PRÁCTICA 7 DISEÑO DE GRAMÁTICAS EN JFLAP

Computabilidad y Algoritmia

Alejandro Rodríguez Rojas
alu0101317038@ull.edu.es

EJERCICIO 1: Gramáticas Independientes

1. Diseñar una gramática independiente del contexto que genere el lenguaje

$$L = \{a^n b^n \mid n \geq 0\}$$

LHS		RHS
S	⟶	aSb
S	⟶	λ

Definición de la gramática

Vamos a construir una gramática con la siguiente estructura de producción:

1. **Símbolo inicial**: S
2. **Producciones**:
    - S→aSb: Esta producción genera una "a" al inicio y una "b" al final, manteniendo la relación a^n b^n.
    - S→ε: Esta producción representa la cadena vacía, que corresponde al caso en que n=0.

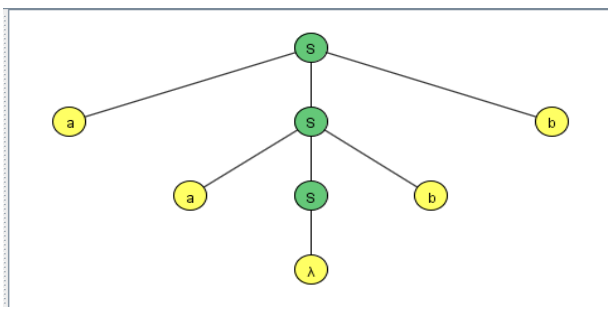
Gramática formalmente definida

La gramática G=(V,Σ,R,S), donde:

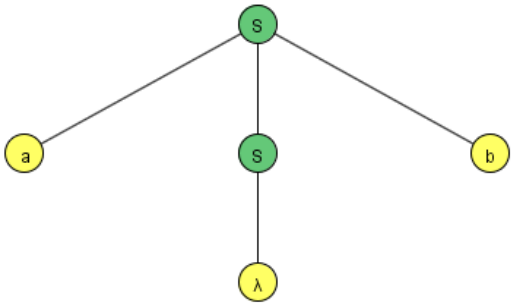
- **V** (variables o no terminales): S
- Σ (alfabeto terminal): {a,b}
- **R** (reglas de producción):
    1. S→aSb
    2. S→ε
- **S** (símbolo inicial): S

Cadenas aceptadas:

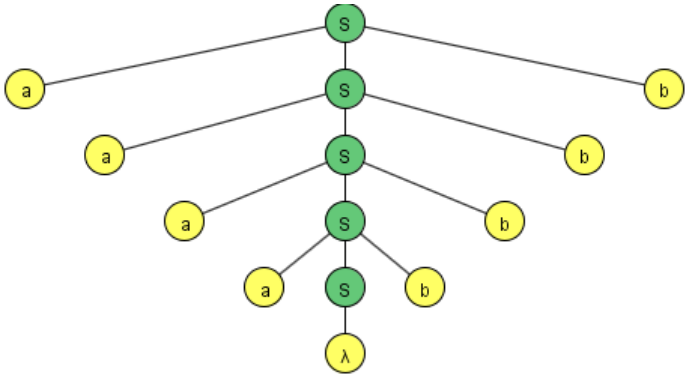
1. aabb:



2. ab



3. aaaabbbb



Gramática simplificada

Paso 1 → Eliminar producciones vacías: solo la del símbolo no terminal S, así que esa la podemos mantener

Paso 2 → Eliminar producciones unitarias: no hay

Paso 3 → Eliminar símbolos y producciones inútiles: no hay

- Etapa 1: Generadores de cadenas: V' = {S}
- Etapa 2: Alcanzables : Σ = {a, b} → Σ' = {a, b}

2. Diseñar una gramática independiente del contexto que genere el lenguaje

$$L = \{a^n b^m \mid n, m \geq 0, n \neq m\}$$

LHS		
S	⟶	A
S	⟶	B
A	⟶	aAb
A	⟶	aA
A	⟶	a
B	⟶	aBb
B	⟶	Bb
B	⟶	b

Definición de la gramática

Vamos a construir una gramática con la siguiente estructura de producción:

3. **Símbolo inicial**: S
4. **Producciones**:
    - S⟶ A | B
    - A⟶ aAb | aA | a: Genera cadenas con más “a’s” que “b’s” (n > m)
    - B⟶ aBb | Bb | b: Genera cadenas con más “b’s” que “a’s” (n < m)

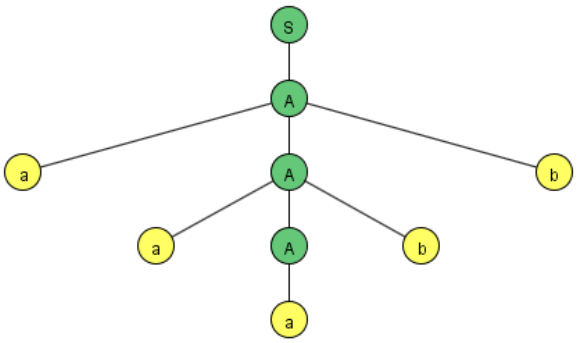
Gramática formalmente definida

La gramática G=(V,Σ,R,S), donde:

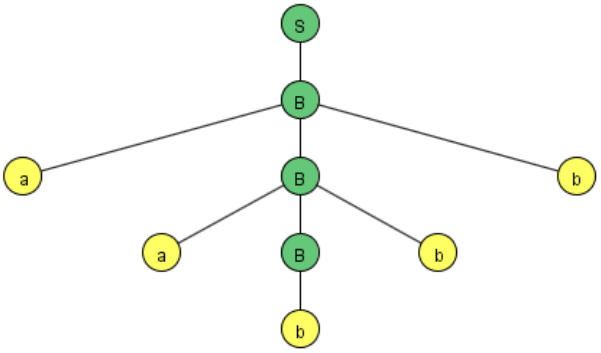
- **V** (variables o no terminales): S, A, B
- Σ (alfabeto terminal): {a,b}
- **R** (reglas de producción):
    1. S⟶ A | B
    2. A⟶ aAb | aA | a
    3. B⟶ aBb | Bb | b
- **S** (símbolo inicial): S

Cadenas aceptadas:

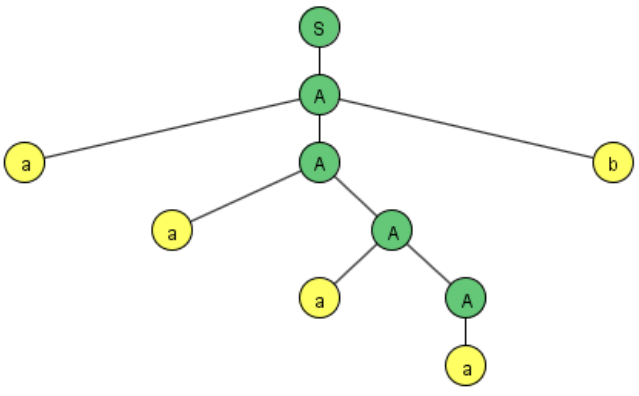
1. aaabb



2. aabbb



3. aaaab



Gramática simplificada

Paso 1 → Eliminar producciones vacías: no hay

Paso 2 → Eliminar producciones unitarias: S → A y S → B

LHS		
S	→	aAb
S	→	aA
S	→	a
S	→	aBb
S	→	Bb
S	→	b
A	→	aAb
A	→	aA
A	→	a
B	→	aBb
B	→	Bb
B	→	b

Paso 3 → Eliminar símbolos y producciones inútiles: no hay

- Etapa 1: Generadores de cadenas: V' = {S,A,B}
- Etapa 2: Alcanzables : Σ = {a, b} → Σ' = {a, b}

3. Diseñar una gramática independiente del contexto para el lenguaje

$$L = \{ww^I \mid w \in \{a, b\}^*\}$$

LHS		RHS
S	$\longrightarrow$	aSa
S	$\longrightarrow$	bSb
S	$\longrightarrow$	λ

Diseño de la Gramática

Para construir esta gramática, podemos usar las siguientes producciones:

1. **Símbolo inicial**: S
2. **Producciones**:
    - S→aSa: Genera un "a" al principio y al final, y genera recursivamente el mismo patrón en el medio.
    - S→bSb: Genera un "b" al principio y al final, y genera recursivamente el mismo patrón en el medio.
    - S→ε: Permite que la cadena termine, representando la cadena vacía (caso base para que la recursión termine).

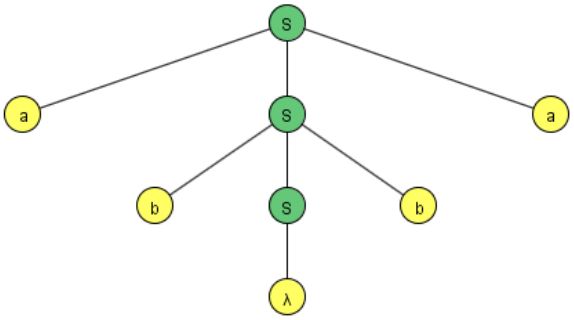
Gramática formalmente definida

La gramática G=(V,Σ,R,S), donde:

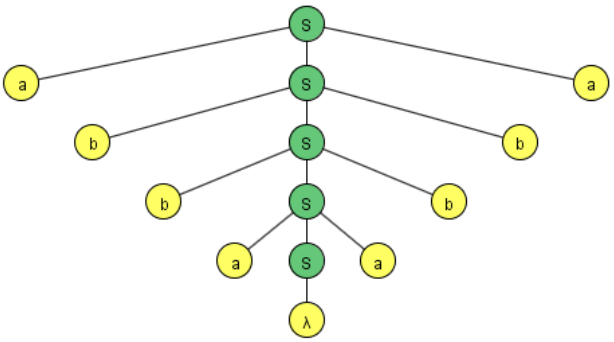
- **V** (variables o no terminales): S
- Σ (alfabeto terminal): {a,b}
- **R** (reglas de producción):
    1. S→aSa
    2. S→bSb
    3. S→ε
- **S** (símbolo inicial): S

Cadenas aceptadas:

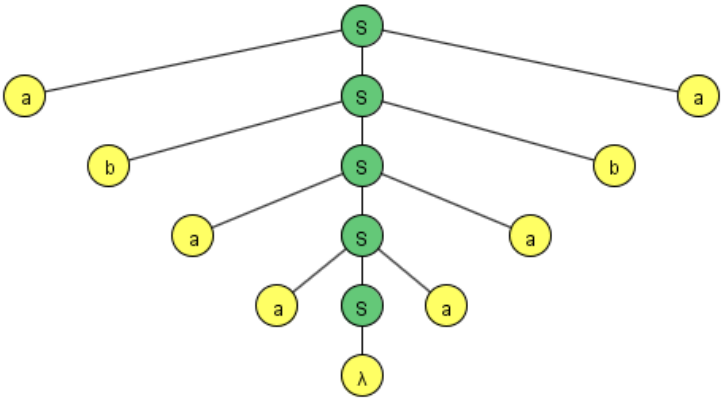
1. abba



2. abbaabba



3. abaaaaba



Gramática simplificada

Paso 1 → Eliminar producciones vacías: solo existe la de S, pero la mantenemos porque el lenguaje lo requiere

Paso 2 → Eliminar producciones unitarias: no hay

Paso 3 → Eliminar símbolos y producciones inútiles: no hay

Etapa 1: Generadores de cadenas: V' = {S}

Etapa 2: Alcanzables : Σ = {a, b} → Σ' = {a, b}

Gramática ya simplificada.

