U1: EJERCICIOS GRAMATICAS DE CONTEXTO LIBRE

ING. SANDRA RODRIGUEZ AVILA

RECURSIVIDAD

Sea la gramatica, cuyas reglas P, son:

$E \rightarrow E$ **op** T $\qquad E \rightarrow T$

$T \rightarrow$ **id** $\qquad T \rightarrow$ **(** E **)**

Eliminar la Recursividad por la Izquierda.

Algoritmo para eliminar la Recursividad por la Izquierda :

Paso 1: $A \rightarrow A\,\alpha_1 \mid A\,\alpha_2 \mid \ldots \mid A\,\alpha_p$

$A \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_q$

Paso 2:

$A \rightarrow \beta_i \qquad A \rightarrow \beta_i A'$

$A' \rightarrow \alpha_j \qquad A' \rightarrow \alpha_j A'$

Desarrollo:

Paso 1: $E \rightarrow E$ **op** T

$E \rightarrow T$

Paso 2:

$E \rightarrow T \qquad E \rightarrow T E'$

$E' \rightarrow$ **op** $T \qquad E' \rightarrow$ **op** $T E'$

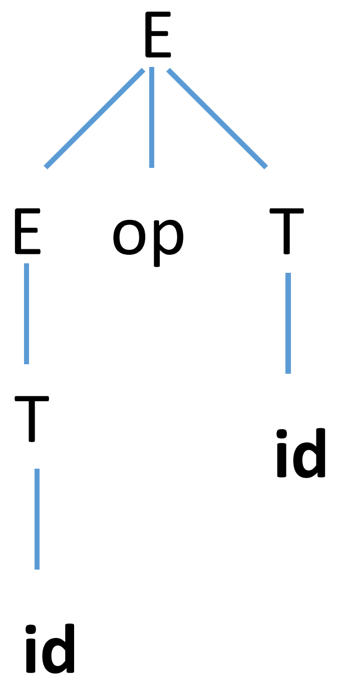
Por lo tanto, la gramática resultante es:
P':

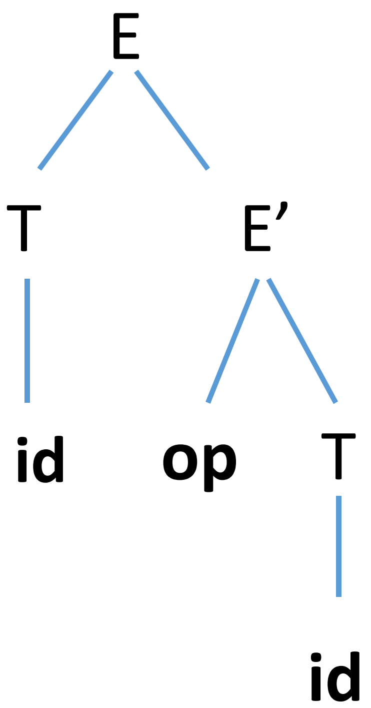
$E \rightarrow T \qquad E \rightarrow T E'$

$E' \rightarrow$ **op** $T \qquad E' \rightarrow$ **op** $T E'$

$T \rightarrow$ **id** $\qquad T \rightarrow$ **(** E **)**

RECURSIVIDAD





RECURSIVIDAD POR LA IZQUIERDA

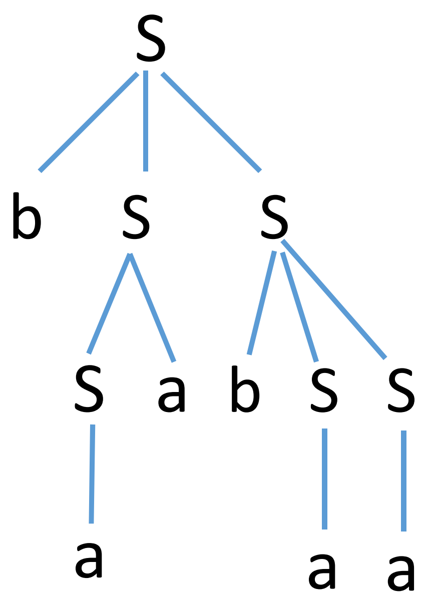
SIN RECURSIVIDAD POR LA IZQUIERDA

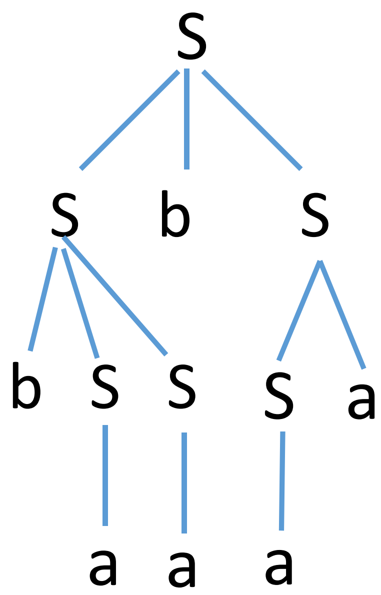
AMBIGUEDAD

Demuestra que la siguiente la gramática es ambigua:

- P = {S → a | S a | b S S | S S b | S b S}

Se deberá encontrar una cadena de terminales que tenga asociadas dos derivaciones por la izquierda diferentes.





