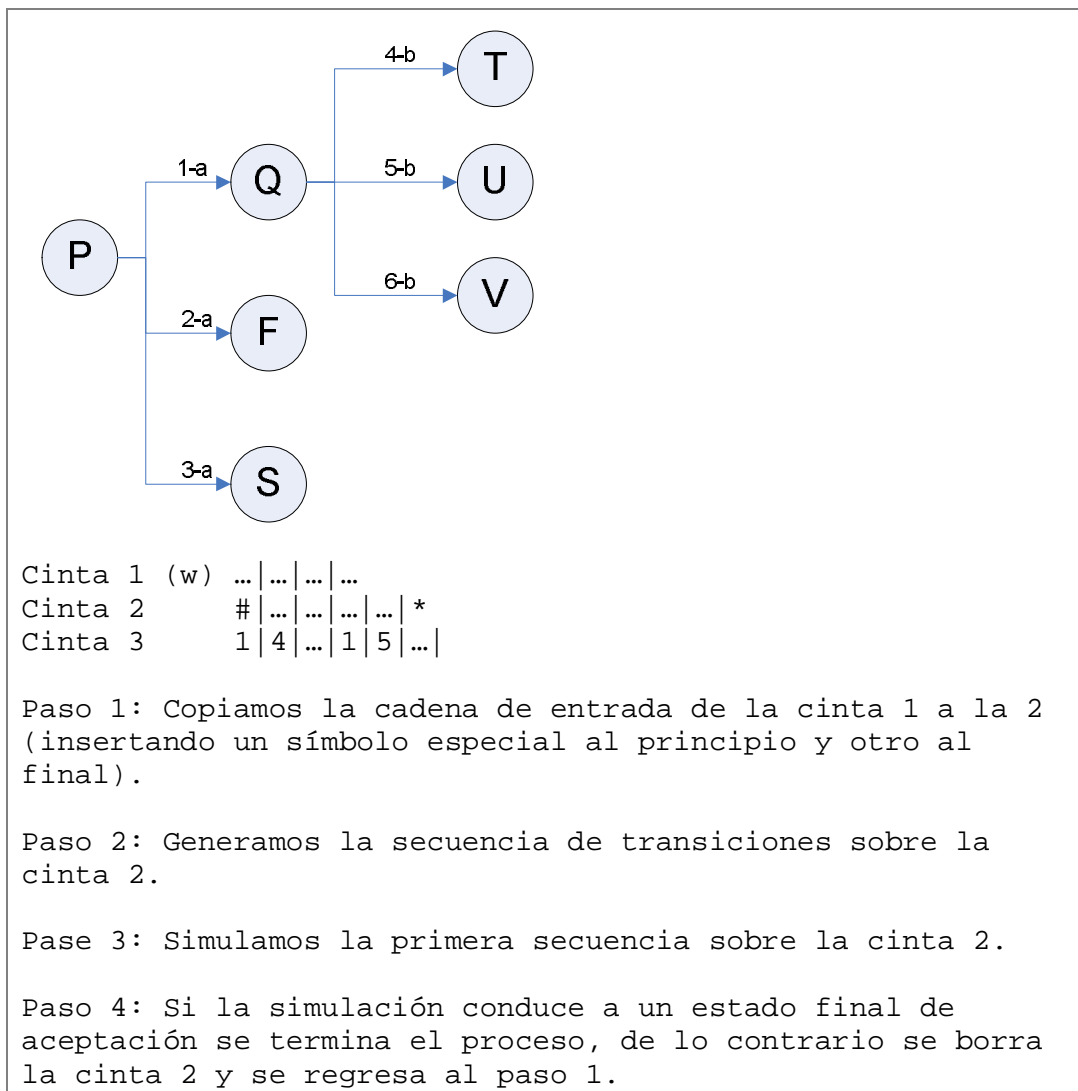
(x $\Phi\Phi\Phi\Phi$ 1)
(y $\Phi$ x $\Phi\Phi\Phi$ )
(x1y $\Phi\Phi\Phi$ )
(x $\Phi$ y $\Phi$ x $\Phi$ )
( $\Phi\Phi$ y1x $\Phi$ )
    
```

Ahora los símbolos de la nueva máquina son:

$$\Gamma = \{ \#, x\Phi\Phi\Phi\Phi 1, y\Phi x\Phi\Phi\Phi, x1y\Phi\Phi\Phi, x\Phi y\Phi x\Phi, \Phi\Phi y1x\Phi \}$$

Tener una máquina de k cintas no aumenta la potencia, pero permite ver más claro el funcionamiento de la misma en comportamientos complejos.

