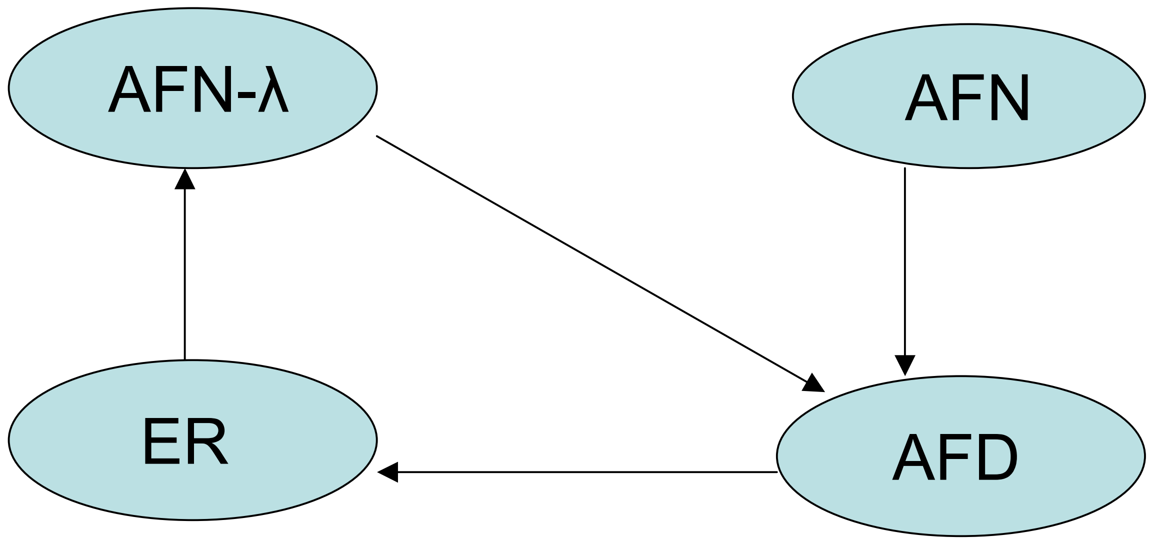
Teoría de la Computación y Lenguajes Formales

Gramáticas Generativas
Gramáticas Regulares

Prof. Hilda Y. Contreras
Departamento de Computación
hyelitza@ula.ve
hildac.teoriadelacomputacion@gmail.com

UNIVERSIDAD
DE LOS ANDES

Lenguaje Regular



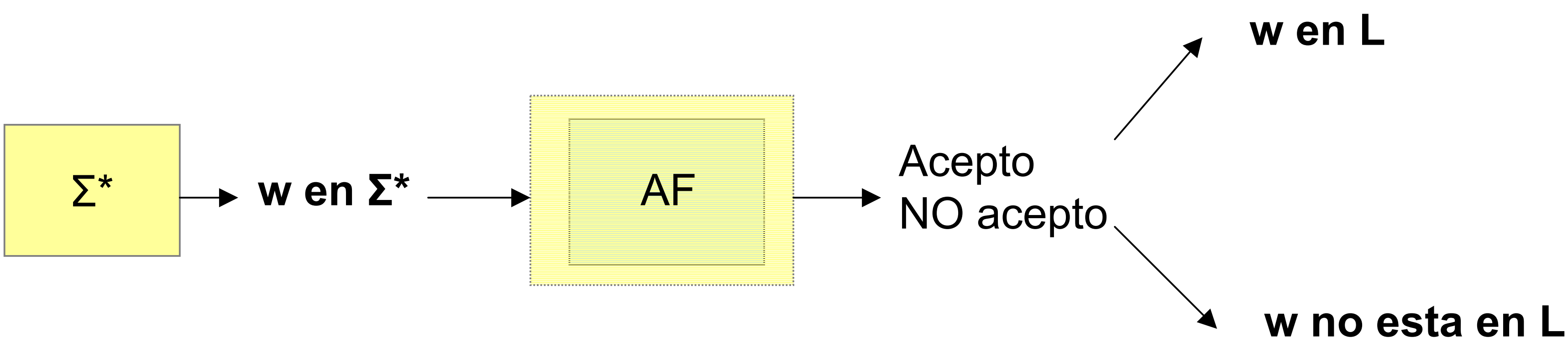
Autómata Finito	**Expresión Regular**
Reconocer, Verificar	Describir, Expresar

Generador de Lenguaje

- ¿Para qué generar un lenguaje?
- Generar un lenguaje L significa enumerar todas y cada una de las cadenas que lo forman. L es subconjunto de Σ^*
- ¿Un AF (ER) no me sirve para generar un lenguaje? Un AF reconoce, pero no puede generar?, no puedo adaptar su funcionamiento para que genere?
- ¿Si el lenguaje L es finito o es infinito?

