Facultad de Economía,
Empresa y Turismo
Universidad de La Laguna

Práctica #07: Gramáticas en JFLAP:

Computabilidad y Algoritmia.

30/10/2024

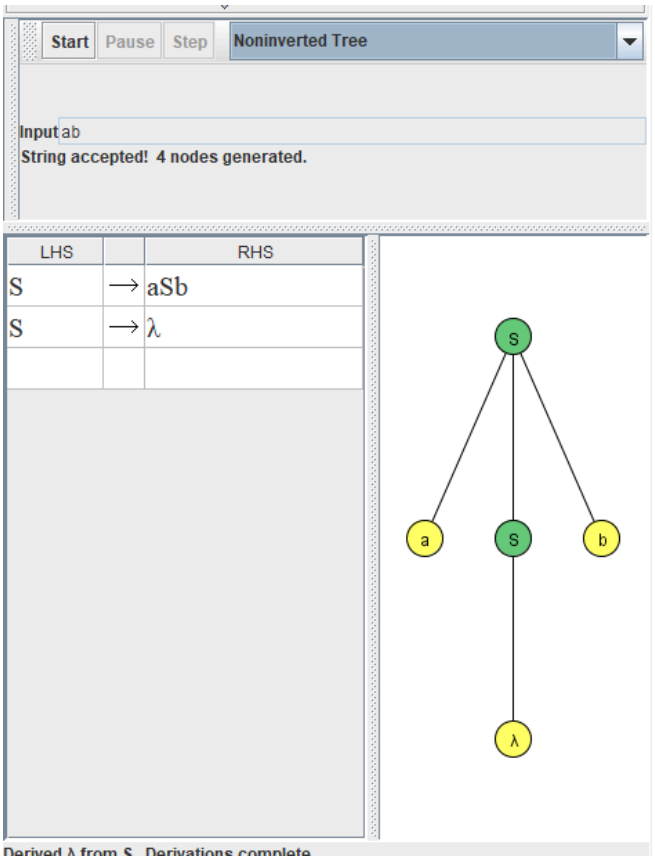
Pablo García De Los Reyes

Ejercicios prácticos:

1. Diseñar una gramática independiente del contexto que genere el lenguaje L = {a^n b^n | n ≥ 0}

   La gramática acepta cadenas con el mismo número de aes seguido del mismo número de bes, por lo que aceptamos cualquier cadena que empiece en a y acabe en b, incluyendo la cadena vacía. Todo esto producido por S.

LHS		
S	⟶	aSb
S	⟶	λ



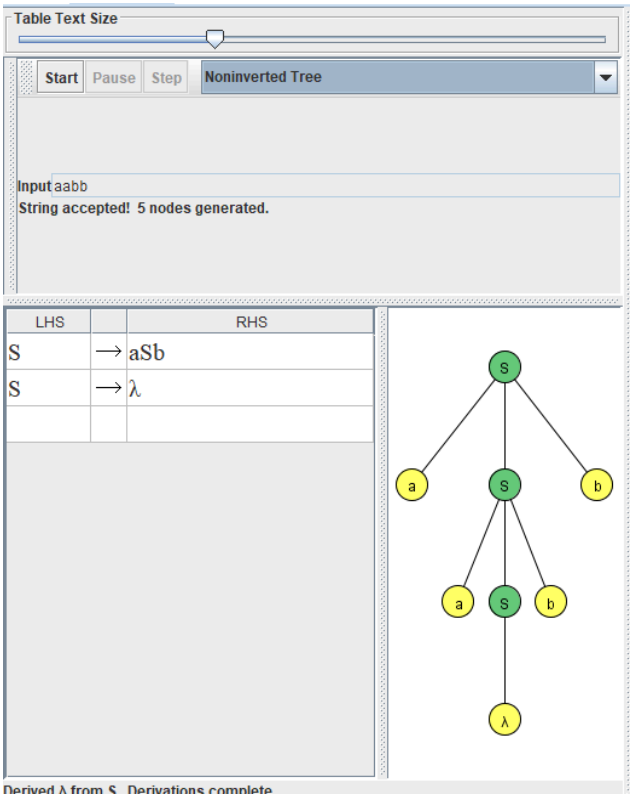
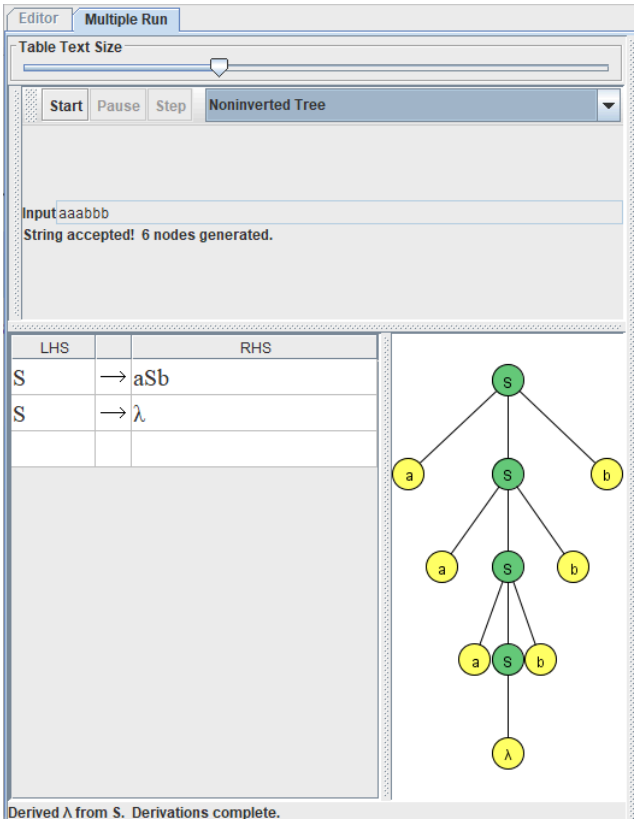
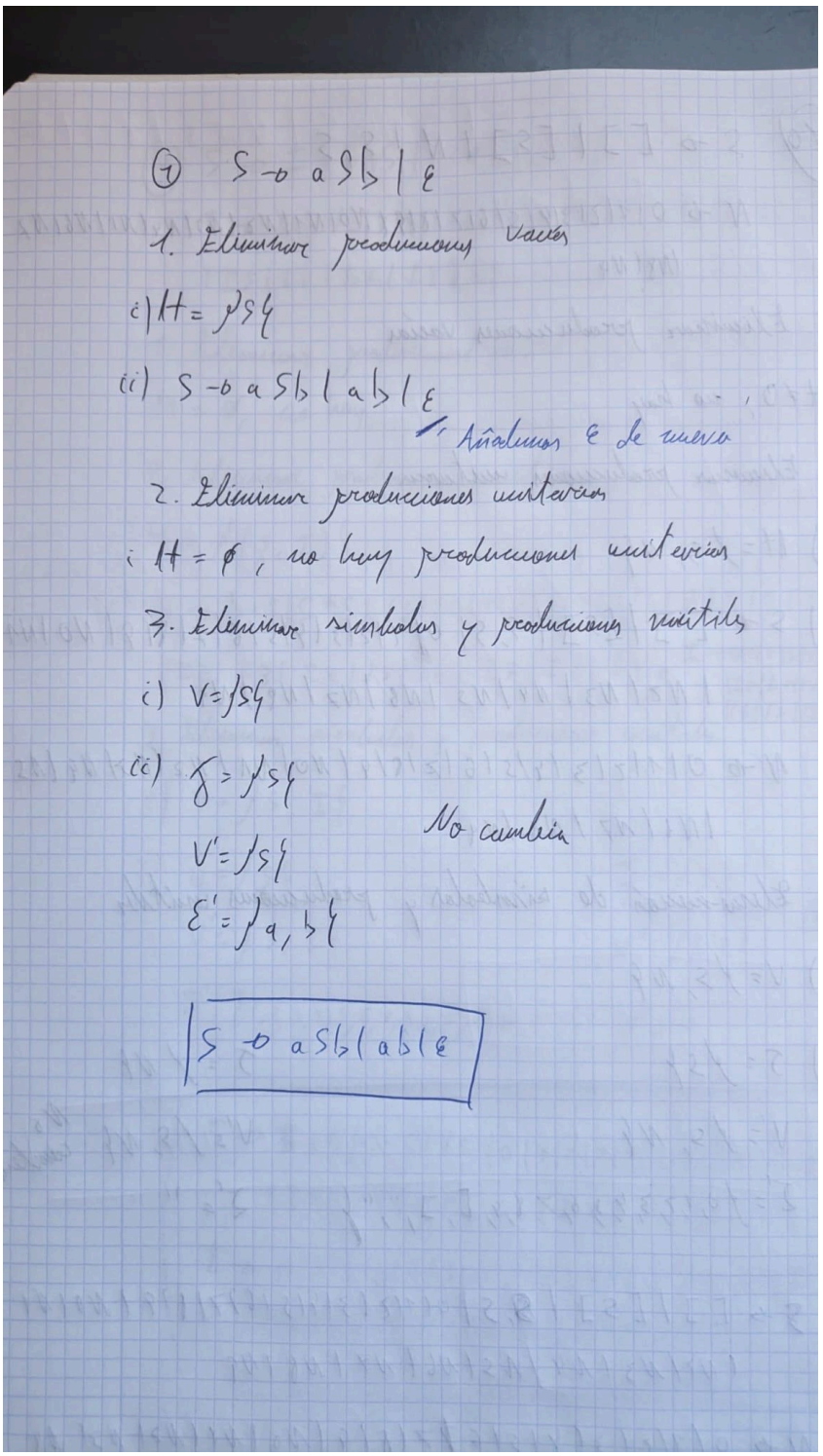


Table Text Size
Start
Pause
Step
Noninverted Tree
Input aabb
String accepted! 5 nodes generated.
LHS
RHS
S → aSb
S → λ
S
a
S
b
a
S
b
λ
Derived λ from S. Derivations complete


Editor
Multiple Run
Table Text Size
Start
Pause
Step
Noninverted Tree
Input aaabbb
String accepted! 6 nodes generated.
LHS
RHS
S → aSb
S → λ
S
a
S
b
a
S
b
a
S
b
λ
Derived λ from S. Derivations complete.



① $S \to aSb \mid \varepsilon$

1. Eliminar producciones vacías

i) $H = \{S\}$

ii) $S \to aSb \mid ab \mid \varepsilon$

Añadimos ε de nuevo

2. Eliminar producciones unitarias

$H = \emptyset$, no hay producciones unitarias

3. Eliminar símbolos y producciones inútiles

i) $V = \{S\}$

ii) $\delta = \{S\}$

No cambia

$V' = \{S\}$

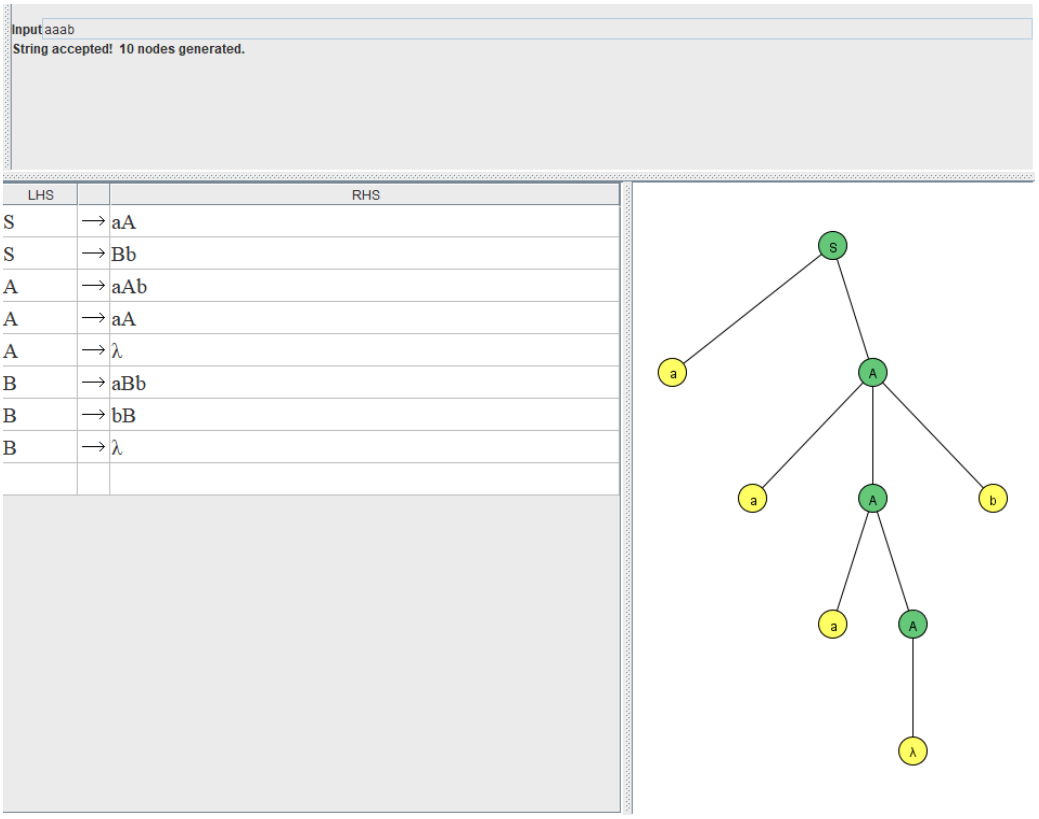
$\Sigma' = \{a, b\}$

$S \to aSb \mid ab \mid \varepsilon$

2. Diseñar una gramática independiente del contexto que genere el lenguaje L = {a n b m | n, m ≥ 0, n ≠ m}

La gramática genera el lenguaje al que pertenecen cadenas de aes seguidas de bes, siendo el número de aes distinto que el de bes.
El símbolo no terminal S se encarga de contemplar dos opciones, que la cadena empiece en a o que acabe en b, para las que empiezan en a, mediante el no terminal A contempla cadenas con mayor número de aes que de bes, sin embargo el no terminal B contempla la opción de mayor número de bes que de aes.