4. Diseñar una gramática independiente del contexto que genere el lenguaje

$$L = \{a^n b^m c^n \mid n \geq 0, m \ impar\}$$

LHS		
S	⟶	aSc
S	⟶	B
B	⟶	bC
C	⟶	bbC
C	⟶	λ

Gramática

1. **Símbolo inicial**: S
2. **Producciones**:
    - **S→aSc**: Esta producción separa la generación de "a's" y "c's" (controlada por A y C, respectivamente) y la generación de una cantidad impar de "b's" (controlada por B).
    - **B→bC**: Genera la primera “b”
    - **C→bbC | ε**: Asegura que el número de "b's" es impar

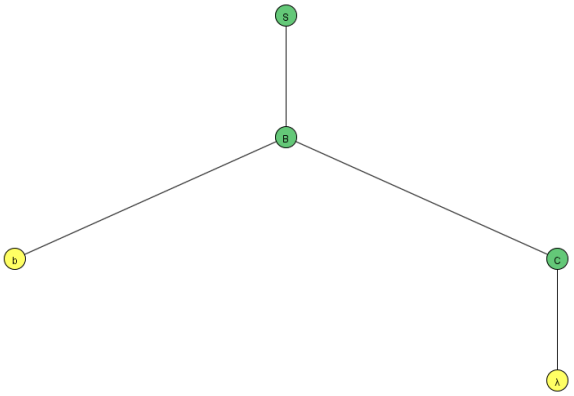
Gramática formalmente definida

La gramática G=(V,Σ,R,S), donde:

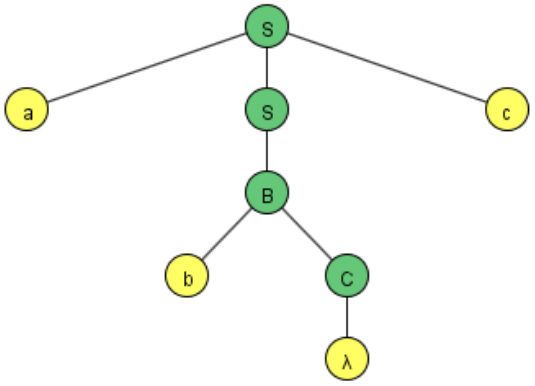
- **V** (variables o no terminales): S,A,B,C
- Σ (alfabeto terminal): {a,b,c}
- **R** (reglas de producción):
    1. S→aSc
    2. B→bC
    3. C→bbC | ε
    4. **S** (símbolo inicial): S

Cadenas aceptadas:

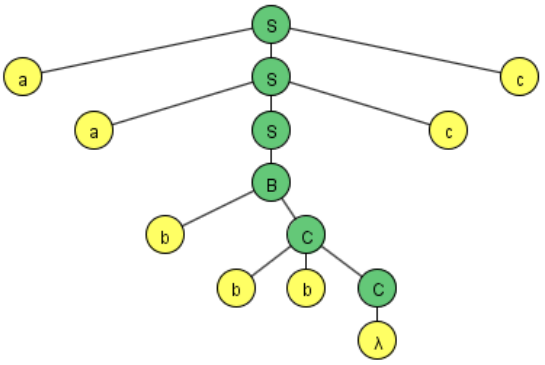
1. b



2. abc



3. aabbbcc



Gramática simplificada

Paso 1 → Eliminar producciones vacías: C → ε

Paso 2 → Eliminar producciones unitarias: S → B, sustituimos las producciones de B en S

Paso 3 → Eliminar símbolos y producciones inútiles: B → bC

Etapa 1: Generadores de cadenas: V' = {S,C}

Etapa 2: Alcanzables : Σ = {a, b, c} → Σ' = {a, b, c}

Gramática final**:**

S	→	aSc
S	→	bC
S	→	b
C	→	bbC
C	→	bb

5. Diseñar una gramática independiente del contexto que genere el lenguaje

$$L = \{a^n b^m c^m d^n \mid n, m \geq 1\}$$

LHS		
S	$\longrightarrow$	aSd
S	$\longrightarrow$	A
A	$\longrightarrow$	bAc
A	$\longrightarrow$	λ

Diseño de la Gramática

Para construir esta gramática, podemos usar las siguientes producciones:

3. **Símbolo inicial**: S
4. **Producciones**:
    - S→aSd: Esta producción separa la generación de "a's" y "d's" y la generación de "b's" y “c’s” (controlada por A).
    - S→A: Pasa a la generación de “b” y “c”
    - A→bAc: Genera el mismo número de “b’s” y “c’s”
    - A → ε: Termina

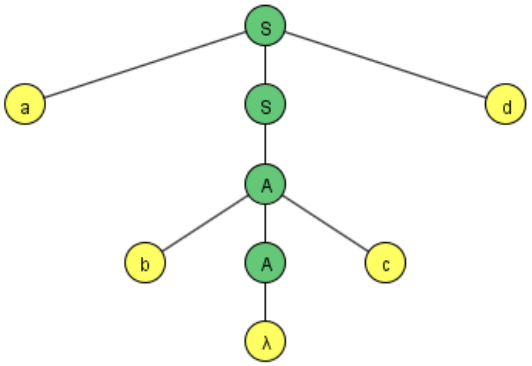
Gramática formalmente definida

La gramática G=(V,Σ,R,S), donde:

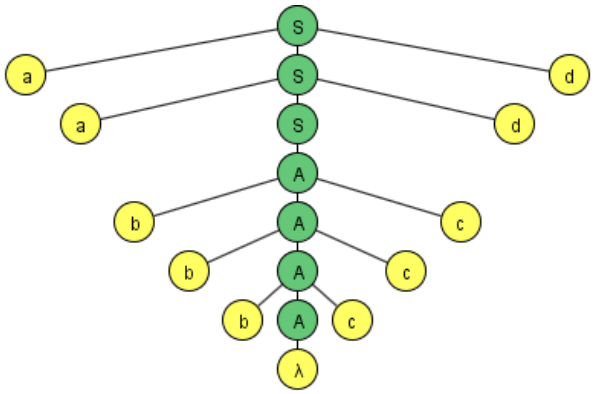
- **V** (variables o no terminales): S, A
- Σ (alfabeto terminal): {a,b,c,d}
- **R** (reglas de producción):
    1. S→aSd |A
    2. A→bAc |ε
- **S** (símbolo inicial): S

Cadenas aceptadas:

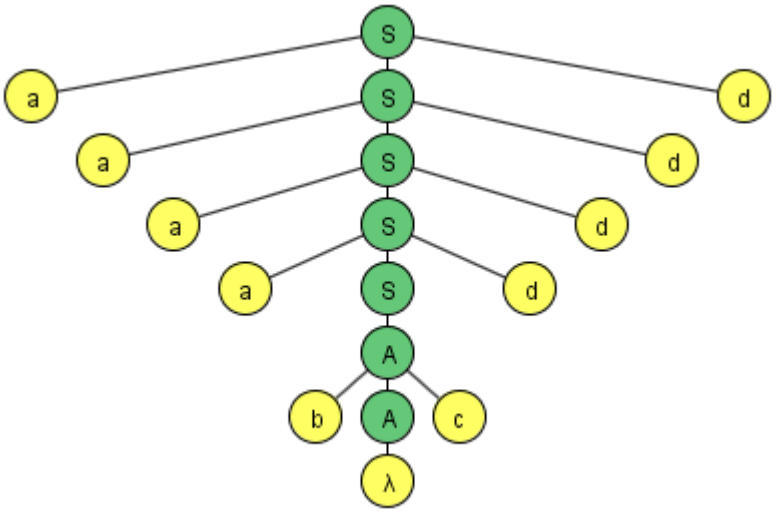
1. abcd



2. aabbbcccdd



3. aaaabcdddd



Gramática simplificada

Paso 1 → Eliminar producciones vacías: A → ε

Paso 2 → Eliminar producciones unitarias: S → A, sustituimos las producciones de A en S

Paso 3 → Eliminar símbolos y producciones inútiles: no hay

Etapa 1: Generadores de cadenas: V' = {S,A}

Etapa 2: Alcanzables : Σ = {a, b, c, d} → Σ' = {a, b, c, d}

Gramática final**:**

LHS		
S	→	aSd
S	→	bAc
S	→	bc
A	→	bAc
A	→	bc

6. Diseñar una gramática independiente del contexto que genere el lenguaje

$$L = \{a^n b^m \mid n > m \geq 0\}$$

LHS		
S	$\longrightarrow$	aSb
S	$\longrightarrow$	aS
S	$\longrightarrow$	a

Diseño de la Gramática

Para construir esta gramática, podemos usar las siguientes producciones:

5. **Símbolo inicial**: S
6. **Producciones**:
    - S→aSb: Genera cadenas de “a’s” seguidas de “b’s”
    - S→aS: Genera cadenas de a
    - S→a: Asegura un mayor número de a

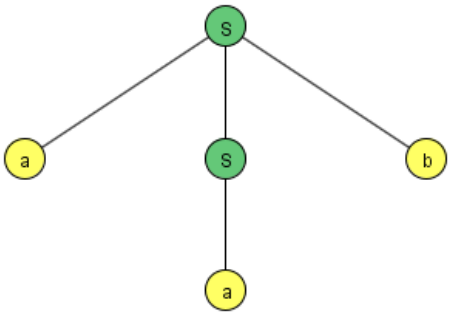
Gramática formalmente definida

La gramática G=(V,Σ,R,S), donde:

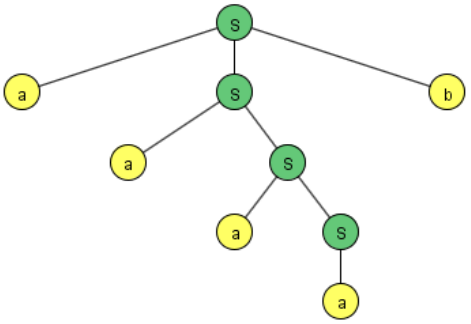
- **V** (variables o no terminales): S
- Σ (alfabeto terminal): {a,b}
- **R** (reglas de producción):
    1. S→aSb | aS | a
- **S** (símbolo inicial): S

Cadenas aceptadas:

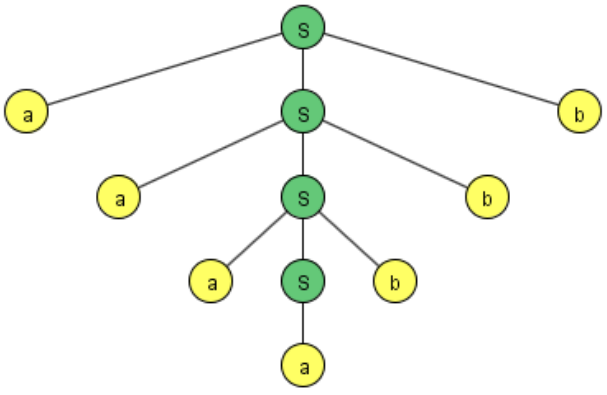
1. aab



2. aaaab



3. aaaabbb



Gramática simplificada

Paso 1 → Eliminar producciones vacías: no hay

Paso 2 → Eliminar producciones unitarias: no hay

Paso 3 → Eliminar símbolos y producciones inútiles: no hay

Etapa 1: Generadores de cadenas: V' = {S}

Etapa 2: Alcanzables : Σ = {a, b} → Σ' = {a, b}

Gramática ya simplificada

7. Diseñar una gramática independiente del contexto que genere el lenguaje

$$L = \{a^i b^j c^{i+j} \mid i, j \geq 1, i + j \ par\}$$

LHS		
S	⟶	aAc
A	⟶	aSc
A	⟶	bBc
B	⟶	bbBcc
B	⟶	λ
S	⟶	B

Diseño de la Gramática

Para construir esta gramática, podemos usar las siguientes producciones:

7. **Símbolo inicial**: S
8. **Producciones**:
    - S→aAc
    - S → B:
    - A→aSc
    - A→bBc
    - B→bbBcc
    - B→ε

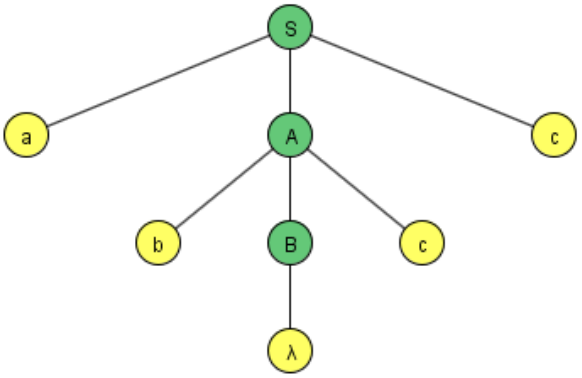
Gramática formalmente definida

La gramática G=(V,Σ,R,S), donde:

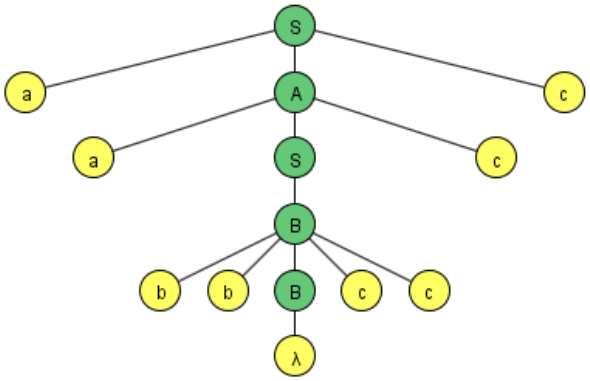
- **V** (variables o no terminales): S, A, B
- Σ (alfabeto terminal): {a,b,c}
- **R** (reglas de producción):
    1. S→aAc | B
    2. A→aSc | bBc
    3. B→bbBcc | ε
- **S** (símbolo inicial): S

Cadenas aceptadas:

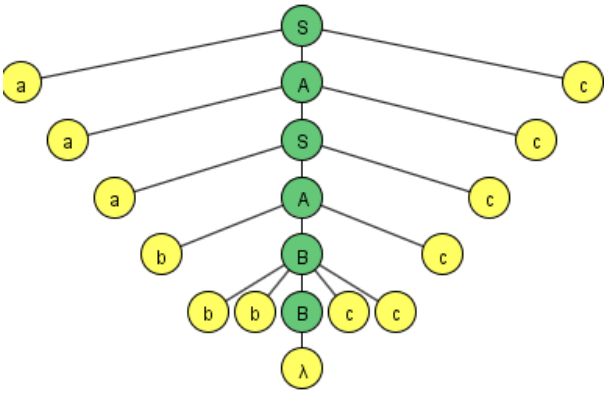
1. abcc



2. aabbcccc



3. aaabbbcccccc



8. Diseñar una gramática independiente del contexto que genere el lenguaje de las expresiones booleanas con los operadores AND, OR, y NOT, usando paréntesis para agrupar. Ejemplos: “(true AND false)”, “NOT (true OR false)”, “true AND (false OR true)”.

LHS		RHS
E	→	T
E	→	T OR E
T	→	F
T	→	F AND T
F	→	NOT F
F	→	true
F	→	false
F	→	(E)

Gramática

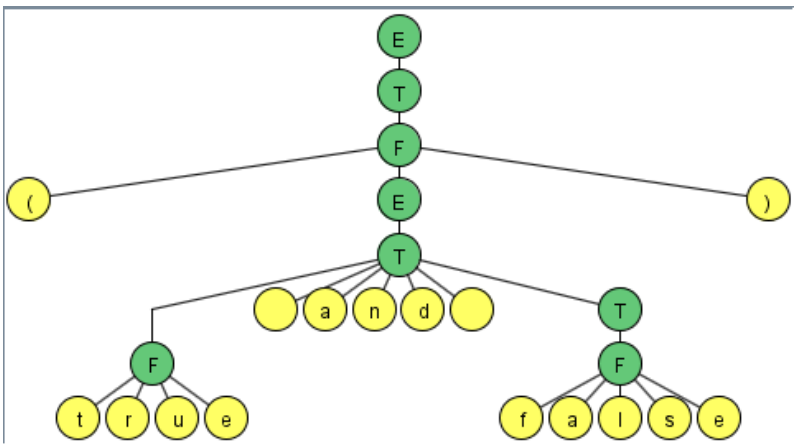
1. **Símbolo inicial**: E (expresión)
2. **Producciones**:
   - E→T
   - E→T OR E
   - T→F
   - T→F AND T
   - F→NOT F
   - F→true
   - F→false
   - F→(E)

Explicación de las producciones

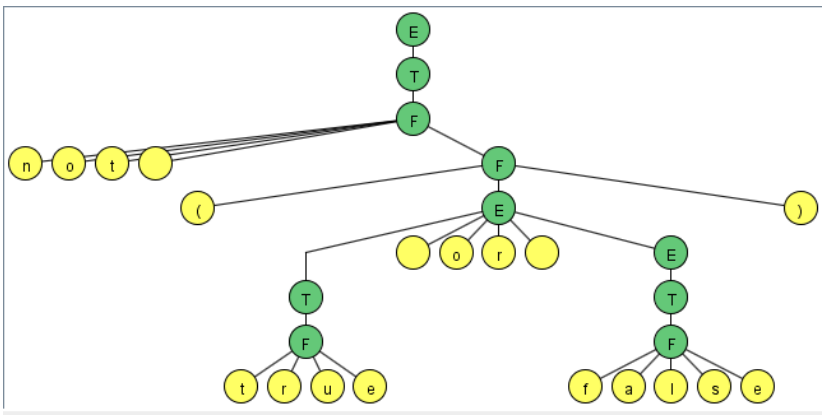
- **E**: Representa una expresión booleana completa y permite la operación **OR**.
  - E→T: Permite que E se reduzca a un término.
  - E→T OR E: Permite la operación **OR** entre dos términos o expresiones.
- **T**: Representa un término, que puede ser una operación **AND** entre factores.
  - T→F: Permite que T se reduzca a un factor.
  - T→F AND T: Permite la operación **AND** entre dos factores o términos.
- **F**: Representa un factor, que puede ser un valor booleano, una operación **NOT**, o una expresión entre paréntesis.
  - F→NOT F: Permite aplicar la operación **NOT** a un factor.
  - F→true: Representa el valor booleano **true**.
  - F→false: Representa el valor booleano **false**.
  - F→(E): Permite agrupar expresiones con paréntesis.