Cómo un AF generaría a L

Dado el AF A, L(A) = L



Para todo w en Σ* debo probar si el AF acepta o no.
Formo L a partir de los w que acepta el AF.

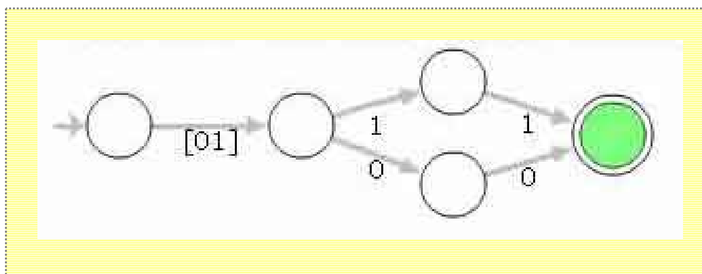
- ¿Cómo generar w en Σ*?. ¿Existe un algoritmo para generar las cadenas de Σ*? **($\Sigma^0 = \{\lambda\}$, $\Sigma^i = \Sigma\Sigma^{i-1}$, $i > 0$)**
- ¿El procedimiento de probar todo w en el AF termina (si el lenguaje es finito o infinito)?

Cómo un AF generaría a L

Dada la ER E = (0 + 1)(00 + 11) con Σ = {0,1}

Σ* = {λ,0,1,00,01,10,11,000,001,010,011,100,101,110,111,…}

w en Σ* →



→ Acepto → **L = {000,011,…}**

NO acepto → **{λ,0,1,00,01,…}**

Para todo w en Σ* debo probar si el AF acepta o no.
Formo L a partir de los w que acepta el AF.

- Puedo generar w en Σ*
- Pero no sé cuando se ha completado el lenguaje (termina de probar todas las cadenas de Σ*)
- **Por tanto no me sirve!**

Gramática

- Lingüística: es el estudio científico del lenguaje y trata de explicar cómo funcionan las lenguas.
- Gramática: Conjunto de reglas que especifican ("generan") las oraciones que están bien ("oraciones posibles") y cuales no y que propiedades estructurales poseen.

Noam Chomsky

- Avram Noam Chomsky (Estados Unidos) lingüista, filósofo, activista, autor y analista político. Profesor emérito de Lingüística en el MIT.
- Lingüística como una teoría de la adquisición individual del lenguaje y una explicación de las estructuras y principios más profundos del lenguaje. Creador de la jerarquía de Chomsky, (clasificación de lenguajes formales) de gran importancia en Teoría de la Computación.

Gramática Generativa

- En 1957 aparece la denominada “Gramática Generativa” tras la publicación de la obra de Noam Chomsky “Estructuras Sintácticas”.

- En este libro Chomsky expone que una gramática de constituyentes inmediatos **no** es totalmente válida para explicar el mecanismo mediante el cual los hablantes de una lengua son capaces de **producir** y **entender** oraciones.

Gramática Generativa

- **Propiedad de recursividad**:

  Una gramática generativa debe ser capaz de *generar* una infinita cantidad de construcciones sintácticas (palabras de un Lenguaje) a partir de un número limitado de reglas y unidades (modelo finito). p.e. El lenguaje humano es el único sistema de comunicación natural con tal propiedad.

Gramáticas Generativas

Una gramática generativa es una cuádrupla

(V, T, P, S) en la que:

- **V** es un conjunto de variables o símbolos no terminales. Sus elementos se suelen representar con letras mayúsculas.
- **T** es el alfabeto, conjunto de símbolos terminales. Sus elementos se suelen representar con letras minúsculas.
- **P** es un conjunto de pares (α, β), llamados reglas de producción, donde α, β en $(V \cup T)^*$ y α contiene, al menos un símbolo de V. El par (α, β) se suele representar como $\alpha \rightarrow \beta$
- **S** es un elemento de V, llamado símbolo de partida o inicial.