LHS		RHS
S	→	aA
S	→	Bb
A	→	aAb
A	→	aA
A	→	λ
B	→	aBb
B	→	bB
B	→	λ



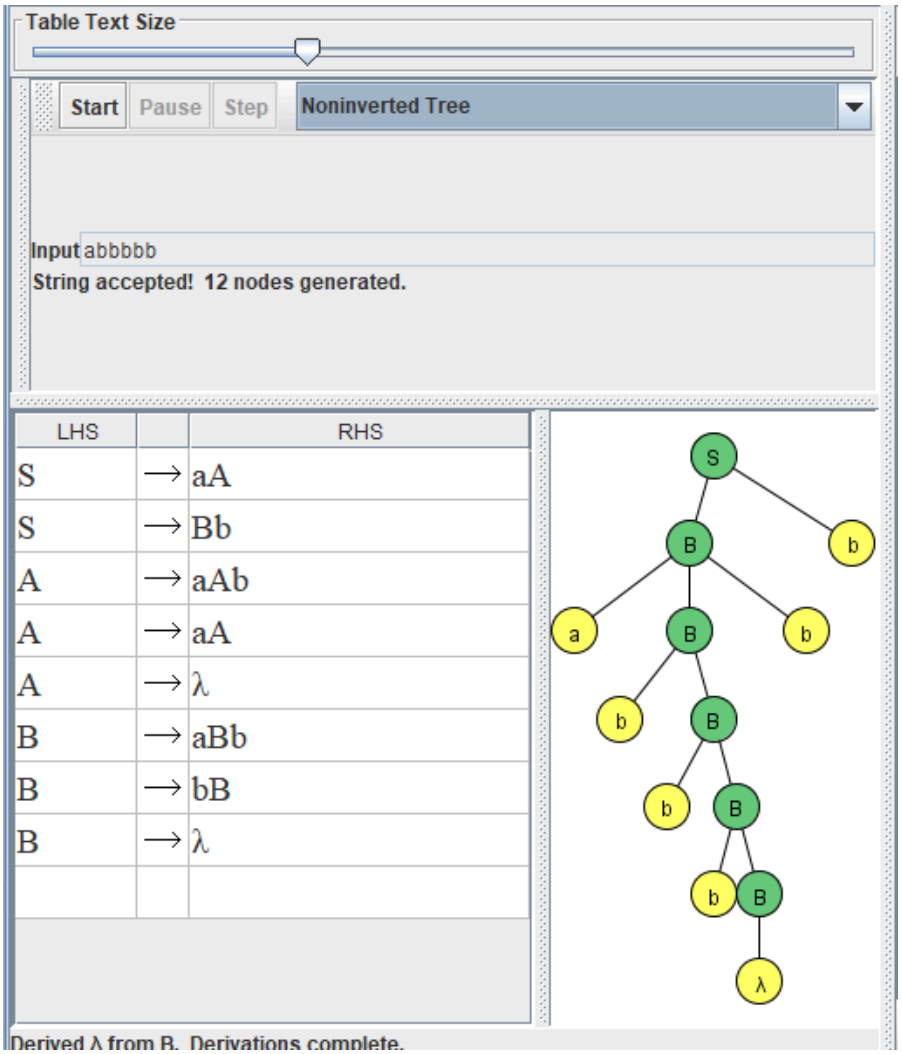
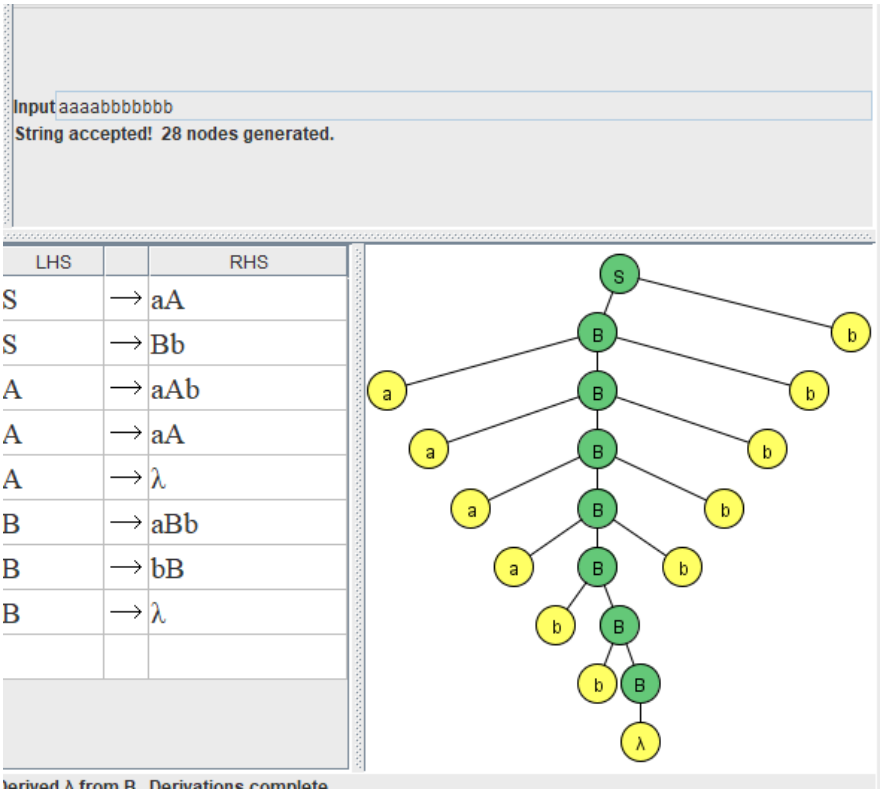
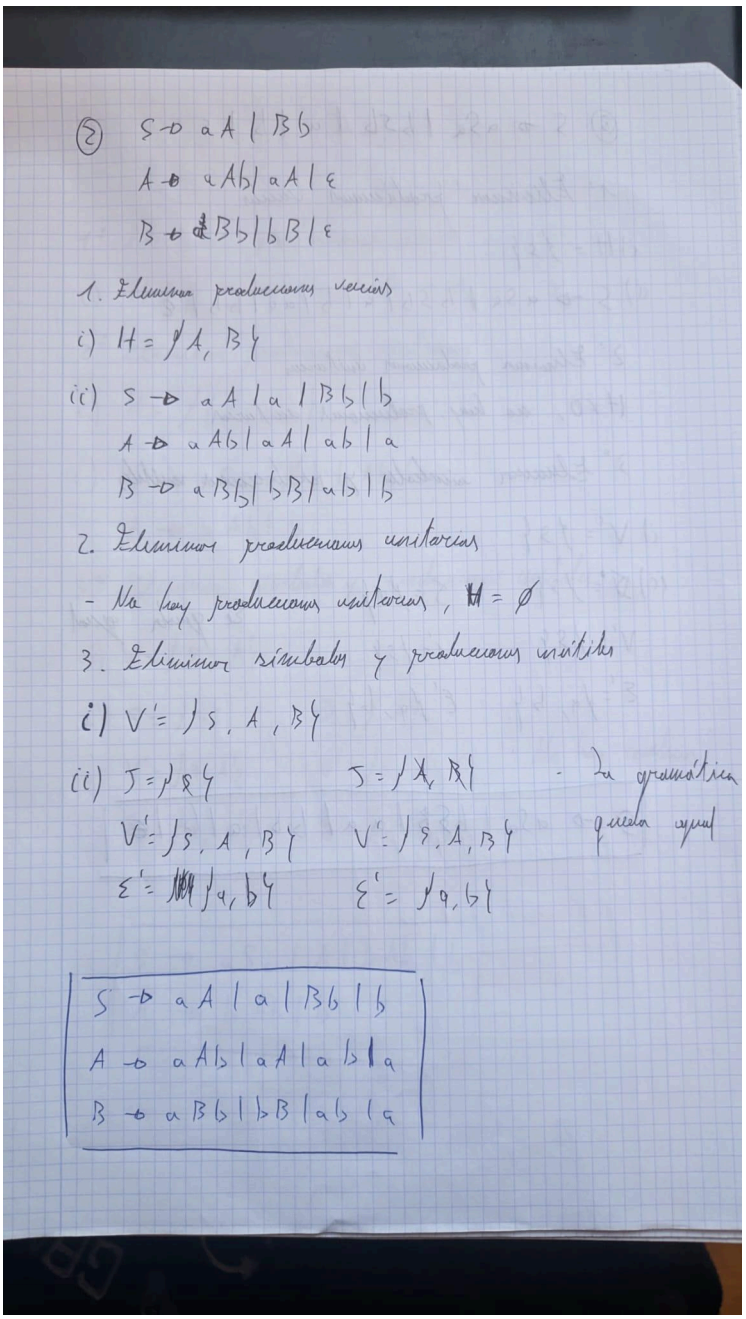


Table Text Size
Start
Pause
Step
Noninverted Tree
Input abbbbb
String accepted! 12 nodes generated.
LHS
RHS
S → aA
S → Bb
A → aAb
A → aA
A → λ
B → aBb
B → bB
B → λ
S
B
b
a
B
b
b
B
b
B
b
B
λ
Derived λ from B. Derivations complete.


Input aaaabbbbbbb
String accepted! 28 nodes generated.
LHS
RHS
S → aA
S → Bb
A → aAb
A → aA
A → λ
B → aBb
B → bB
B → λ
S
B
a
b
λ



② $S \to aA \mid Bb$

$A \to aAb \mid aA \mid \varepsilon$

$B \to aBb \mid bB \mid \varepsilon$

1. Eliminar producciones vacías

i) $H = \{A, B\}$

ii) $S \to aA \mid a \mid Bb \mid b$

$A \to aAb \mid aA \mid ab \mid a$

$B \to aBb \mid bB \mid ab \mid b$

2. Eliminar producciones unitarias

- No hay producciones unitarias, $H = \emptyset$

3. Eliminar símbolos y producciones inútiles

i) $V' = \{S, A, B\}$

ii) $T = \{S\}$ $\quad S = \{A, B\}$ - La gramática queda igual

$V' = \{S, A, B\}$ $\quad V' = \{S, A, B\}$

$\Sigma' = \{a, b\}$ $\quad \Sigma' = \{a, b\}$

$S \to aA \mid a \mid Bb \mid b$

$A \to aAb \mid aA \mid ab \mid a$

$B \to aBb \mid bB \mid ab \mid a$

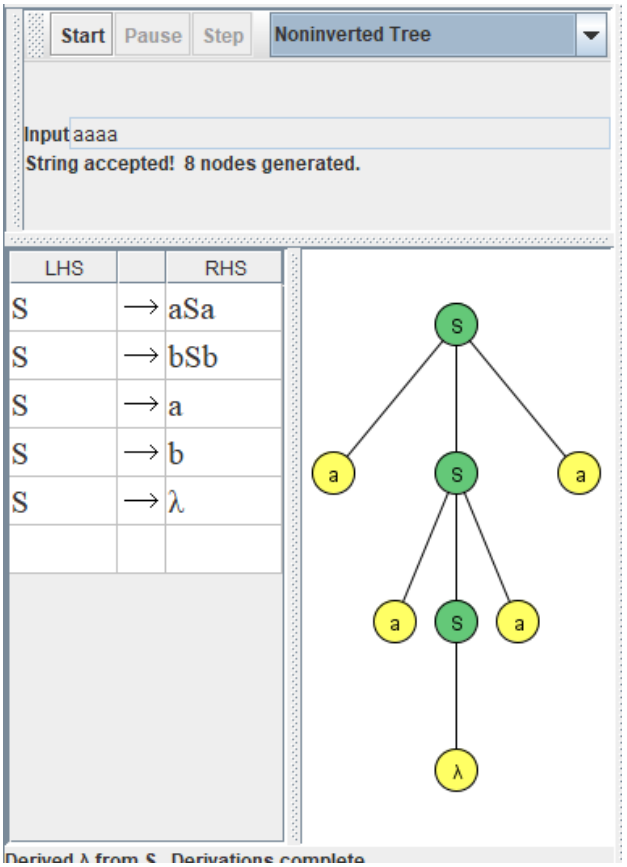
LHS		
S	→	aA
S	→	Bb
A	→	aAb
A	→	aA
B	→	aBb
B	→	bB
S	→	a
S	→	b
A	→	ab
A	→	a
B	→	ab
B	→	b