La tira o cadena de terminales baabaa presenta dos derivaciones por la izquierda diferentes.

Por lo tanto, la gramática es AMBIGUA.

VACUIDAD DEL LENGUAJE O LENGUAJE VACIO

Comprueba si la siguiente gramática genera un lenguaje vacío o no:

- P ={ S → aACB | AD | Cba   A → aD | BCe | Bb

  B → Ba | ABDa | b   C → Dca

  D → ACb | cAS }

```
Algoritmo Lenguaje Vacío
begin
    VIEJO := Ø
    NUEVO := {A|(A →t) de P y se cumple (t ε T*)}
    while NUEVO <>VIEJO do begin
        VIEJO := NUEVO;
        NUEVO :=  VIEJO U {B|(B →α) de P y
                           α ε (T U VIEJO)*}
    end;
    if S ε NUEVO then VACIO:="no" else VACIO:= "si"
end
```

	VIEJO	NUEVO
1.	Ø	{ B }
2.	{ B }	{ B, A }
3.	{ B, A }	{ B, A }

son iguales

$S \notin$ NUEVO = { B, A }

Por lo tanto, el Lenguaje que genera esta gramática SI es VACIO

SUPRESION DE SIMBOLOS INUTILES

Dada la siguiente gramática, construye otra equivalentes sin símbolos inútiles.

- P ={ S → ACBd | BaB   A → aAd | BCa | ab
  B → bBb | a   C → CAC | ACc}

PASO 1: Algoritmo para determinar símbolos terminables

```
begin
    VIEJO := Ø
    NUEVO := {A|(A →t) de P y se cumple (t ε T*)}
    while VIEJO <>NUEVO do begin
        VIEJO := NUEVO;
        NUEVO :=  VIEJO U {B|(B →α) de P y
                          α ε (T U VIEJO)*}
    end;
    N':=NUEVO
end
```

Se incluyen en N' todas las variables A que tengan una regla A →t

P'. conj. de reglas cuyos símbolos están en N' U T

	VIEJO	NUEVO	
1.	Ø	{ A, B }	
2.	{ A, B }	{A, B, S }	
3.	{ A, B, S }	{A, B, S }	son iguales

N' = NUEVO = {A, B, S }

P' = { S → BaB   A → aAd | ab   B → bBb | a }

Por lo tanto, "C" es un símbolo inútil No Terminable.

SUPRESION DE SIMBOLOS INUTILES

Teniendo en cuenta la Gramática resultante del Paso 1:

- P’ = { S → BaB   A → aAd | ab   B → bBb | a }

PASO 2: Algoritmo para determinar símbolos accesibles desde S

```
begin
    VIEJO := {S}
    NUEVO := {X |(S → αXβ) de P U VIEJO
    while VIEJO <>NUEVO do begin
        VIEJO := NUEVO;
        NUEVO :=  VIEJO U {Y | A → αYβ de
            P y A esta en VIEJO}
    end;
    N’:=NUEVO ∩ N; T’:=NUEVO ∩ T;
  (P’:conj. de reglas cuyos símbolos están en N’
  U T’)
  end
```

	VIEJO	NUEVO
1.	{ S }	{ S, B, a }
2.	{ S, B, a }	{ S, B, a, b }
3.	{S, B, a, b }	{S, B, a, b } son iguales

N’’ = N’ ∩ NUEVO = { S, B}

T’ = T ∩ NUEVO = { a, b}

P’’ = { S → BaB   B → bBb | a }

Por lo tanto, “A y d” son símbolos inútiles No Accesibles desde S.

REGLAS LAMBDA λ

Dada la siguiente gramática, obtén otra gramática equivalente sin reglas λ.