Lenguaje Regular y GR

- Jerarquía de Chomsky (Lenguaje Regular - Tipo 3)

Tipo	Lenguaje	Máquina	Gramática
0	Recursivamente enumerable	Máquina de Turing	Sin restricciones
1	Dependiente del Contexto	Autómata linealmente acotado	*Gramática dependiente del contexto* $\alpha A\beta \rightarrow \alpha\gamma\beta$
2	Independiente del Contexto	Autómata de Pila	*Gramática libre de contexto* $A \rightarrow \gamma$
3	**Lenguaje Regular**	**Autómata finito**	***Gramática Regular*** $\boldsymbol{A \rightarrow aB}$ $\boldsymbol{A \rightarrow a}$

Paso de Derivación

Dada la Gramática G = (V, T, P, S) y dos palabras α, β en $(V \cup T)^*$ β es derivable a partir de α en un paso ($\alpha \Rightarrow \beta$) si y solo si existen palabras δ_1, δ_2 en $(V \cup T)^*$ y una producción $\gamma \rightarrow \eta$ tales que:

$\alpha = \delta_1\gamma\delta_2$ y $\beta = \delta_1 \eta \delta_2$

la derivación sería:

$\delta_1\gamma\delta_2 \Rightarrow \delta_1 \eta \delta_2$

Secuencia de Derivación

Sucesión [1 a ∞] de pasos de derivación

β es derivable de α ($\alpha \Rightarrow^* \beta$) si y solo si existe una sucesión de palabras

$\gamma_1, \ldots, \gamma_n$ ($n \geq 1$) tales que:

$$\alpha \Rightarrow \gamma_1 \Rightarrow \gamma_2 \Rightarrow \ldots \Rightarrow \gamma_n \Rightarrow \beta$$

Lenguaje Generado

Se define como lenguaje generado por una gramática G = (V, T, P, S) al conjunto de cadenas formadas por símbolos terminales en V y que son derivables a partir del símbolo de partida S.

Es decir,

$L(G) = \{ w \mid w \text{ en } T^* \text{ y } S \Rightarrow^* w \}$

Gramática Regular

Una Gramática Regular es una Gramática Generativa G = (V,T,P,S)

Donde el conjunto P de producciones tiene la forma:

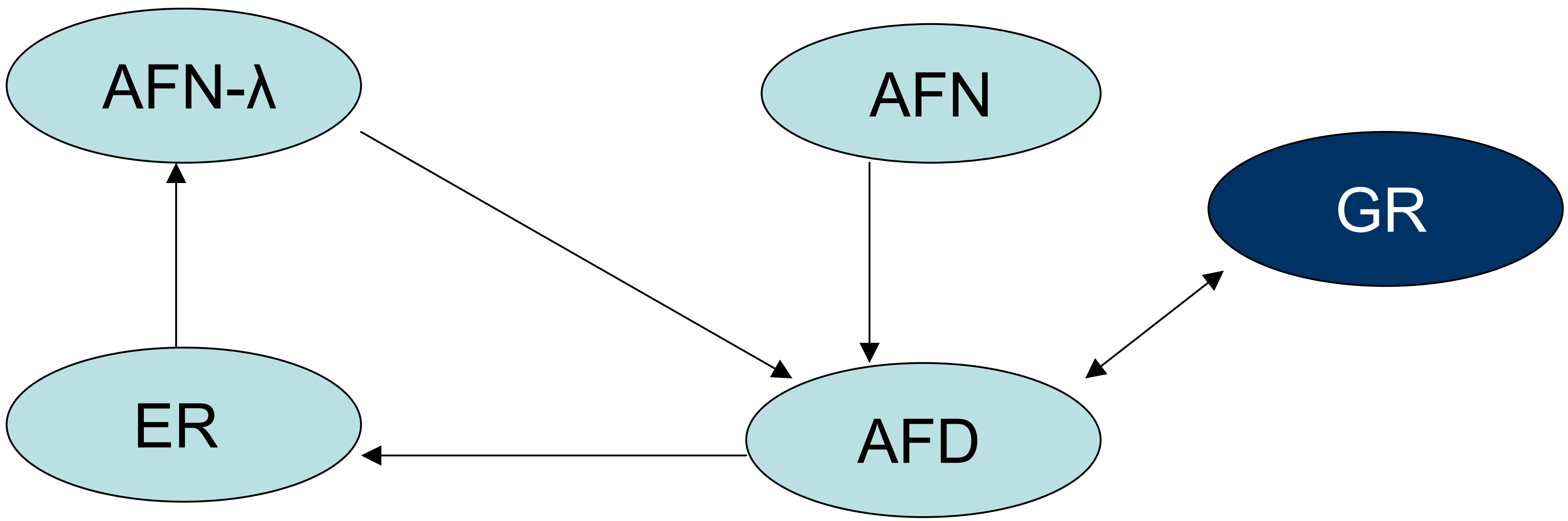
A -> aB ,

A -> a

donde A,B en V y a en T

El lenguaje generado L(G) es regular

Relación GR – AF



Autómata Finito	Gramática Regular	Expresión Regular
Reconocer, Verificar	Generar	Describir, Expresar