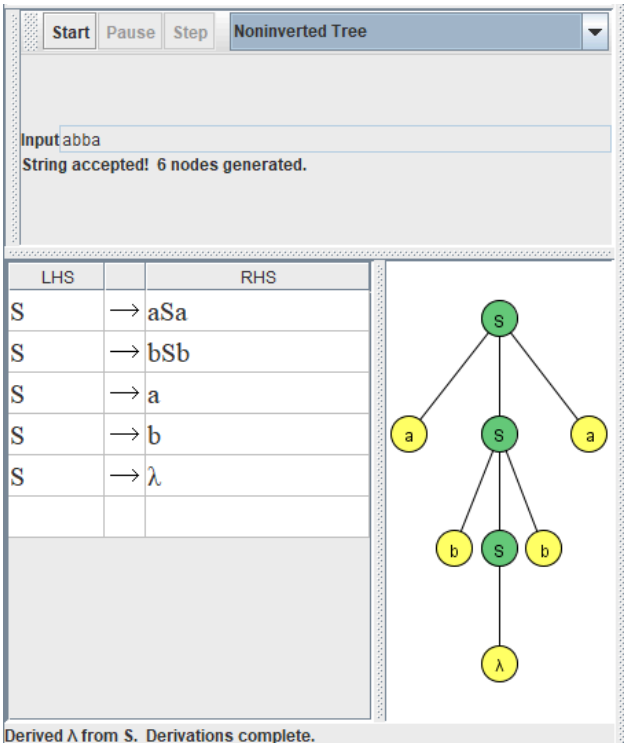
3. Diseñar una gramática independiente del contexto para el lenguaje L = {wwI | w ∈ {a, b} ∗ }

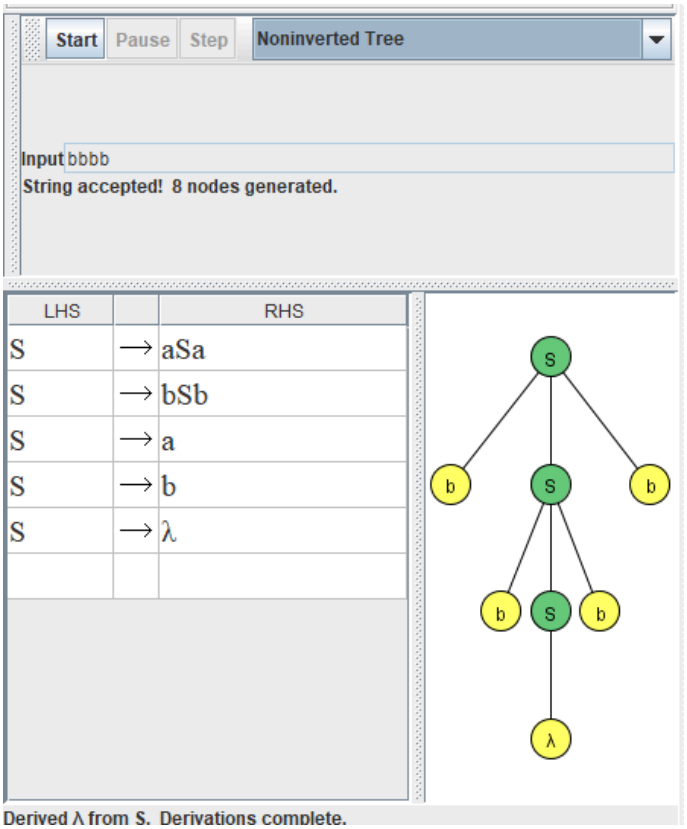
Esta gramática genera el lenguaje que acepta las cadenas cuya w vendrá seguida de su inversa, para realizar el diseño construimos la gramática desde dentro, para así poder utilizar únicamente el símbolo no terminal S cuyas producciones son siempre de la manera xYx que nos aseguran tener la inversa de la cadena construida desde dentro, sea de longitud par o impar, pues también contemplamos la cadena vacía.

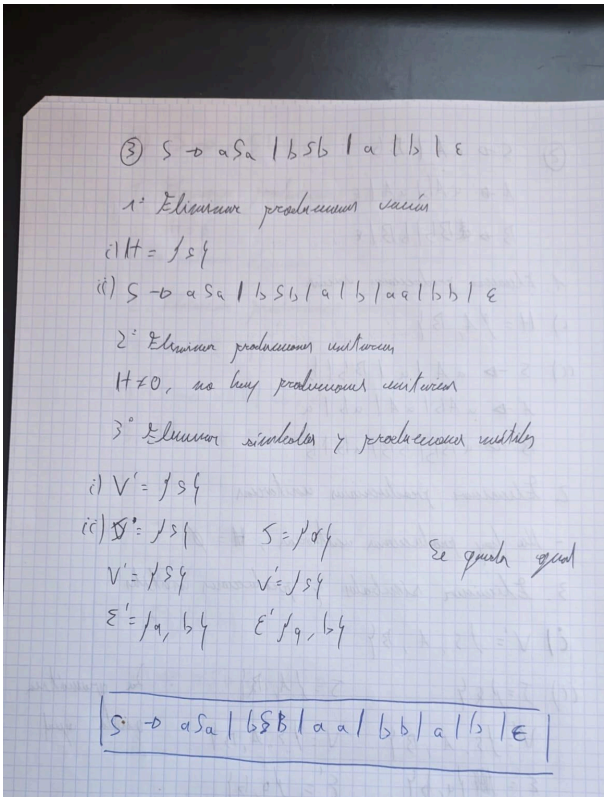
LHS		RHS
S	→	aSa
S	→	bSb
S	→	λ

*CORREGIDO EN PRÁCTICA


Start
Pause
Step
Noninverted Tree
Input aaaa
String accepted! 8 nodes generated.
LHS
RHS
S → aSa
S → bSb
S → a
S → b
S → λ
S
a
S
a
a
S
a
λ


Start
Pause
Step
Noninverted Tree
Input abba
String accepted! 6 nodes generated.
LHS
RHS
S → aSa
S → bSb
S → a
S → b
S → λ
S
a
S
a
b
S
b
λ
Derived λ from S. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input bbbb
String accepted! 8 nodes generated.
LHS
RHS
S → aSa
S → bSb
S → a
S → b
S → λ
S
b
S
b
b
S
b
λ
Derived λ from S. Derivations complete.

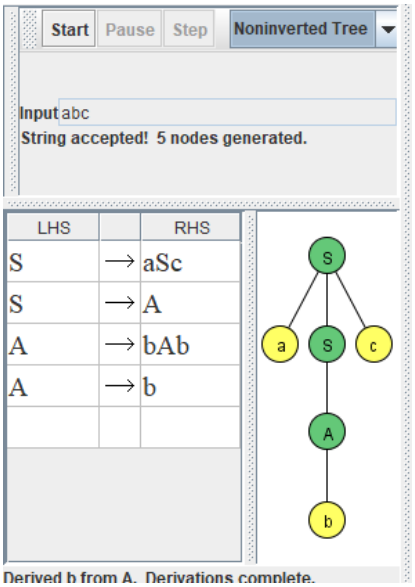

③ S → aSa | bSb | a | b | ε
1° Eliminar producciones vacías
i) H = {S}
ii) S → aSa | bSb | a | b | aa | bb | ε
2° Eliminar producciones unitarias
H ≠ ∅, no hay producciones unitarias
3° Eliminar símbolos y producciones inútiles
i) V' = {S}
ii) V' = {S}
V' = {S}
ε' = {a, b}
S = {S}
V' = {S}
ε' = {a, b}
Se queda igual
S → aSa | bSb | aa | bb | a | b | ε

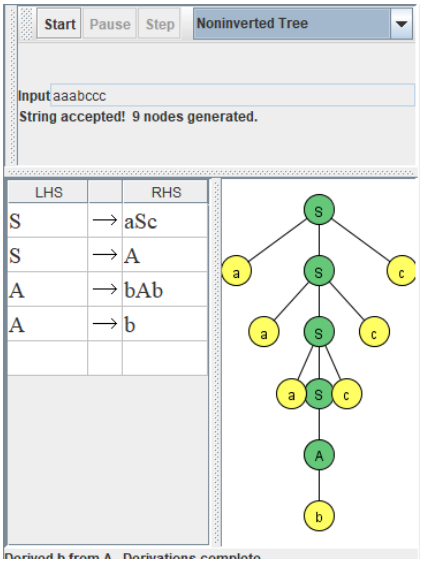
Welcome to the Chomsky converter. 4 production(s) must be converted.		
LHS		
S	→	aSa
S	→	bSb
S	→	a
S	→	b
S	→	aa
S	→	bb

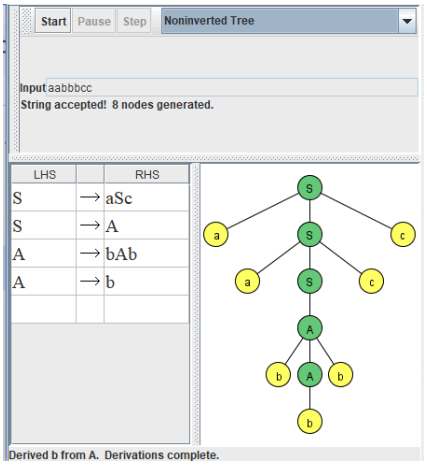
4. Diseñar una gramática independiente del contexto que genere el lenguaje L = {a n b mc n | n ≥ 0, m impar}

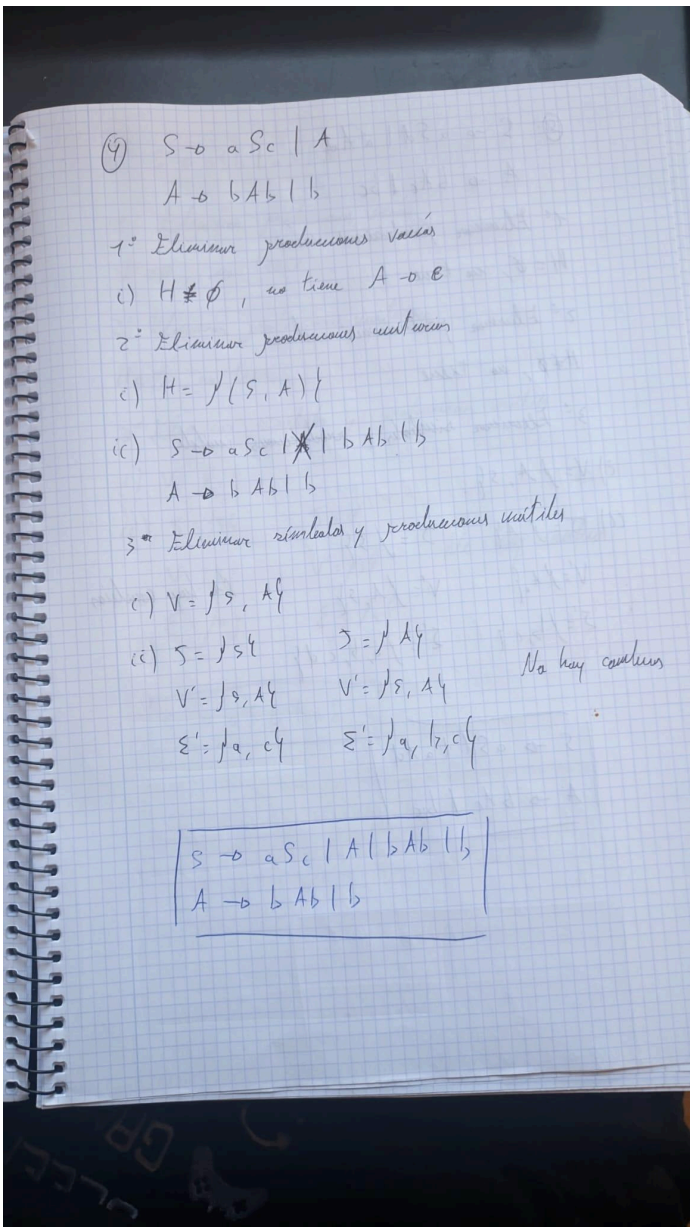
Esta gramática genera el lenguaje que acepta las cadenas de aes seguidas de cualquier número de bes seguidas de el mismo número de ces que de aes. Siendo m impar. Para realizar esta gramática utilizamos el símbolo no terminal de arranque S para determinar cómo va a ser nuestra cadena generada, en este caso será aSc, con una producción de S -> A para poder tener en medio de cualquier número de a y c, un número de b impar. De esto se encarga el no terminal A, que solo nos permite producir un número de b impar, ya sea una b, o tres, concatenadas entre sí para poder obtener cualquier longitud impar.