Ejemplo: conversión de no determinista a determinista (con tres cintas).



Un **problema de decisión** siempre tiene una solución, que pertenece al conjunto {S,N} (Sí,No).

Lenguaje Decidible (para una máquina de Turing): cuando la máquina termina normalmente y a su terminación queda una indicación correspondiente al conjunto {S,N}.

$$F \subset Q \quad F = \{a, r\}$$

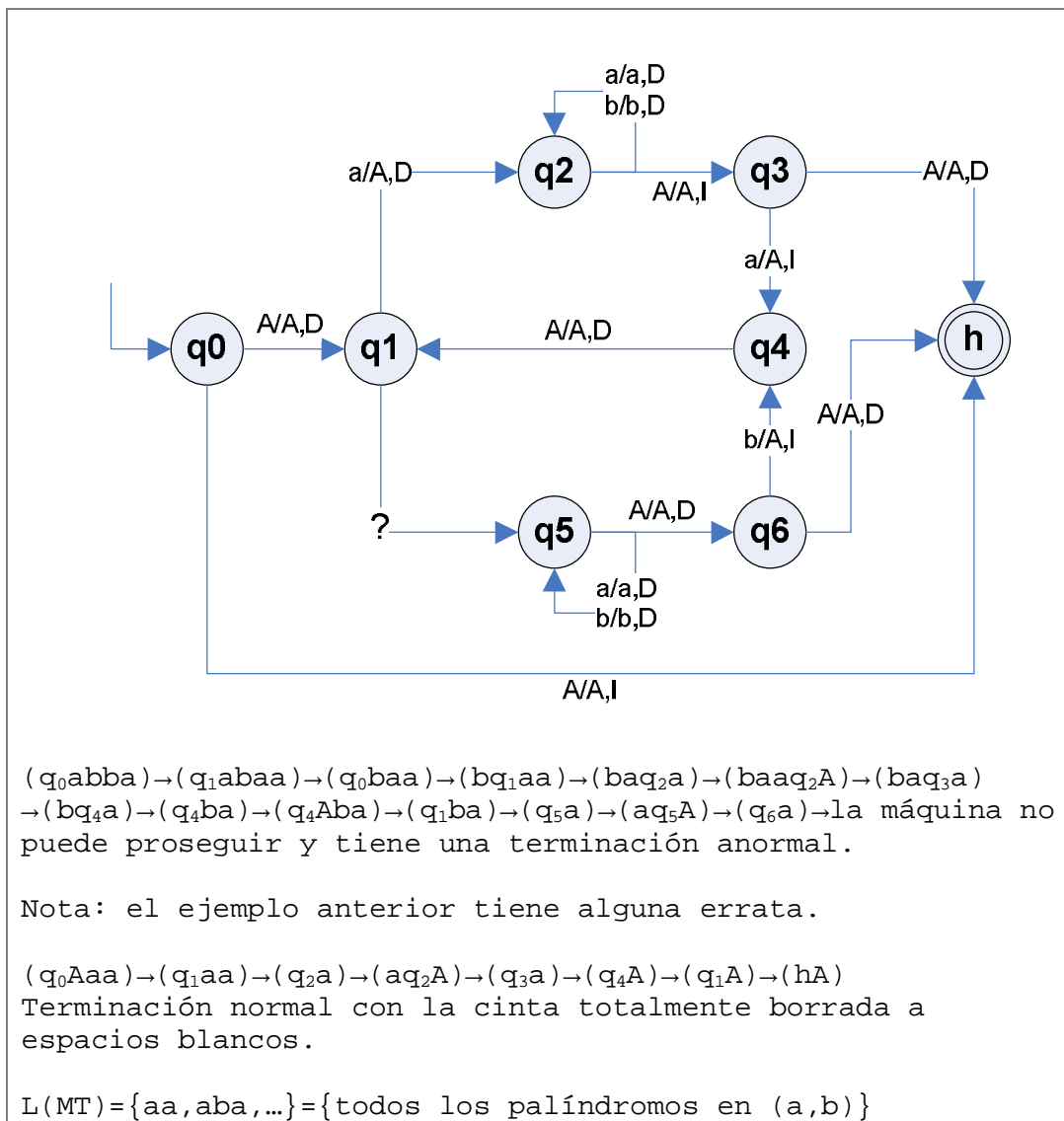
a = estado de aceptación (SÍ)

r = estado de rechazo (NO)

$$F \subset Q \quad F = \{n\} \quad \begin{aligned} \text{Descripción instantánea final} &= \dots|S|\dots \text{ estado final de aceptación} \\ \text{Descripción instantánea final} &= \dots|N|\dots \text{ estado final de rechazo} \end{aligned}$$

Cuando se arranca con una descripción instantánea inicial que contiene una palabra w y resulta un estado de aceptación/rechazo según si la palabra es decidible.