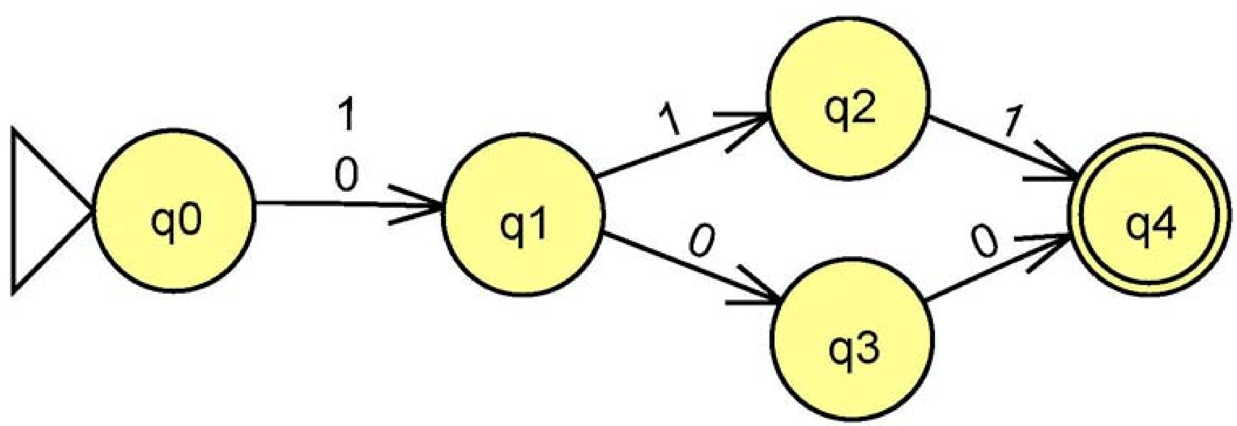
Lenguaje Regular ER-AF-GR

- Por ejemplo: $\Sigma = \{0,1\}$

  Lenguaje del alfabeto binario que comienzan con 0 o 1, seguido de dos ceros o dos unos.

ER: $(0 + 1).(00 + 11)$

AF:



Equivalencia (AF → GR)

<u>Teorema</u>: Si L es un lenguaje aceptado por un autómata finito M, entonces existe una gramática regular G tal que L = L(M) = L(G).

Suponemos que $M = (Q, \Sigma, \delta, q_0, F)$ es un AF sin transiciones nulas.

Podemos obtener la gramática $G = (Q, \Sigma, P, q_0)$ a partir del diagrama de transición del AF con el siguiente método:

1. Si tenemos el arco etiqueta **a** del estado **q** a **p** entonces añadimos a P la regla $q \rightarrow ap$
2. Si q_f en F añadimos la regla $q_f \rightarrow \lambda$

AF-GR

$L = (0 + 1).(00 + 11)$

$G = (\{q_0,q_1,q_2,q_3,q_4\},\{0,1\},P,q_0)$

P: $\mathbf{q}_0 \dashrightarrow \mathbf{0q}_1$

$\mathbf{q}_0 \dashrightarrow \mathbf{1q}_1$

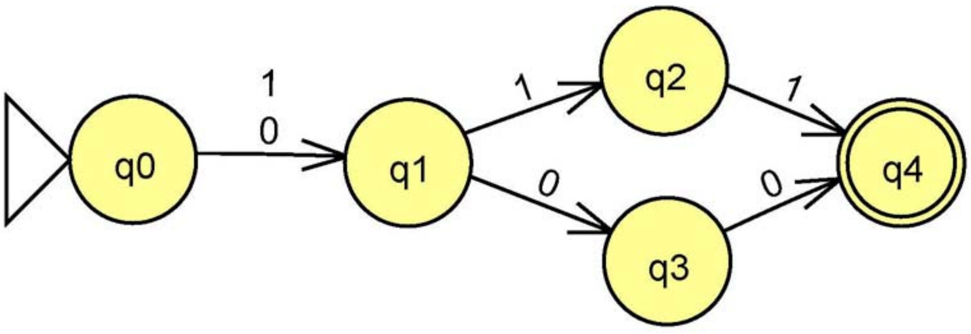
$\mathbf{q}_1 \dashrightarrow \mathbf{1q}_2$