LHS		
S	→	aSc
S	→	A
A	→	bAb
A	→	b


Start
Pause
Step
Noninverted Tree
Input abc
String accepted! 5 nodes generated.
LHS
RHS
S → aSc
S → A
A → bAb
A → b
S
a
S
c
A
b
Derived b from A. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input aaabccc
String accepted! 9 nodes generated.
LHS
RHS
S → aSc
S → A
A → bAb
A → b


Start
Pause
Step
Noninverted Tree
Input aabbbcc
String accepted! 8 nodes generated.
LHS
RHS
S → aSc
S → A
A → bAb
A → b
Derived b from A. Derivations complete.

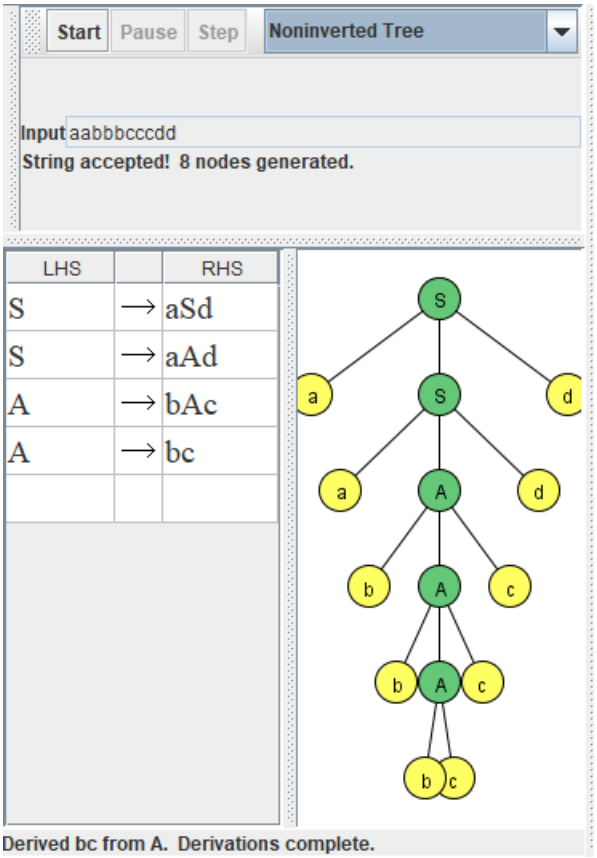


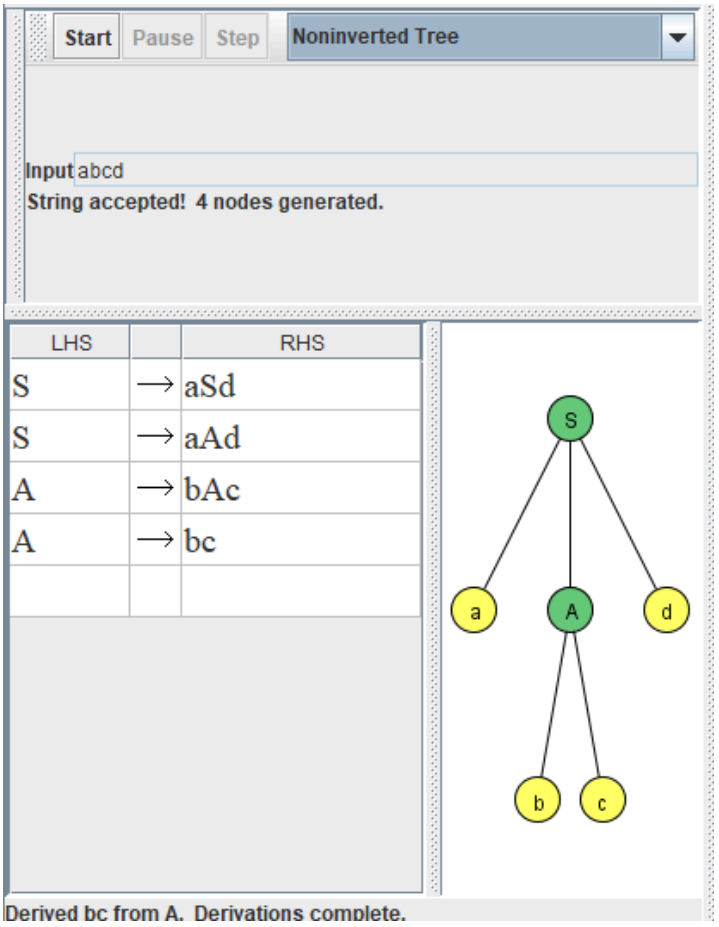
LHS		
S	→	aSc
S	→	bAb
S	→	b
A	→	b
A	→	bAb

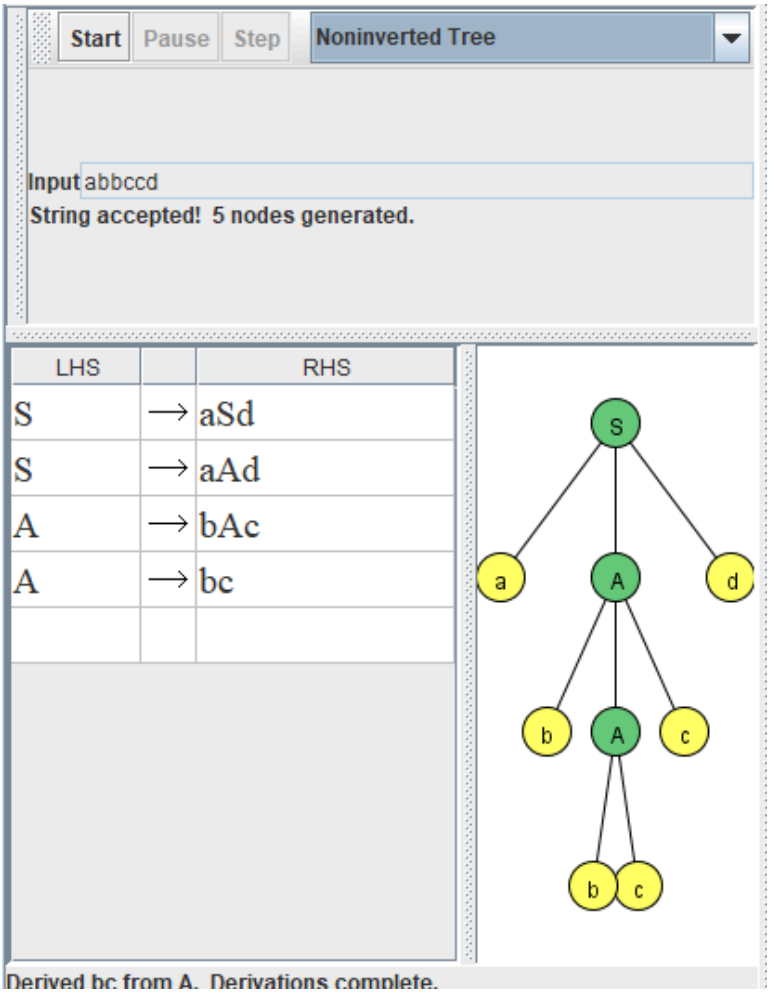
5. Diseñar una gramática independiente del contexto que genere el lenguaje L = {a^n b^m c^m d^n | n, m ≥ 1}

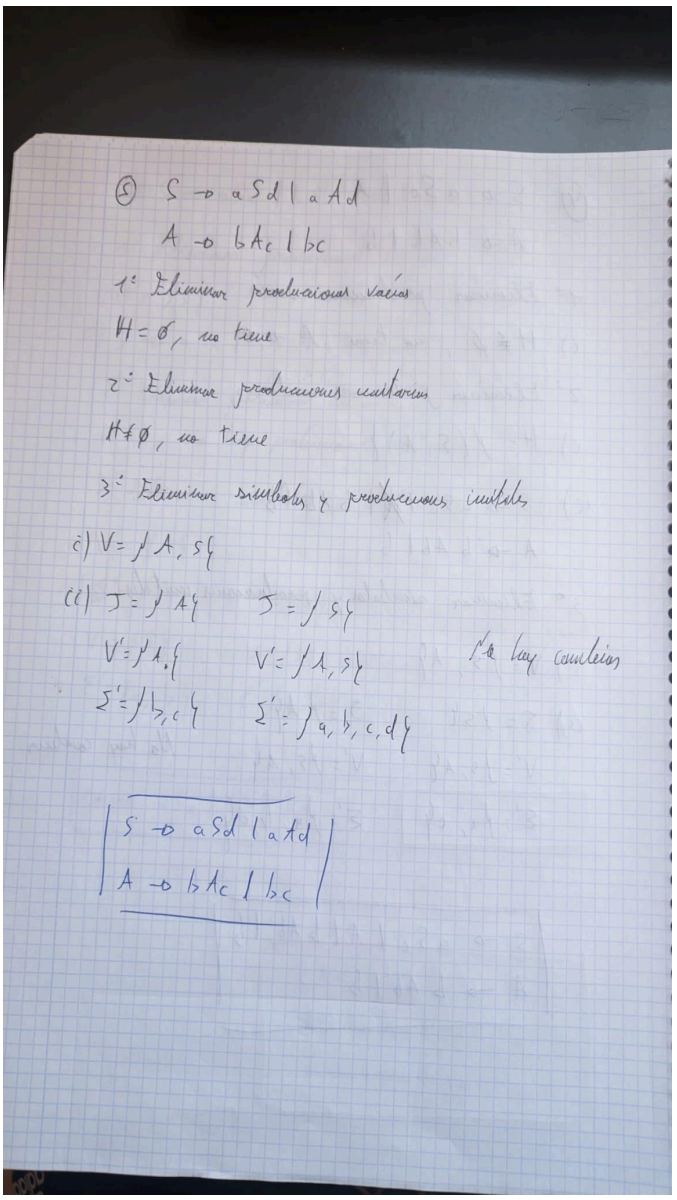
Esta gramática genera el lenguaje que acepta cadenas de aes seguidas del mismo número de bes que ces seguidas por el mismo número de des que de aes.
El símbolo no terminal S construye la gramática desde dentro, obligando a que haya el mismo número de aes que des mientras que el no terminal A asegura que haya el mismo número de bes que de ces entre las aes y las des.

LHS		
S	→	aSd
S	→	aAd
A	→	bAc
A	→	bc




Start
Pause
Step
Noninverted Tree
Input abcd
String accepted! 4 nodes generated.
LHS
RHS
S → aSd
S → aAd
A → bAc
A → bc
S
a
A
d
b
c
Derived bc from A. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input abbccd
String accepted! 5 nodes generated.
LHS
RHS
S → aSd
S → aAd
A → bAc
A → bc
S
a
A
d
b
A
c
b
c
Derived bc from A. Derivations complete.