Ejemplo:
 A: carácter de relleno



Complejidad

La **complejidad** depende de los recursos necesarios que se empleen en la resolución de un problema. Los recursos se descomponen en espacio y tiempo. En base a éstos obtenemos dos tipos de complejidad:

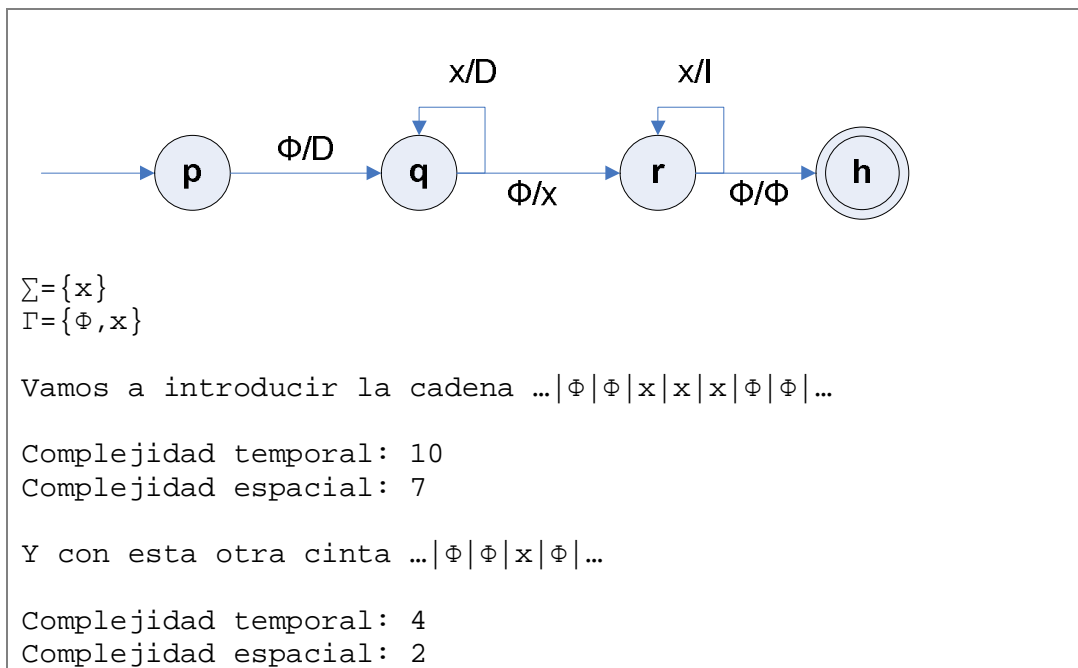
- **Complejidad temporal:** cantidad de tiempo empleado en la resolución de un problema.
- **Complejidad espacial:** cantidad de espacio empleado en la resolución de un problema.