$\mathbf{q}_1 \dashrightarrow \mathbf{0q}_3$

$\mathbf{q}_3 \dashrightarrow \mathbf{0q}_4$

$\mathbf{q}_2 \dashrightarrow \mathbf{1q}_4$

$\mathbf{q}_4 \dashrightarrow \lambda$



Derivar:

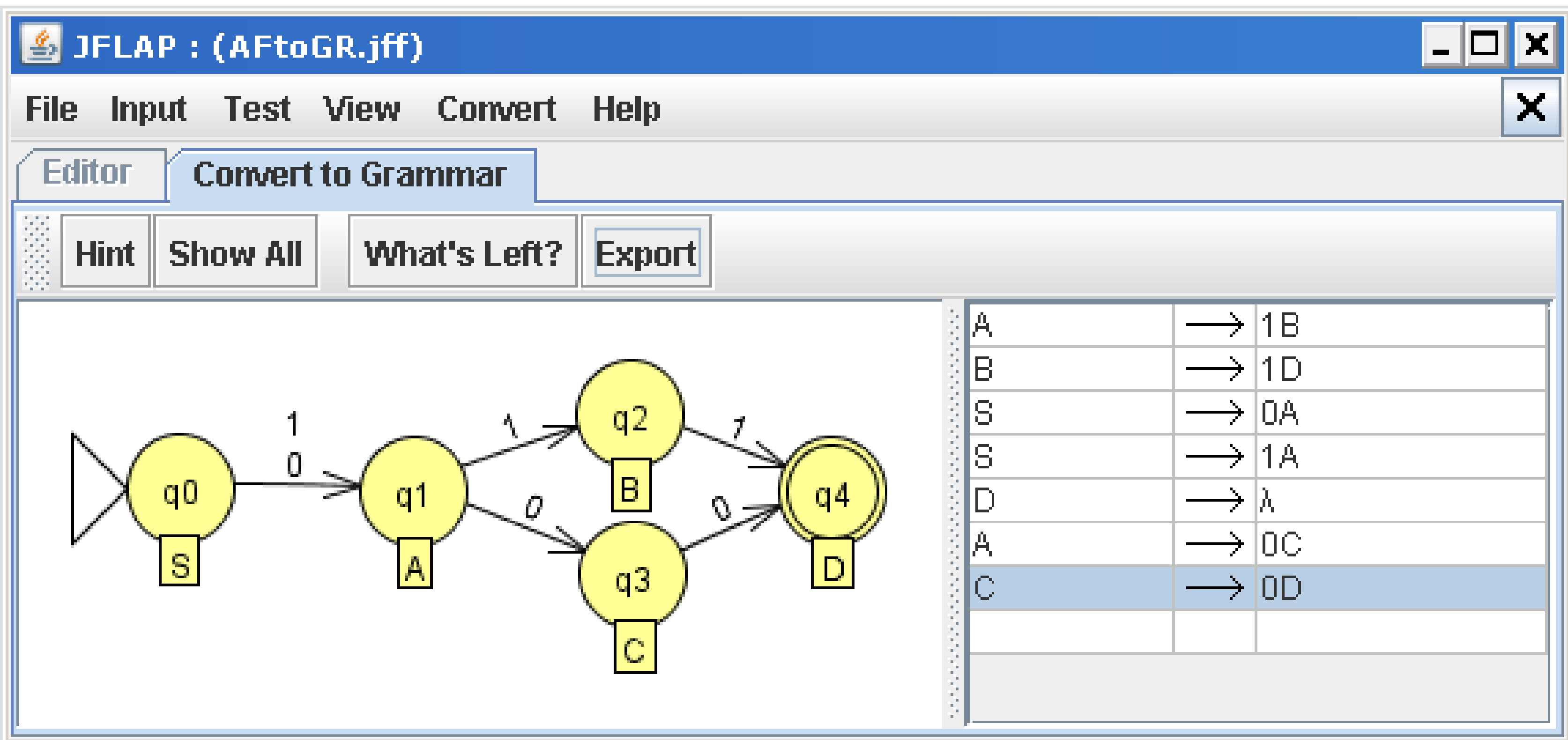
$\mathbf{q}_0 \Rightarrow \mathbf{0q}_1 \Rightarrow \mathbf{00q}_3 \Rightarrow \mathbf{000q}_4$

$\Rightarrow \mathbf{000\lambda = 000}$

AF-GR

L = (0 + 1).(00 + 11)