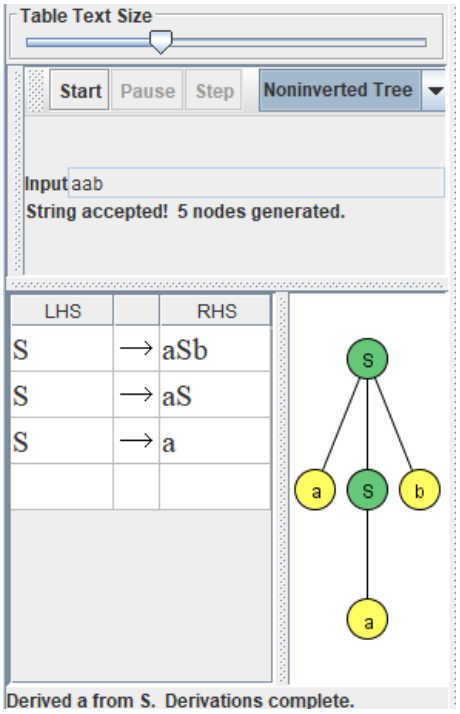


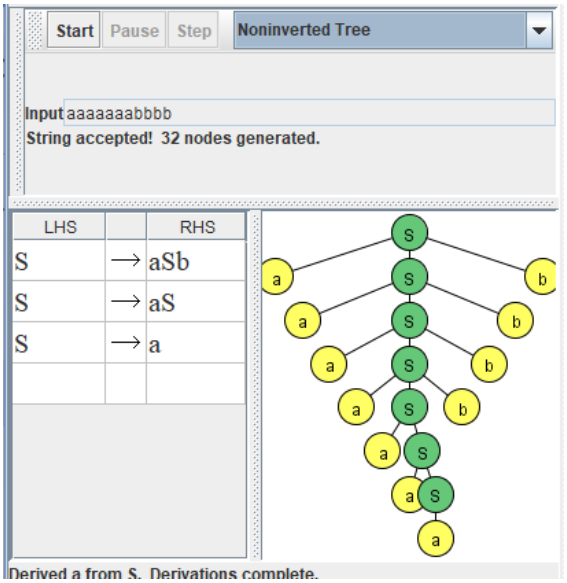
LHS		
S	→	aSd
S	→	aAd
A	→	bAc
A	→	bc

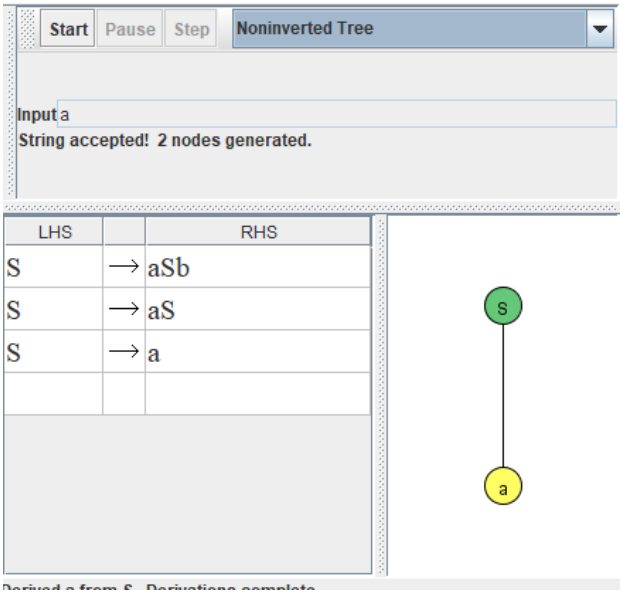
6. Diseñar una gramática independiente del contexto que genere el lenguaje L = {a^n b^m | n > m ≥ 0}

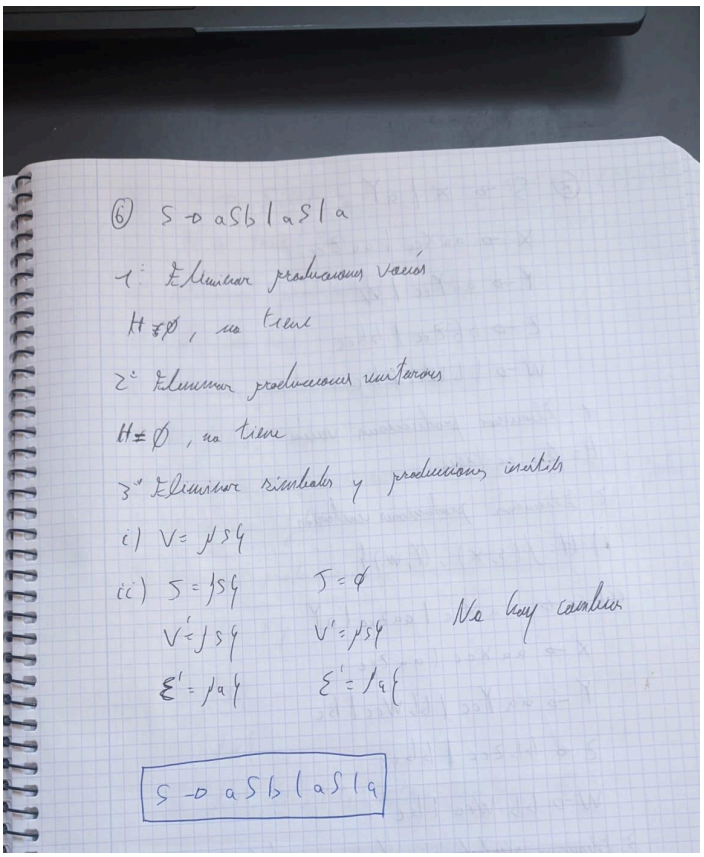
Esta gramática genera el lenguaje que acepta las cadena de aes seguidas de bes, habiendo más aes que bes. Para asegurar esto nos basta con usar el símbolo no terminal S que con sus producciones nos aseguramos de que haya al menos una a más que b.

LHS		
S	→	aSb
S	→	aS
S	→	a




Start
Pause
Step
Noninverted Tree
Input aaaaaaabbbb
String accepted! 32 nodes generated.
LHS
RHS
S → aSb
S → aS
S → a
Derived a from S. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input a
String accepted! 2 nodes generated.
LHS
RHS
S → aSb
S → aS
S → a

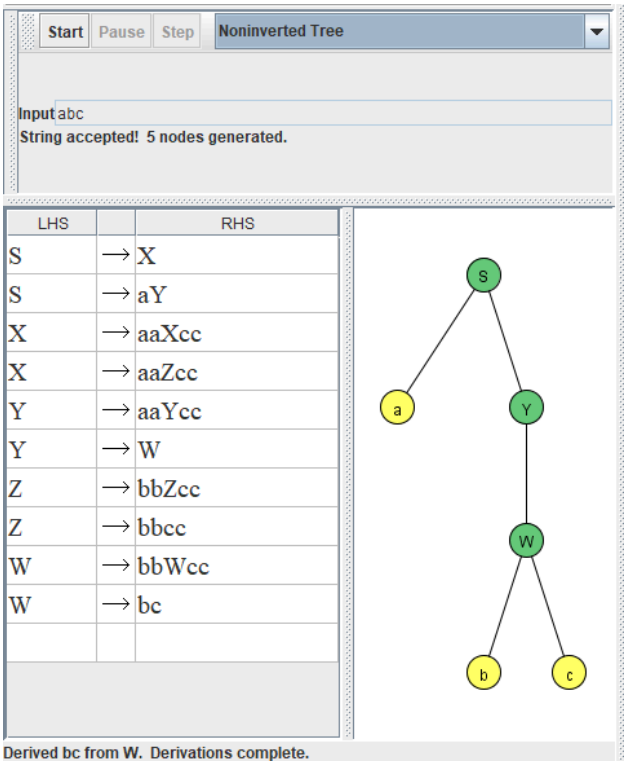


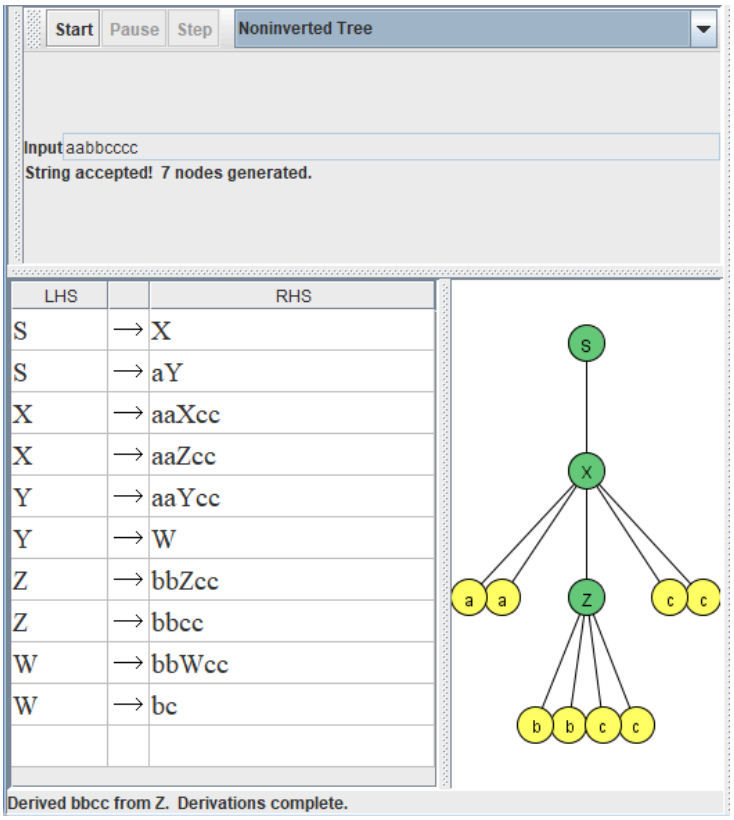
LHS		
S	→	aSb
S	→	aS
S	→	a

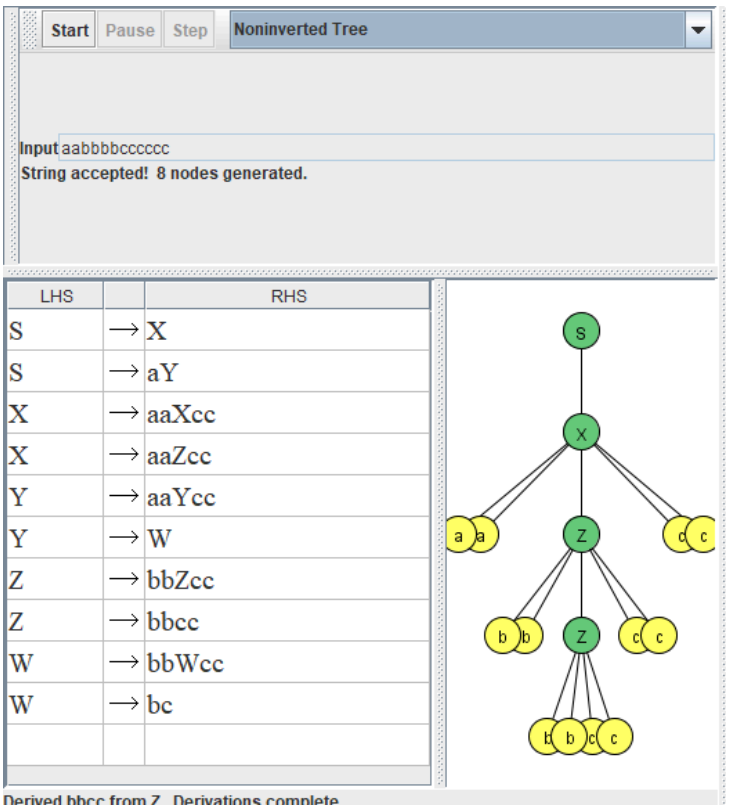
7. Dise˜nar una gramática independiente del contexto que genere el lenguaje L = {a^i b^j c^i+j | i, j ≥ 1, i + j par}

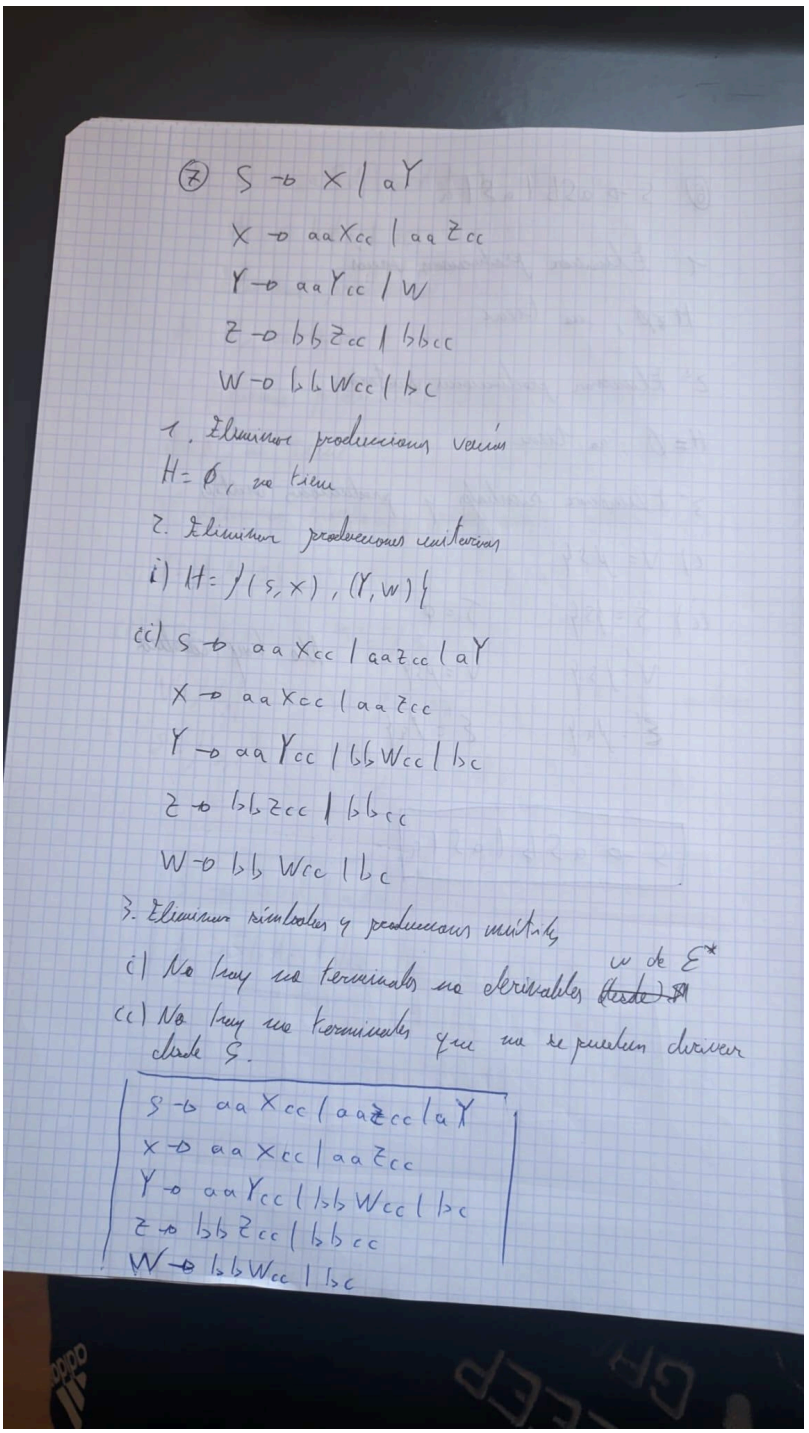
Esta gramática genera el lenguaje que acepta cadenas de aes seguidas de bes seguidas de el mismo número de ces que de la suma de aes y bes. Siendo esta suma de manera par. La complejidad de este diseño recae en la paridad de la suma de i y j. Para asegurar esto debemos tomar dos caminos, el primero en el que i y j son dos números pares, o en el que ambos son impares, para que su suma sea par. Para esto dividimos el problema con varios símbolos terminales. S elige el camino a tomar sobre la decisión comentada anteriormente, X asegura que la cadena empiece y acabe por número de a y c par. Y tiene esta misma función con el añadido de tener en cuenta que i y j pueden ser impares las dos. Z genera un número de b y c par, y W contempla la opción de la imparidad de i y j.