Cadenas Aceptadas:

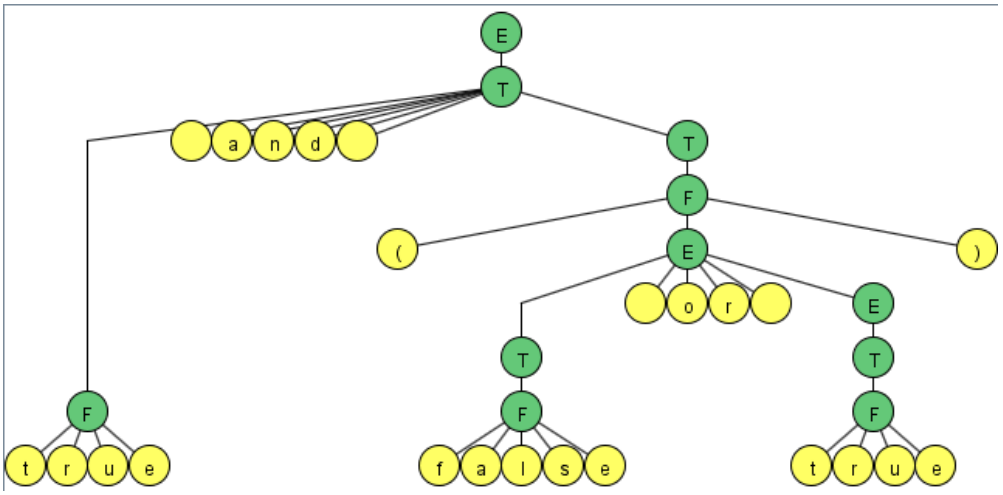
1. (true and false)



2. not (true or false)



3. true and (false or true)



Gramática simplificada

Dado que no había producciones innecesarias, lambda, unitarias ni símbolos inaccesibles, la gramática inicial es ya la versión simplificada.

9. Diseñar una gramática independiente del contexto que genere expresiones aritméticas simples con suma y multiplicación, utilizando los símbolos +, ∗, (,) y los números 0, 1, etc. Ejemplos: “1+2”, “(1+2)*3”, “4*(5+6)”.

LHS		
E	⟶	E+T
E	⟶	T
T	⟶	T*F
T	⟶	F
F	⟶	(E)
F	⟶	0
F	⟶	1
F	⟶	2
F	⟶	3
F	⟶	4
F	⟶	5
F	⟶	6
F	⟶	7
F	⟶	8
F	⟶	9

Diseño de la Gramática

Para construir esta gramática, podemos usar las siguientes producciones:

9. **Símbolo inicial**: E (Expresión)
10. **Producciones**:
    - E⟶ E + T: Define una expresión general que puede ser una suma de términos o un solo término.
    - T⟶T ∗ F ∣ F: Define un término que puede ser una multiplicación de factores o un solo factor.
    - F⟶(E) ∣0 ∣1 ∣2 ∣3 ∣4 ∣5 ∣6 ∣7 ∣8 ∣9: Define un factor como una expresión entre paréntesis o cualquier número entre 0 y 9.

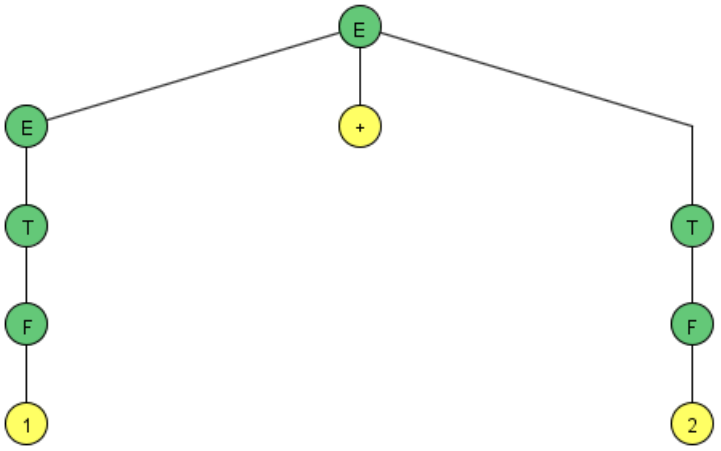
Gramática formalmente definida

La gramática **G=(V,Σ,R,S)**, donde:

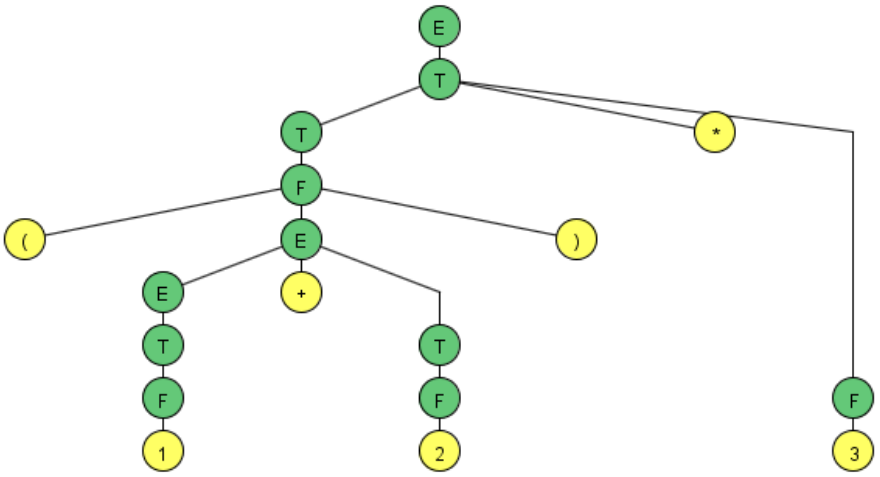
- **V** (variables o no terminales): E, T, F
- **Σ** (alfabeto terminal): {+,*,(),0, 1, 2, 3, 4, 5, 6, 7,8 ,9}
- **R:** Ya mencionaadas
- **S** (símbolo inicial): E

Cadenas aceptadas:

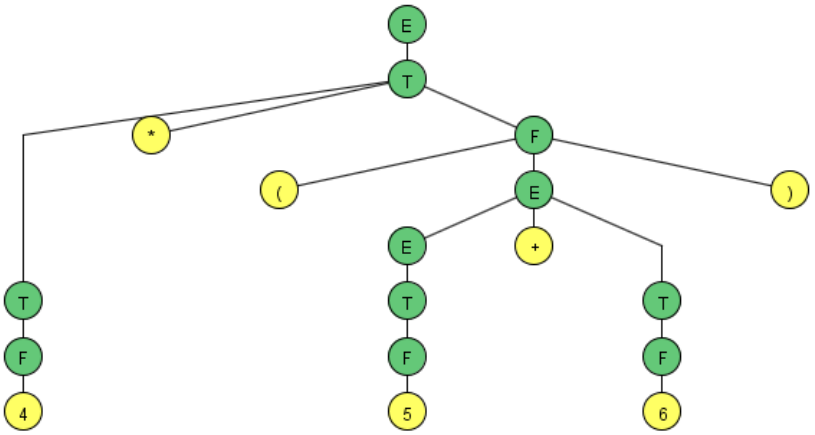
1. 1+2



2. (1+2)*3



3. 4*(5+6)



Gramática simplificada

Dado que no había producciones innecesarias, lambda, unitarias ni símbolos inaccesibles, la gramática inicial es ya la versión simplificada.