Complejidad de los procesos realizados por las máquinas de Turing

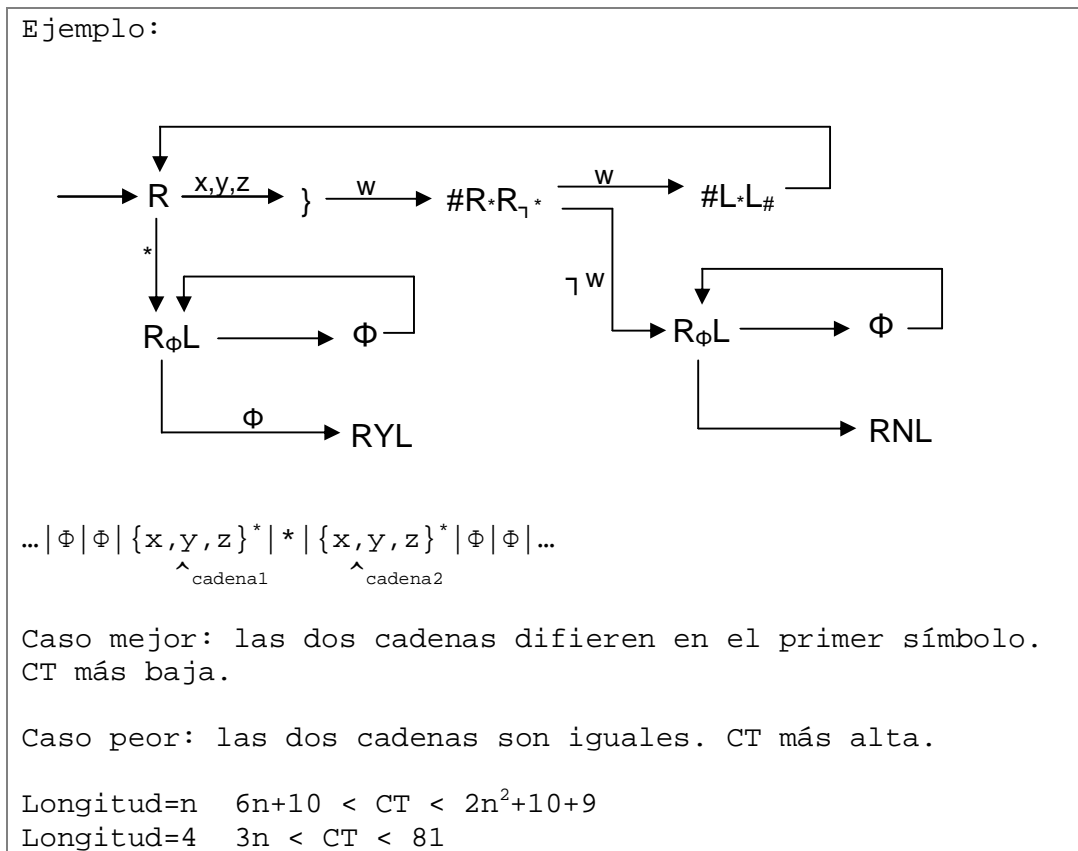
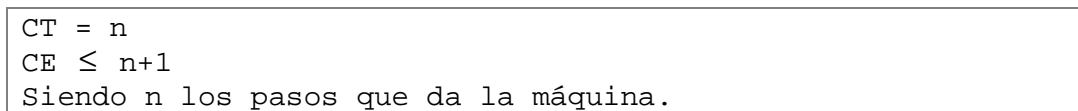
Definimos **complejidad espacial** de un proceso realizado por una máquina de Turing como el número de casillas por las que pasa la cabeza de L/E a lo largo de su proceso.

Definimos **complejidad temporal** de un proceso realizado por una máquina de Turing como el número de pasos o transiciones que efectúa la máquina de Turing a lo largo de su proceso.

Ejemplo:



Las complejidades se calculan de la siguiente manera:



Probabilidad

Nos interesa la complejidad temporal media.

$$CT_{media} = \sum_{i=1}^n p_i * c_i$$

Siendo c_i la longitud del cálculo (número de transiciones) y p_i la probabilidad de ocurrencia de ese caso.