LHS		
S	→	X
S	→	aY
X	→	aaXcc
X	→	aaZcc
Y	→	aaYcc
Y	→	W
Z	→	bbZcc
Z	→	bbcc
W	→	bbWcc
W	→	bc


Start
Pause
Step
Noninverted Tree
Input abc
String accepted! 5 nodes generated.
LHS
RHS
S → X
S → aY
X → aaXcc
X → aaZcc
Y → aaYcc
Y → W
Z → bbZcc
Z → bbcc
W → bbWcc
W → bc
S
a
Y
W
b
c
Derived bc from W. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input aabbcccc
String accepted! 7 nodes generated.
LHS
RHS
S → X
S → aY
X → aaXcc
X → aaZcc
Y → aaYcc
Y → W
Z → bbZcc
Z → bbcc
W → bbWcc
W → bc
S
X
a
a
Z
c
c
b
b
c
c
Derived bbcc from Z. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input aabbbbcccccc
String accepted! 8 nodes generated.
LHS
RHS
S → X
S → aY
X → aaXcc
X → aaZcc
Y → aaYcc
Y → W
Z → bbZcc
Z → bbcc
W → bbWcc
W → bc
S
X
a a
Z
c c
b b
Z
c c
b b c c
Derived bbcc from Z. Derivations complete.



⑦ $S \to X \mid aY$

$X \to aaXcc \mid aaZcc$

$Y \to aaYcc \mid W$

$Z \to bbZcc \mid bbcc$

$W \to bbWcc \mid bc$

1. Eliminar producciones vacías

$H = \emptyset$, no tiene

2. Eliminar producciones unitarias

i) $H = \{(S, X), (Y, W)\}$

ii) $S \to aaXcc \mid aaZcc \mid aY$

$X \to aaXcc \mid aaZcc$

$Y \to aaYcc \mid bbWcc \mid bc$

$Z \to bbZcc \mid bbcc$

$W \to bbWcc \mid bc$

3. Eliminar símbolos y producciones inútiles

i) No hay ni terminales ni derivables ~~desde~~ w de Σ^*

ii) No hay ni terminales que no se puedan derivar desde S.

$S \to aaXcc \mid aaZcc \mid aY$

$X \to aaXcc \mid aaZcc$

$Y \to aaYcc \mid bbWcc \mid bc$

$Z \to bbZcc \mid bbcc$

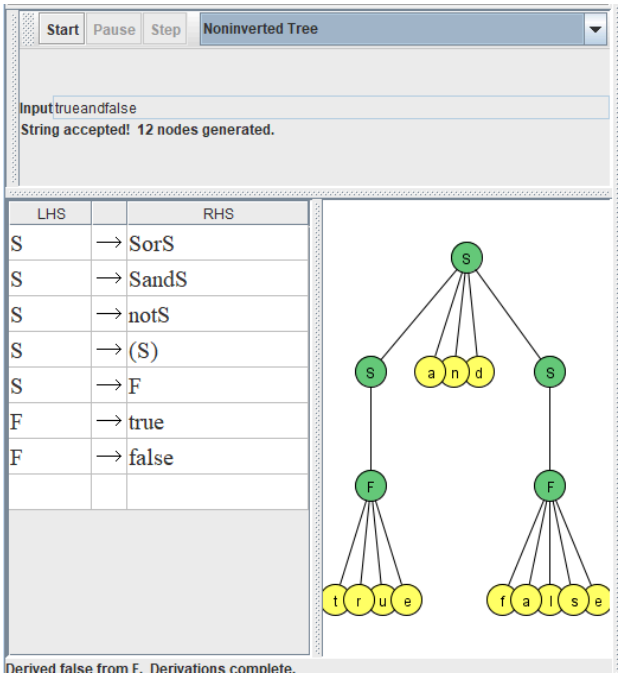
$W \to bbWcc \mid bc$

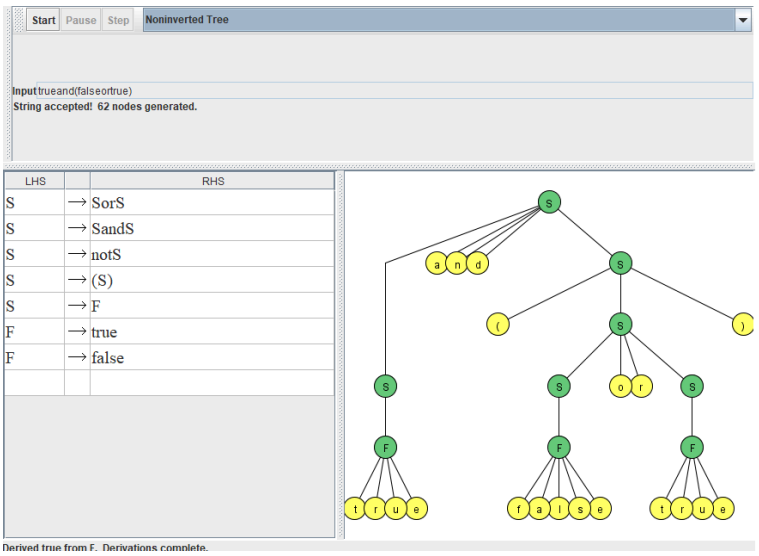
LHS		
S	→	aY
S	→	aaZcc
S	→	aaXcc
Y	→	bbWcc
Y	→	bc
W	→	bc
W	→	bbWcc
Z	→	bbcc
Z	→	bbZcc
Y	→	aaYcc
X	→	aaZcc
X	→	aaXcc

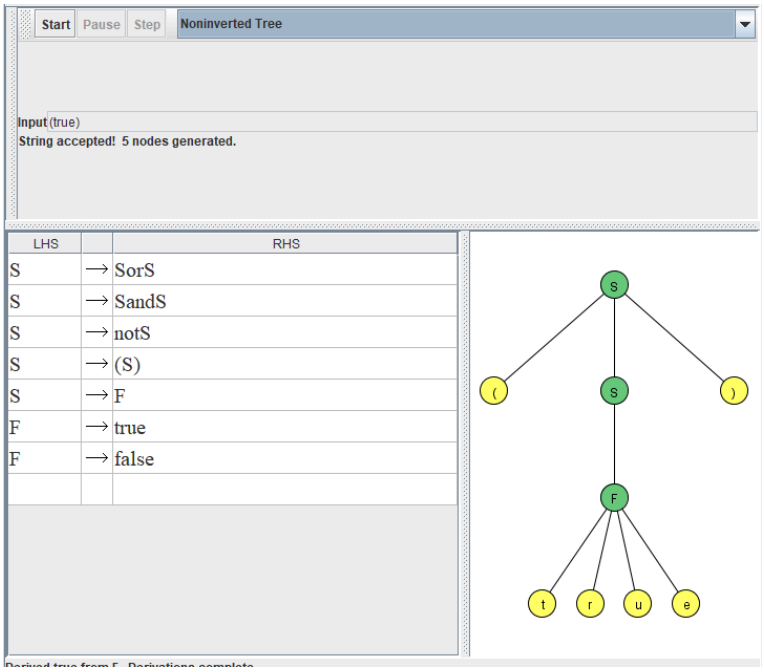
8. Diseñar una gramática independiente del contexto que genere el lenguaje de las expresiones booleanas con los operadores AND, OR, y NOT, usando paréntesis para agrupar. Ejemplos: “(true AND false)”, “NOT (true OR false)”, “true AND (false OR true)”.

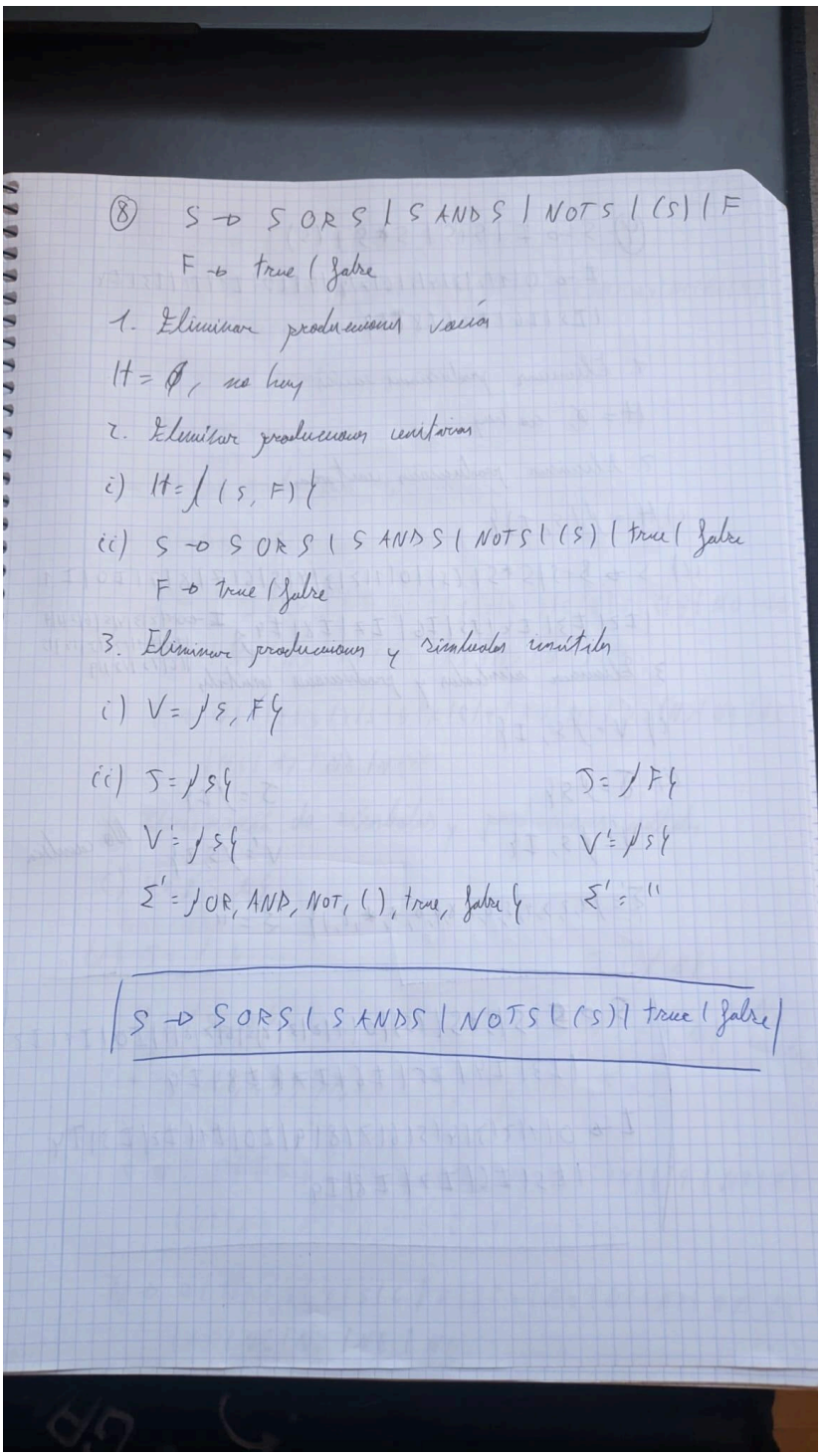
Esta gramática genera el lenguaje que acepta las operaciones lógicas con los valores booleanos true y false. Para realizar esto S tendrá en cuenta los operadores lógicos mientras que F contendrá los valores con los que operaremos en S.