JFALP: AFtoGR.jff



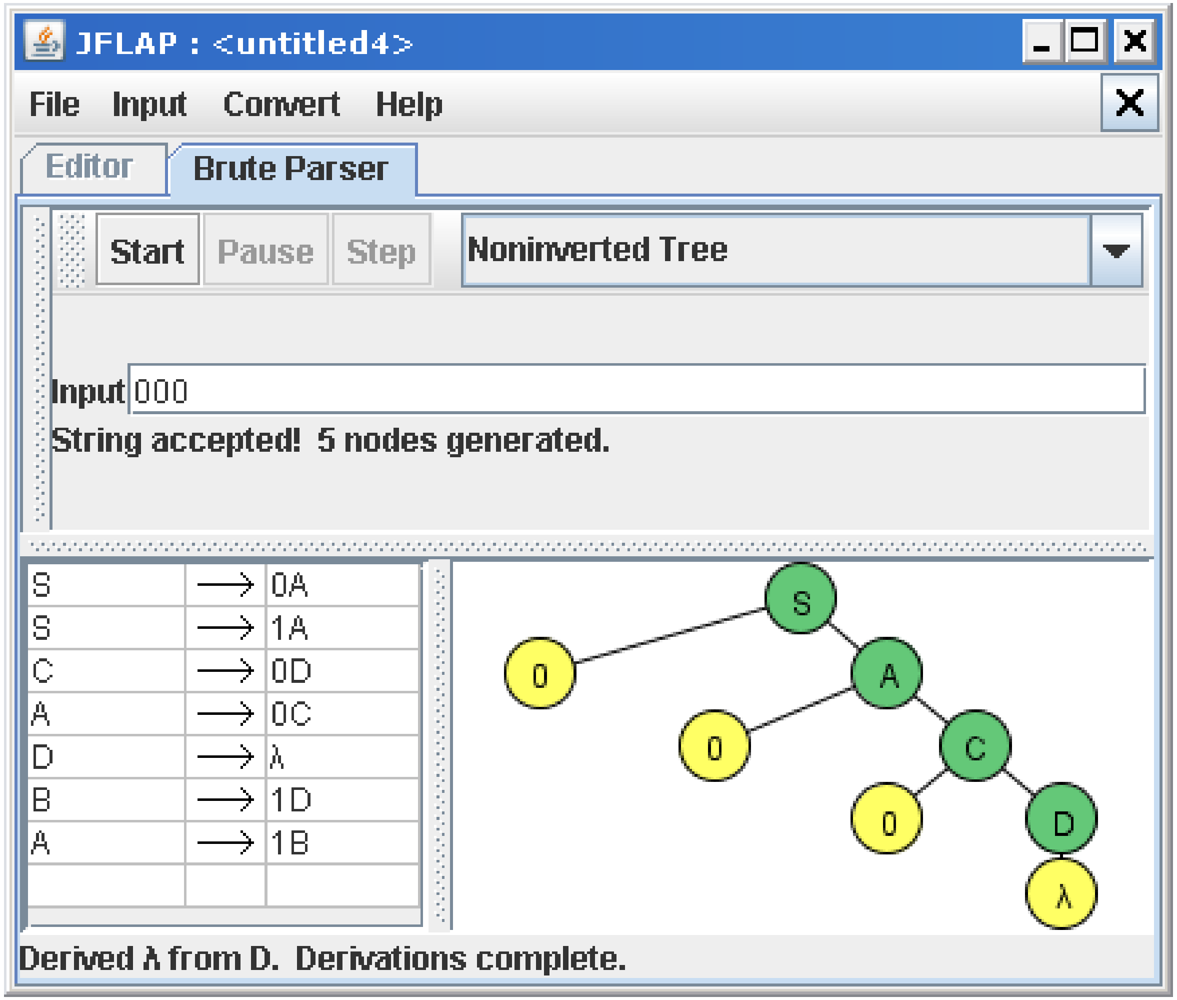
AF-GR

L = (0+1)(00+11)

derivar w = 000

JFALP:

[AFtoGR.jff]



Árbol de derivación

Equivalencia (GR → AF)

Teorema: Si L es un lenguaje generado por una gramática regular G, entonces existe un autómata finito M tal que L = L(G) = L(M).