10. Diseñar una gramática independiente del contexto que genere el lenguaje de listas anidadas usando corchetes, como en los lenguajes de programación. Ejemplos: [], [[]], [[[]]], [[1,2],[3,4]].

LHS		
E	→	[L]
L	→	E
L	→	E,L
L	→	λ
L	→	0
L	→	1
L	→	2
L	→	3
L	→	4
L	→	5
L	→	6
L	→	7
L	→	8
L	→	9

Diseño de la Gramática

Para construir esta gramática, podemos usar las siguientes producciones:

11. **Símbolo inicial**: E (Expresión)
12. **Producciones**:

    E→[L]: Define que una lista está entre corchetes y contiene elementos E.

    L→E ∣E,L ∣ε:

- E: Permite que una lista contenga una sublista.
- E,L: Permite listas con múltiples elementos separados por comas.
- ε: Define la lista vacía cuando no hay elementos entre los corchetes.

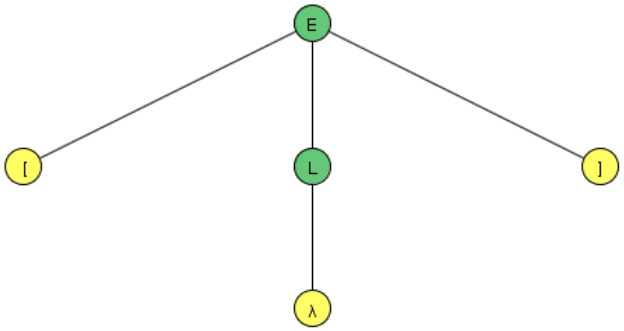
Gramática formalmente definida

La gramática **G=(V,Σ,R,S)**, donde:

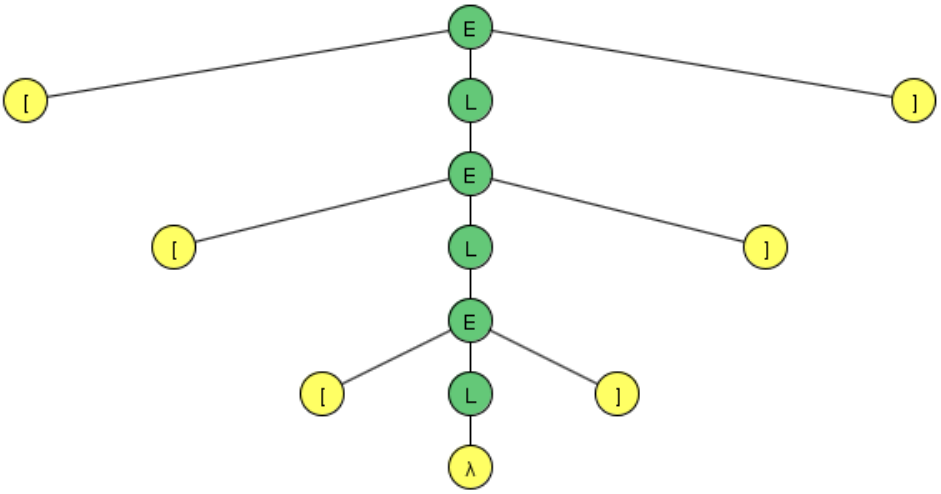
- **V** (variables o no terminales): E, L
- **Σ** (alfabeto terminal): {[,],0, 1, 2, 3, 4, 5, 6, 7,8 ,9}
- **R:** Ya mencionadas
- **S** (símbolo inicial): E

Cadenas Aceptadas:

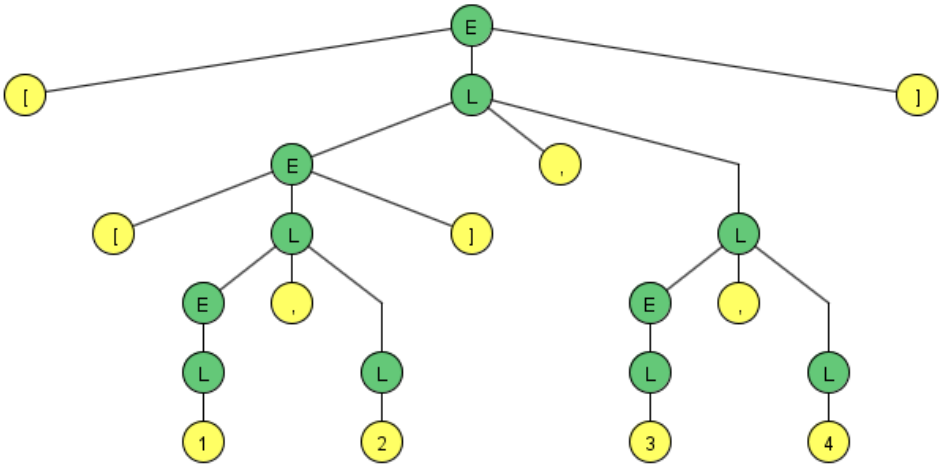
1. []



2. [[[]]]



3. [[1,2],3,4]



Modificacion:

Diseñar una gramática independiente del contexto que genere el lenguaje

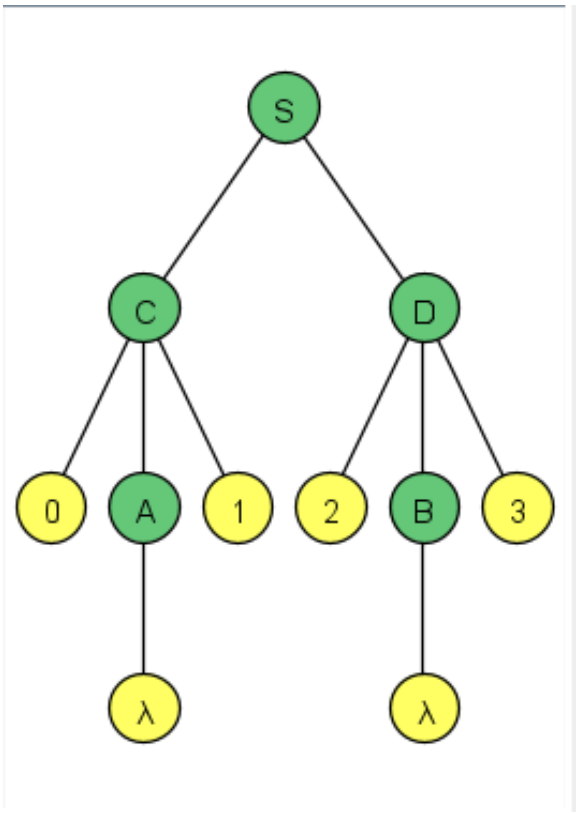
L = {0^n 1^n 2^m 3m | n + m es par }

Explicación: El símbolo inicial S combina dos subgramáticas: una (A y C) que genera la parte 0^n1^n controlando si n es par o impar, y otra (B y D) que genera la parte 2^m3^m controlando la paridad de m. De este modo, S → AB produce casos en los que n y m son ambos pares, mientras que S → CD produce casos en los que ambos son impares, asegurando que la suma n+m sea siempre par.

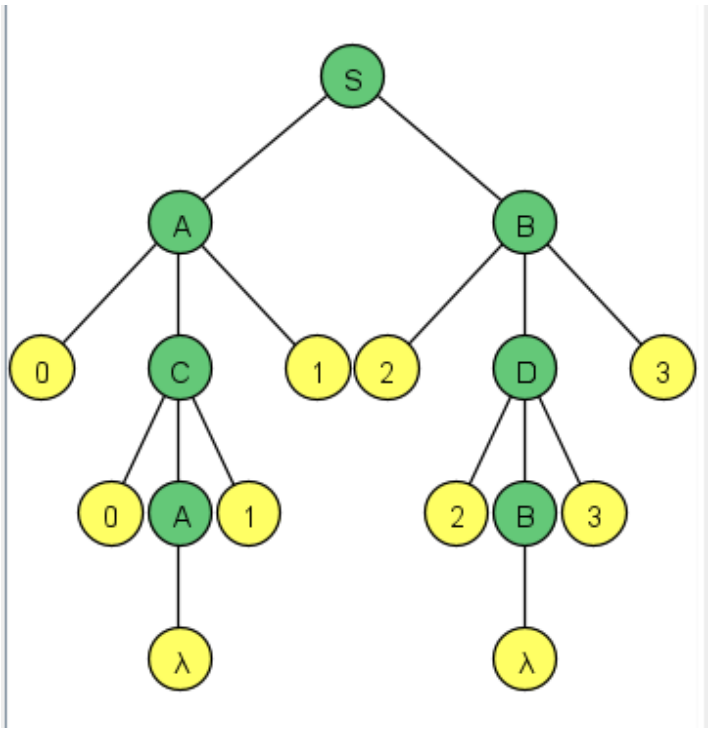
LHS		RHS
S	→	AB
S	→	CD
A	→	0C1
A	→	λ
C	→	0A1
B	→	2D3
B	→	λ
D	→	2B3

Cadenas:

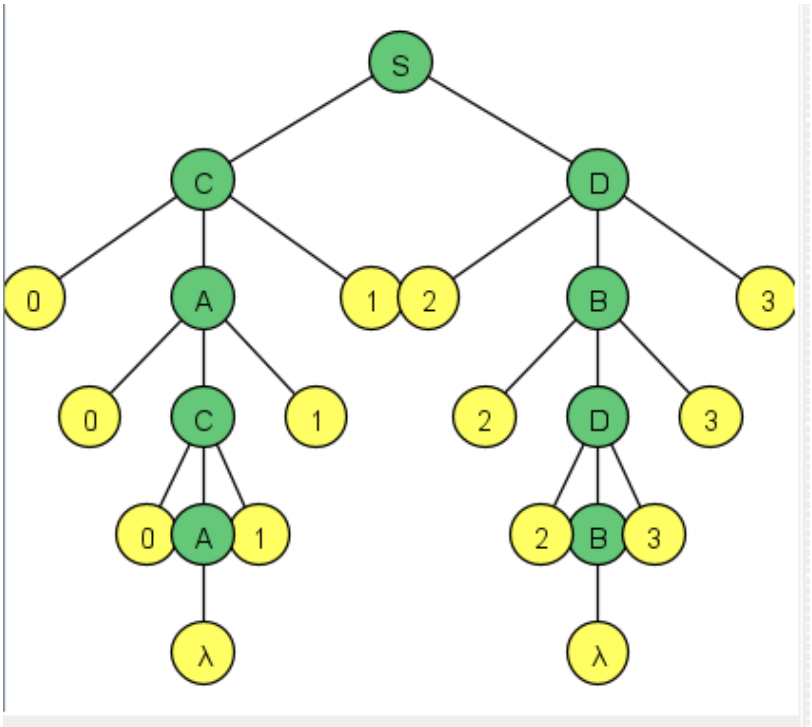
1. 0123



2. 00112233



3. 000111222333



Proceso de simplificación de gramática:

Paso 1 → Eliminar producciones vacías (ε)

No terminales anulables: A, B.

De S -> AB añadimos S -> A, S -> B, y S -> ε.

También añadimos alternativas sin los anulables en C y D.

Paso 2 → Eliminar producciones unitarias

Sustituimos las unitarias S -> A y S -> B por sus producciones no unitarias.

Paso 3 → Eliminar símbolos y producciones inútiles
Todos los no terminales generan terminales y son alcanzables desde S.

Etapa 1: Generadores de cadenas:
V' = {S, A, B, C, D}

Etapa 2: Alcanzables:
Σ = {0, 1, 2, 3} → Σ' = {0, 1, 2, 3}

Gramática simplificada:

LHS		RHS
S	→	AB
S	→	CD
A	→	0C1
C	→	0A1
B	→	2D3
D	→	2B3
S	→	A
S	→	B
C	→	01
D	→	23