Para una longitud de 4:

Posición de la 1ª discrepancia	Ci	Pi	Ci*Pi
1	34	2/3	(34*2)/3
2	46	2/9	(46*2)/9
3	50	2/27	(50*2)/27
4	70	2/81	(70*2)/81
NO	81	1/81	(81*1)/81
			39

Necesitamos una medida para las complejidades de los procesos.

Tasas de crecimiento

Ω : conjunto de funciones de N sobre N siendo $N=\{0,1,2,\dots\}$, es decir, el conjunto de números naturales. Dada la función f perteneciente a Ω , vamos a definir $O(f)$ como el conjunto g de todas las funciones de Ω para las cuales existe una constante $c>0$ y un número n_0 de N tales que:

$$g(n) \leq c * f(n) \quad \forall n \geq n_0 \quad (g \text{ está limitada superiormente})$$

Ejemplo:

$$31n^2 + 17n + 23 = O(n^2) \quad \text{es decir, } \forall n \geq 1 \quad 31n^2 + 17n + 23 \leq 71n^2$$

$f = O(g)$
 $g = O(f)$

f y g son equivalentes.

Tenemos una relación de equivalencia sobre un conjunto de funciones Ω , estableciendo una serie de clases de equivalencia o **tasas de crecimiento** (θ).

Ejemplo: Polinomio de grado $d = O(n^d)$

$$a_0 + a_1n + a_2n^2 + \dots + a_d n^d = O(n^d)$$

$$\sum_{i=0}^d a_i n^i = \lim_{n \rightarrow \infty} \frac{1}{|a_d + a_{d-1} \frac{1}{n} + \dots + a_0 \frac{1}{n^d}|} = \frac{1}{|a_d|}$$

$$\frac{1}{2|a_d|} \leq \frac{n^d}{|\sum_{i=0}^d a_i n^i|} \leq \frac{3}{2|a_d|}$$

Entonces...

$$\sum_{i=0}^d a_i n^i \leq 2n^d |a_d|$$

$$n^d \leq \frac{3}{2|a_d|} \sum_{i=0}^d a_i n^i$$

$$a_0 + a_1 n + a_2 n^2 + \dots + a_d n^d = O(n^d)$$

$$n^d = O(a_0 + a_1 n + a_2 n^2 + \dots + a_d n^d)$$

Todo polinomio de grado d tiene la misma tasa de crecimiento que n^d .

Existen tasas de crecimiento:

- Polinómicas
- Logarítmicas
- Factoriales

Un algoritmo que resuelve un problema, tiene una complejidad.

Complejidad temporal de los problemas de reconocimiento de lenguajes

La máquina de Turing efectúa todo el cálculo de la función f en tiempo polinómico (algo razonable) con una complejidad polinómica.

Dada una máquina de Turing M , decimos que la máquina calcula una función $f: \Sigma_1^* \rightarrow \Sigma^{2*}$ en tiempo polinómico si existe un polinomio $p(x)$ tal que para cada palabra w perteneciente a Σ_1^* , la máquina calcula la $f(w)$ en no más de $p(|w|)$ pasos.

Dada una máquina de Turing M , se dice que acepta el lenguaje L en tiempo polinómico. Si existe un polinomio $p(n)$ tal que el número de pasos necesarios para aceptar cualquier palabra w perteneciente a L , no es mayor que el valor del polinomio $p(|n|)$.

Podemos hablar de una **clase de lenguaje P**, que son los lenguajes que una máquina de Turing puede aceptar en **tiempo polinómico**.

Ejemplo:

M : máquina que acepta cualquier cadena w de L con una complejidad temporal proporcional al polinomio $|w|^2$.
 M' : máquina que va a aceptar el mismo lenguaje con una complejidad temporal proporcional al polinomio $2^{|w|}$.

n : longitud de la cadena.
 Cada paso 1 microsegundo.

Para $n=10$
 M : 10^{-4} s
 M' : 10^{-4} s

Para $n=20$
 M : $4 \cdot 10^{-4}$ s
 M' : 1'05 s

La clase P agrupa a todos los lenguajes que las máquinas de Turing aceptan en tiempo polinómico o en un tiempo $O(n^d)$.