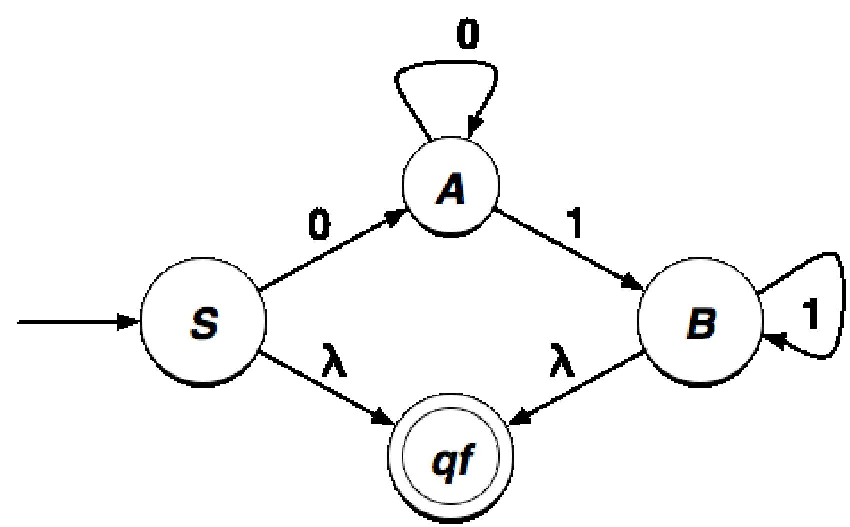
Podemos suponer que G = (V, T, P, S) es una gramática lineal derecha. Obtenemos el diagrama del autómata finito

$M = (V \cup \{q_F\}, T, \delta, S, \{q_F\})$ a partir de la gramática con el siguiente método:

Equivalencia (GR → AF)

1. Si la regla $A \rightarrow aB$ en P entonces añadimos el arco $A \rightarrow B$ etiquetado a
2. Si la regla $A \rightarrow a$ en P añadimos el arco $A \rightarrow q_F$ etiquetado a
3. Si la regla $A \rightarrow \lambda$ en P añadimos el arco $A \rightarrow q_F$ etiquetado λ

p.e: Dada la gramática con Las siguientes reglas: $P = \{S \rightarrow 0A \mid \lambda,\ A \rightarrow 0A \mid 1B,\ B \rightarrow 1B \mid \lambda\}$, el AF que reconoce el lenguaje L(G) es:



Implementación de GR

Lenguajes de programación swi-Prolog (usando DCG = Definite Clause Grammars):

Regla	DCG Prolog
$\mathbf{q_0 \dashrightarrow 0q_1}$	q0 --> [0],q1.
$\mathbf{q_4 \dashrightarrow \lambda}$	q4 --> [].

Notación DCG

- Flecha “-->” transformación.
- Todas las reglas terminan en punto.
- A la izquierda de cada regla siempre hay un solo símbolo no terminal (atomos prolog).
- A la derecha los símbolo están separados por comas.
- Los símbolos terminales van encerrados entre corchetes.

Implementación de GR

Lenguajes de programación swi-Prolog (usando DCG):

```
q0 --> [0],q1.
q0 --> [1],q1.
q1 --> [0],q2.
q1 --> [1],q3.
q2 --> [0],q4.
q3 --> [1],q4.
q4 --> [].
```

Ejemplo: finito.pl
(Swi-Prolog: http://www.swi-prolog.org/)

Implementación de GR

<u>Consultas a Prolog</u>:

?- q0([0,0,0],[]).

true .

?- q0([1,0,X],[]).

X = 0

<u>Ejemplo de generación de Lenguaje</u>:

?- findall(A, q0(A,[]), L).

L = [[0, 0, 0], [0, 1, 1], [1, 0, 0], [1, 1, 1]].

Procedimiento finito para **generar** un lenguaje (finito).

Ejemplo

La GR del trabajo:

q0 --> [el],q1.

q0 --> [este],q1.

q0 --> [un],q1.