LHS		
S	→	SorS
S	→	SandS
S	→	notS
S	→	(S)
S	→	F
F	→	true
F	→	false




Start
Pause
Step
Noninverted Tree
Input trueand(falseortrue)
String accepted! 62 nodes generated.
LHS
RHS
S → SorS
S → SandS
S → notS
S → (S)
S → F
F → true
F → false
Derived true from F. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input (true)
String accepted! 5 nodes generated.
LHS
RHS
S → SorS
S → SandS
S → notS
S → (S)
S → F
F → true
F → false



⑧ S → S OR S | S AND S | NOT S | (S) | F

F → true | false

1. Eliminar producciones vacías

$H = \emptyset$, no hay

2. Eliminar producciones unitarias

i) $H = \{(S, F)\}$

ii) S → S OR S | S AND S | NOT S | (S) | true | false

F → true | false

3. Eliminar producciones y símbolos unitarios

i) $V = \{S, F\}$

ii) $S = \{S\}$ $S = \{F\}$

$V = \{S\}$ $V = \{S\}$

$\Sigma' = \{OR, AND, NOT, (), true, false\}$ $\Sigma' = \{\}$

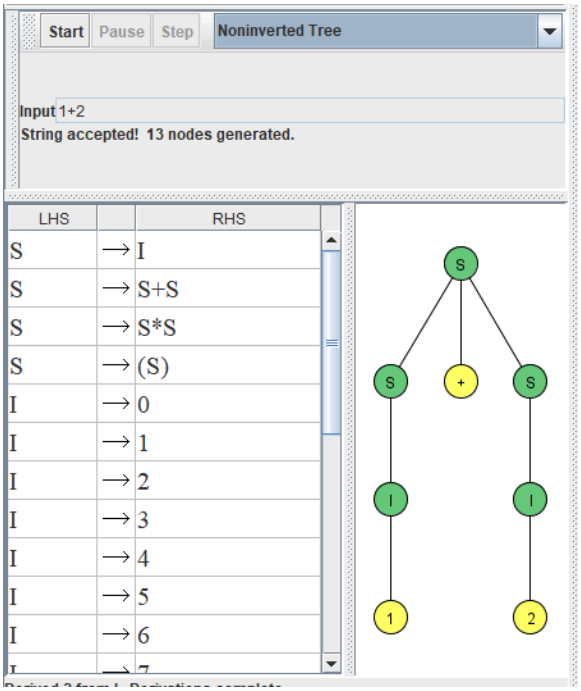
S → S OR S | S AND S | NOT S | (S) | true | false

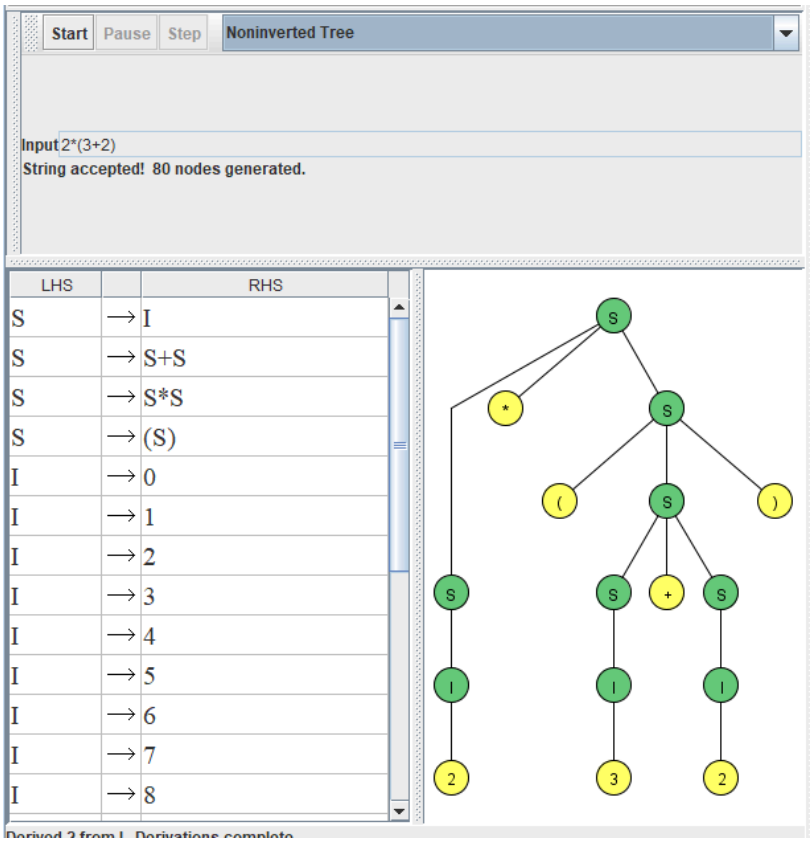
LHS		
S	→	SorS
S	→	SandS
S	→	notS
S	→	(S)
S	→	true
S	→	false

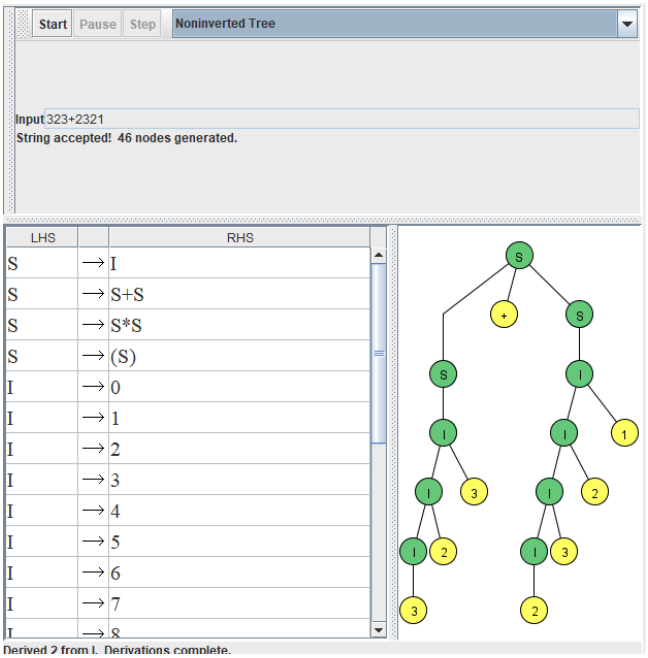
9. Diseñar una gramática independiente del contexto que genere expresiones aritméticas simples con suma y multiplicación, utilizando los s´ımbolos +, ∗, (,) y los n´umeros 0, 1, etc. Ejemplos: “1+2”, “(1+2)*3”, “4*(5+6)”.

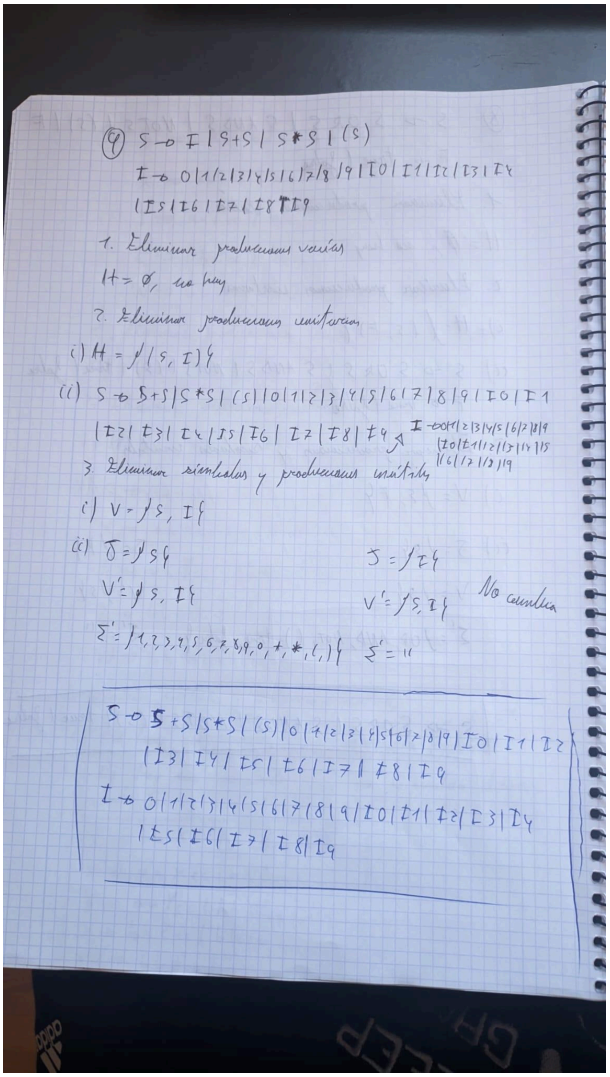
Para diseñar esta gramática aplicamos la misma lógica que en la anterior, solo que esta vez tenemos el no terminal I que tendrá en cuenta a cualquier número natural.

LHS		
S	→	I
S	→	S+S
S	→	S*S
S	→	(S)
I	→	0
I	→	1
I	→	2
I	→	3
I	→	4
I	→	5
I	→	6
I	→	7
I	→	8
I	→	9
I	→	I0
I	→	I1
I	→	I2
I	→	I3
I	→	I4
I	→	I5
I	→	I6
I	→	I7
I	→	I8
I	→	I9


Start
Pause
Step
Noninverted Tree
Input 1+2
String accepted! 13 nodes generated.
LHS
RHS
S → I
S → S+S
S → S*S
S → (S)
I → 0
I → 1
I → 2
I → 3
I → 4
I → 5
I → 6


Start
Pause
Step
Noninverted Tree
Input 2*(3+2)
String accepted! 80 nodes generated.
LHS
RHS
S → I
S → S+S
S → S*S
S → (S)
I → 0
I → 1
I → 2
I → 3
I → 4
I → 5
I → 6
I → 7
I → 8


Start
Pause
Step
Noninverted Tree
Input 323+2321
String accepted! 46 nodes generated.
LHS
RHS
S → I
S → S+S
S → S*S
S → (S)
I → 0
I → 1
I → 2
I → 3
I → 4
I → 5
I → 6
I → 7
Derived 2 from I. Derivations complete.



LHS		RHS
S	→	S+S
S	→	S*S
S	→	(S)
I	→	0
I	→	1
I	→	2
I	→	3
I	→	4
I	→	5
I	→	6
I	→	7
I	→	8
I	→	9
I	→	I0
I	→	I1
I	→	I2
I	→	I3
I	→	I4
I	→	I5
I	→	I6
I	→	I7
I	→	I8
I	→	I9

S	→	I9
S	→	I0
S	→	I4
S	→	I3
S	→	I2
S	→	I1
S	→	I8
S	→	I7
S	→	I6
S	→	I5
S	→	3
S	→	2
S	→	1
S	→	0
S	→	7
S	→	6
S	→	5
S	→	4
S	→	9
S	→	8

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]].

Para esta gramática aplicamos la misma metodología que en las dos anteriores, con la única diferencia que esta vez será con corchetes en lugar de paréntesis, y comas en lugar de operadores.