Decisión de un lenguaje

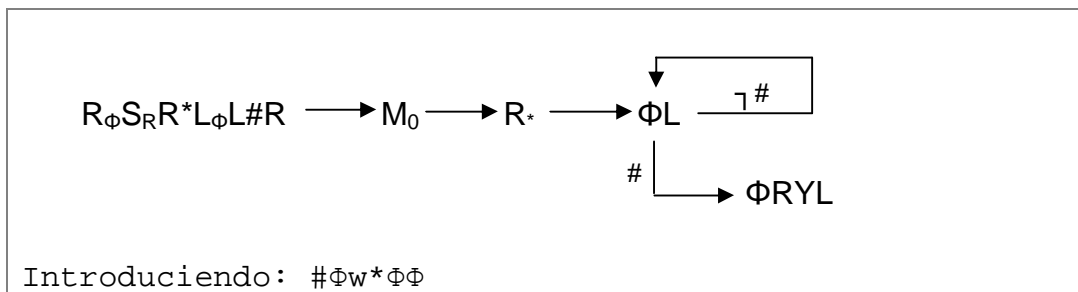
M decide un lenguaje L si dada cualquier palabra w, la máquina arrancando con w siempre termina su proceso, normalmente indicando si la palabra pertenece o no a un lenguaje.

M decide un lenguaje L en tiempo polinómico si existe un polinomio $p(n)$ tal que la máquina no da más de $p(|w|)$ pasos para conocer si w pertenece a L o no.

Si una máquina de Turing M acepta un lenguaje L en tiempo polinómico, entonces existe otra máquina de Turing M' que acepta el mismo lenguaje L en tiempo polinómico, pero indicando su aceptación parándose con la descripción instantánea final (SΦY) S=estado final.

Y: la cadena pertenece al lenguaje.
 N: la cadena no pertenece al lenguaje.

La máquina M' sería así:



M_0 ejecuta las acciones de la máquina y da como máximo 4 pasos adicionales cuando encuentra los símbolos * o # completa su tarea en no más de $4p(|w|)$. La parte con datos de la cinta de entrada no es mayor que:

$$K = |w| + 3 + 4p(|w|)$$

Esto quiere decir que se requiere un máximo de $3K + 4$ pasos. Para borrar la cinta, escribir una Y y retroceder la cabeza de L/E al extremo izquierdo de la cinta.

Una máquina de Turing puede decidir si una cadena se encuentra o no en un lenguaje. Los lenguajes que no cumplen esta condición serán lenguajes aceptables. Los lenguajes **independientes del contexto** son todos **decidibles**. Los lenguajes **estructurados por frases** son todos **aceptables**.

Lenguajes decidibles

La **clase NP**: una máquina de Turing no determinista (MTND) acepta un lenguaje L en tiempo polinómico si existe un polinomio $p(x)$ tal que para cualquier palabra w perteneciente a L, la MTND acepta w realizando no más de $p(|w|)$ transiciones o pasos. La clase NP es la clase de lenguajes que la MTND acepta en tiempo polinómico. Está claro que $P \subseteq NP$, pues la MTD se puede convertir a MTND, pero ¿se cumple que $P = NP$? Es un problema que todavía está por resolver.

Ejemplo: Problema de decisión del viajante.

Se tiene un conjunto V de n ciudades. Se conocen las distancias $d(V_i, V_j)$ entre las ciudades V_i, V_j . Se establece una distancia del viaje total d .

¿Existe una forma de viajar entre las ciudades de manera que se visite cada ciudad y que la ruta finalice en la ciudad de origen, sin exceder la distancia máxima?

Vamos a comprobar todas las rutas posibles:

$$\sum_{1 \leq i \leq n} d(v_{p_i}, v_{p_{i+1}}) + d(v_{p_n}, v_{p_1}) \leq d$$

El algoritmo tiene complejidad factorial. Podemos crear una máquina de Turing no determinista que acepte el lenguaje L_v . Esta MTND genera una ruta entre las ciudades de forma no determinista, evalúa la ruta generada, si la ruta es corta, la MTND se detiene, si no entra en un bucle infinito.

L_v pertenece a NP y no puede afirmarse que sea decidido en tiempo polinómico.