- P = { S → aS | AB | AC   A → aA | λ
  B → bB | bS   C → cC | λ}

Algoritmo Reglas λ
begin
    VIEJO := Ø
    NUEVO := {A|(A → λ) de P}
    while VIEJO <>NUEVO do
    begin
        VIEJO := NUEVO;
        NUEVO := VIEJO U {B|(B →α) y todos los símbolos de α son anulables}
    end;
    ANULABLES :=NUEVO
end
Si el axioma S es anulable, entonces λ pertenece al lenguaje y para aislarla añadimos a P', la regla:
    S' → λ |S   ampliándose N con S'

	VIEJO	NUEVO	
1.	Ø	{ A, C }	
2.	{ A, C }	{A,C, S }	
3.	{ A, C, S }	{A, C, S }	son iguales

ANULABLES = NUEVO = {A, C, S }

P' = { S'→ S | λ   S → aS | a | AB | B | AC | A | C
A → aA | a   B → Bb | bS | b   C →cC | c }

Como "S" es un símbolo anulable, se agrega el No terminal S' como Símbolo inicial.

REGLAS UNITARIAS

Dada la siguiente gramática, obtén otra gramática equivalente sin reglas λ.

- P = { E → E+T | T T → T*F | F

  F → (E) | a }

Algoritmo Reglas Unitarias
Los conjuntos NA para cada A de N con las meta nociones B tales que A ⇒* B

```
begin
    VIEJO := Ø;
    NUEVO := A;
    while VIEJO <>NUEVO do
    begin
        VIEJO := NUEVO;
        NUEVO :=  VIEJO U {C|(B →C)en P y B esta en VIEJO}
    end;
    NA:=NUEVO
end
```

P', si B →α esta en P y no es una regla unitaria, poner A →α en P' para todas las A para las que B este en NA.

Para NE:	VIEJO	NUEVO	
1.	Ø	{ E }	
2.	{ E }	{ E, T}	
3.	{E, T}	{E, T, F }	
4.	{E, T, F}	{E, T, F }	son iguales

NE = NUEVO

Reglas para NE = { E → E+T | T*F | (E) | a }

REGLAS UNITARIAS

Algoritmo Reglas Unitarias
Los conjuntos NA para cada A de N con las meta nociones B tales que A ⇒* B

```
begin
    VIEJO := Ø;
    NUEVO := A;
    while VIEJO <>NUEVO do
    begin
        VIEJO := NUEVO;
        NUEVO :=  VIEJO U {C|(B →C)en P y B esta en VIEJO}
    end;
    NA:=NUEVO
end
```

P', si B →α esta en P y no es una regla unitaria, poner A →α en P' para todas las A para las que B este en NA.

Para NT:	VIEJO	NUEVO	
1.	Ø	{ T }	
2.	{ T }	{ T, F}	
3.	{ T, F }	{ T, F }	son iguales

NT = NUEVO

Reglas para NT = { T → T*F | (E) | a }

Para NF:	VIEJO	NUEVO	
1.	Ø	{ F }	
2.	{ F }	{ F}	son iguales

NF = NUEVO

Reglas para NE = { F → (E) | a }

P' = { E → E+T | T*F | (E) | a
T → T*F | (E) | a
F → (E) | a }

FORMA NORMAL DE CHOMSKY

$$A \rightarrow BC \qquad\qquad A \rightarrow a$$

Convertir la siguiente gramática a la FNC.

$P = \{S \rightarrow BA \qquad A \rightarrow 01ABO \qquad A \rightarrow 0 \qquad B \rightarrow 1\}$

Procedimiento:

a. Las reglas $S \rightarrow BA \qquad A \rightarrow 0 \qquad B \rightarrow 1$ ya están en FNC

b. Para la regla restante $A \rightarrow 01AB0$ aplicar transformaciones

Regla:

$$A \rightarrow 0\underbrace{1ABO}_{A_2} \qquad A \rightarrow A_1A_2 \qquad A_1 \rightarrow 0$$

$$A_2 \rightarrow 1\underbrace{ABO}_{A_4} \qquad A_2 \rightarrow A_3A_4 \qquad A_3 \rightarrow 1$$

$$A_4 \rightarrow A\underbrace{BO}_{A_5} \qquad A_4 \rightarrow AA_5$$

$$A_5 \rightarrow B\underbrace{O}_{A_6} \qquad A_5 \rightarrow BA_6 \qquad A_6 \rightarrow 0$$

GRAMATICA EN FNC:

$P' = \{ \quad S \rightarrow BA \qquad A \rightarrow 0 \qquad B \rightarrow 1 \qquad A \rightarrow A_1A_2$

$A_1 \rightarrow 0 \qquad A_2 \rightarrow A_3A_4 \qquad A_3 \rightarrow 1$

$A_4 \rightarrow AA_5 \qquad A_5 \rightarrow BA_6 \qquad A_6 \rightarrow 0 \ \}$