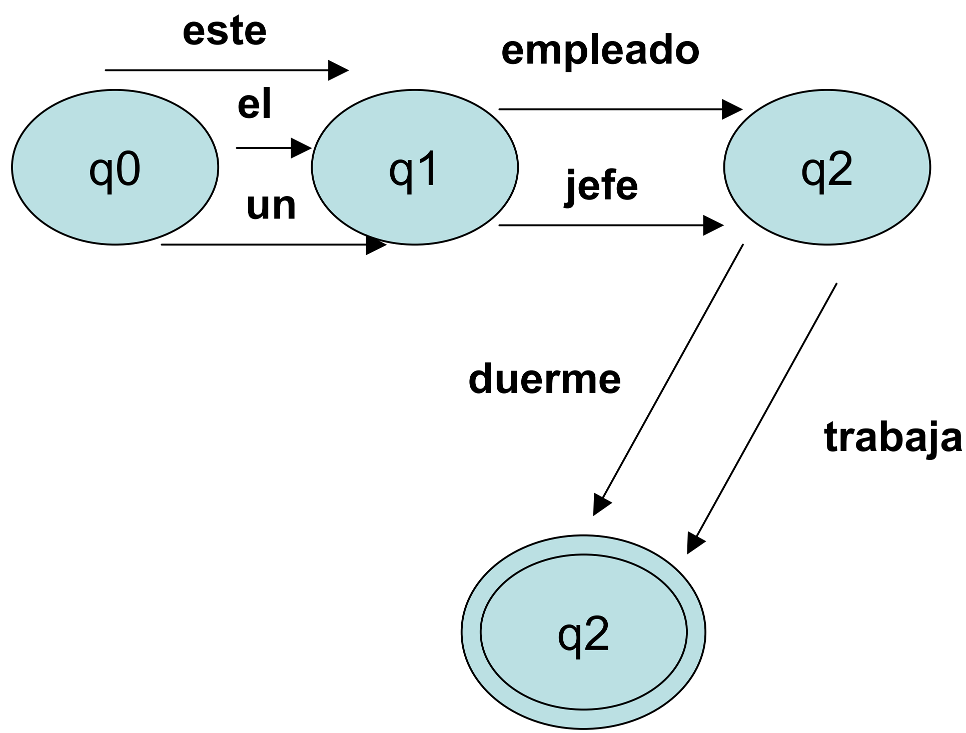
q1 --> [jefe],q2.

q1 --> [empleado],q2.

q2 --> [duerme],q3.

q2 --> [trabaja],q3.

q3 --> [].



Ejemplo

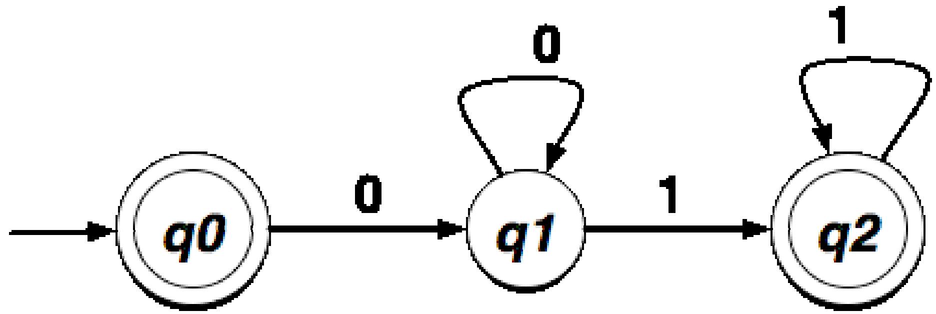
La GR del trabajo:

```
q0 --> [el],q1.
q0 --> [este],q1.
q0 --> [un],q1.
q1 --> [jefe],q2.
q1 --> [empleado],q2.
q2 --> [duerme],q3.
q2 --> [trabaja],q3.
q3 --> [].
```

```
?- findall(A,(q0(A,[]),write_ln(A)),L).
[el, jefe, duerme]
[el, jefe, trabaja]
[el, empleado, duerme]
[el, empleado, trabaja]
[este, jefe, duerme]
[este, jefe, trabaja]
[este, empleado, duerme]
[este, empleado, trabaja]
[un, jefe, duerme]
[un, jefe, trabaja]
[un, empleado, duerme]
[un, empleado, trabaja]
```

Implementación de GR

Ejemplo de generación de Lenguaje infinito: 00*11*



$G = (\{q_0, q_1, q_2\}, \{0, 1\}, q_0, \{q_0 \rightarrow 0q_1 \mid \lambda, q_1 \rightarrow 0q_1 \mid 1q_2, q_2 \rightarrow 1q_2 \mid \lambda\})$

Ejemplo: infinito.pl
?- findall(A, q0(A,[]), L).
Out of local stack

```
q0 --> [0],q1.
q0 --> [].
q1 --> [0],q1.
q1 --> [1],q2.
q2 --> [1],q2.
q2 --> [].
```

Implementación de GR

Reconocer un Lenguaje infinito:

G = ({q0, q1, q2}, {0, 1}, q0, {q0 → 0q1 | λ, q1 → 0q1 | 1q2, q2 → 1q2 | λ})

Ejemplo: infinito.pl

?- q0([0,1,1,0],[]).

fail

?- q0([0,0,1,1,1],[]).

true