LHS		
S	→	[]
S	→	[S]
S	→	N
S	→	S,S
N	→	0
N	→	1
N	→	2
N	→	3
N	→	4
N	→	5
N	→	6
N	→	7
N	→	8
N	→	9
N	→	N0
N	→	N1
N	→	N2
N	→	N3
N	→	N4
N	→	N5
N	→	N6
N	→	N7
N	→	N8
N	→	N9

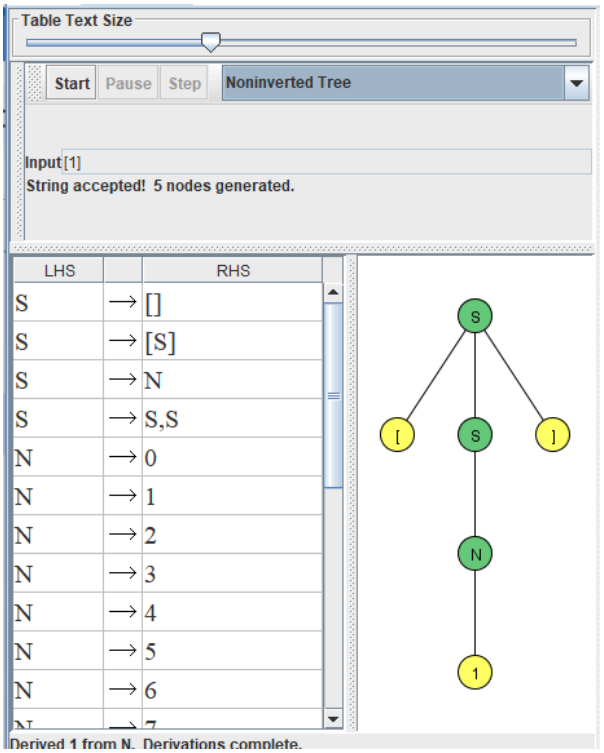
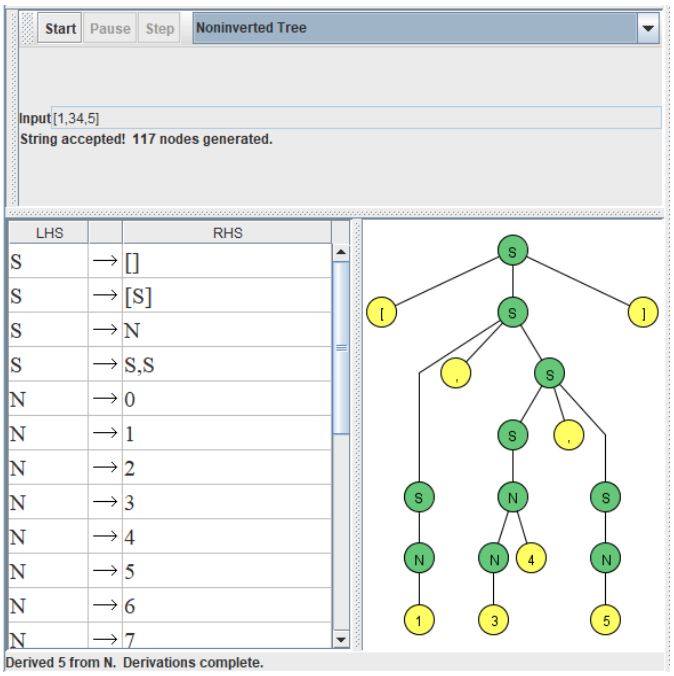
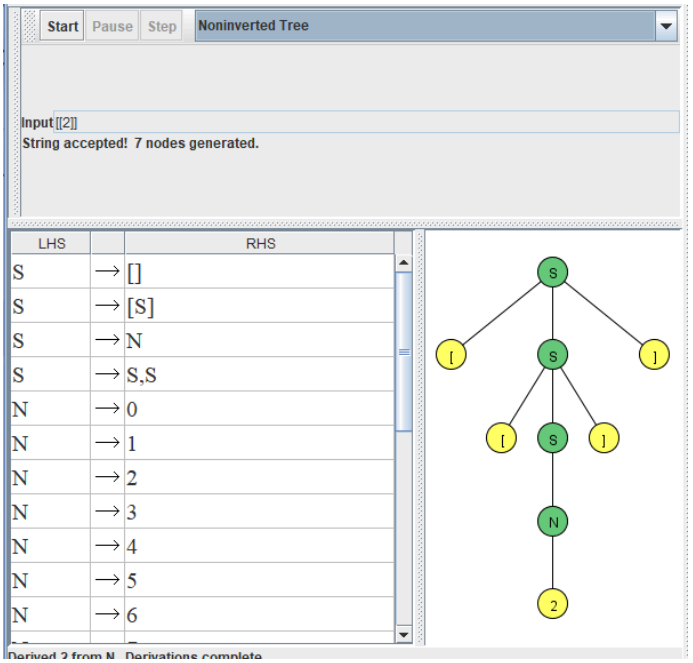
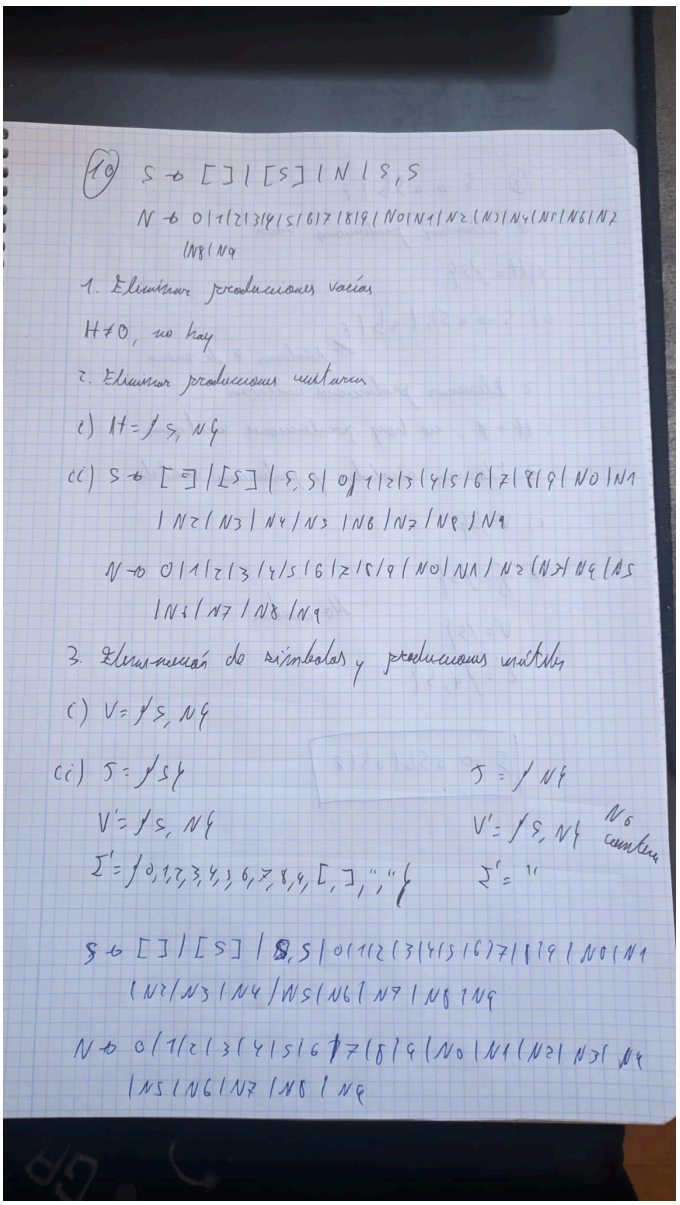


Table Text Size
Start
Pause
Step
Noninverted Tree
Input [1]
String accepted! 5 nodes generated.
LHS
RHS
S → []
S → [S]
S → N
S → S,S
N → 0
N → 1
N → 2
N → 3
N → 4
N → 5
N → 6
S
[
S
]
N
1
Derived 1 from N. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input [1,34,5]
String accepted! 117 nodes generated.
LHS
RHS
S → []
S → [S]
S → N
S → S,S
N → 0
N → 1
N → 2
N → 3
N → 4
N → 5
N → 6
N → 7
S
[
S
]
,
S
S
,
S
N
S
N
N
4
N
1
3
5
Derived 5 from N. Derivations complete.


Start
Pause
Step
Noninverted Tree
Input [[2]]
String accepted! 7 nodes generated.
LHS
RHS
S → []
S → [S]
S → N
S → S,S
N → 0
N → 1
N → 2
N → 3
N → 4
N → 5
N → 6
S
[
S
]
[
S
]
N
2


10) S → [] | [S] | N | S,S
1. Eliminar producciones vacías
H ≠ 0, no hay
2. Eliminar producciones unitarias
i) H = {S, N}
3. Eliminación de símbolos y producciones inútiles
i) V = {S, N}
ii) S = {S}
V' = {S, N}
Σ' = {"[", "]", ","}
S = {N}
V' = {S, N}
Σ' = {}
No cambian

LHS		RHS
S	→	[]
S	→	[S]
S	→	S,S
S	→	3
S	→	2
S	→	1
S	→	0
S	→	7
S	→	6
S	→	5
S	→	4
S	→	9
S	→	8
S	→	N1
S	→	N0
S	→	N5
S	→	N4
S	→	N3
S	→	N2
S	→	N9
S	→	N8
S	→	N7
S	→	N6
N	→	0
N	→	1
N	→	2
N	→	3
N	→	4
N	→	5
N	→	6
N	→	7
N	→	8
N	→	9
N	→	N8
N	→	N7
N	→	N6
N	→	N5
N	→	N4
N	→	N3
N	→	N2
N	→	N1
N	→	N0
N	→	N9

MODIFICACIÓN

L = {a^i b^j c^k | i=j o j=k}

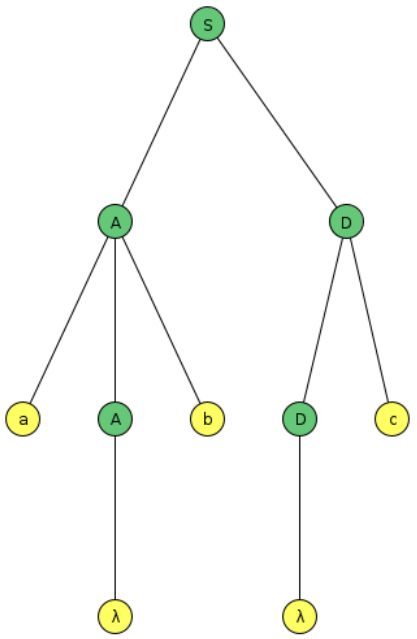
Esta gramática genera un lenguaje que acepta las cadenas con mismo numero de aes que de bes o mismo numero de bes que de ces. Para realizar esto tomamos dos caminos, uno en el que aceptamos el mismo número de aes que de bes con cualquier número de ces, cuya producción es generada por S -> AD , siendo A la producción para mantener el mismo número de aes que de bes y siendo D la producción de cualquier número de ces. Para el otro camino aplicamos la misma metodología.

LHS		RHS
S	→	AD
S	→	CB
A	→	aAb
A	→	λ
B	→	bBc
B	→	λ
C	→	Ca
C	→	λ
D	→	Dc
D	→	λ

Input abc

String accepted! 21 nodes generated.

LHS		RHS
S	→	AD
S	→	CB
A	→	aAb
A	→	λ
B	→	bBc
B	→	λ
C	→	Ca
C	→	λ
D	→	Dc
D	→	λ

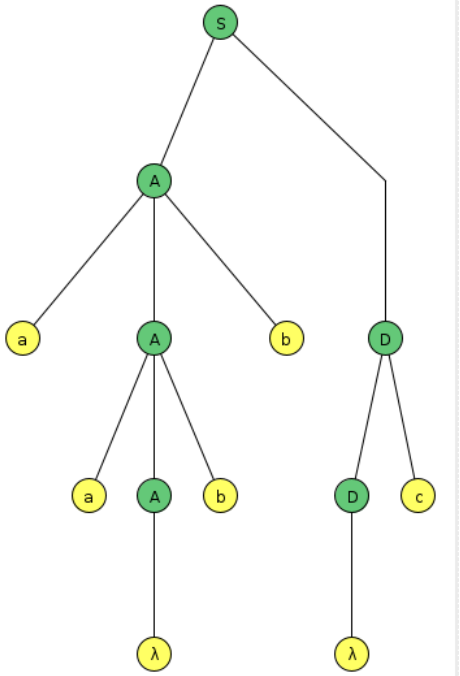


Derived λ from D. Derivations complete.

Input aabbc

String accepted! 29 nodes generated.

LHS		RHS
S	→	AD
S	→	CB
A	→	aAb
A	→	λ
B	→	bBc
B	→	λ
C	→	Ca
C	→	λ
D	→	Dc
D	→	λ

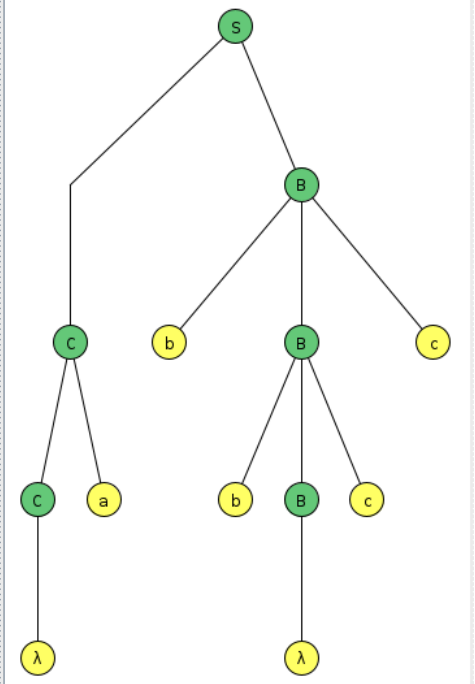


Derived λ from D. Derivations complete.

Input abbcc

String accepted! 34 nodes generated.

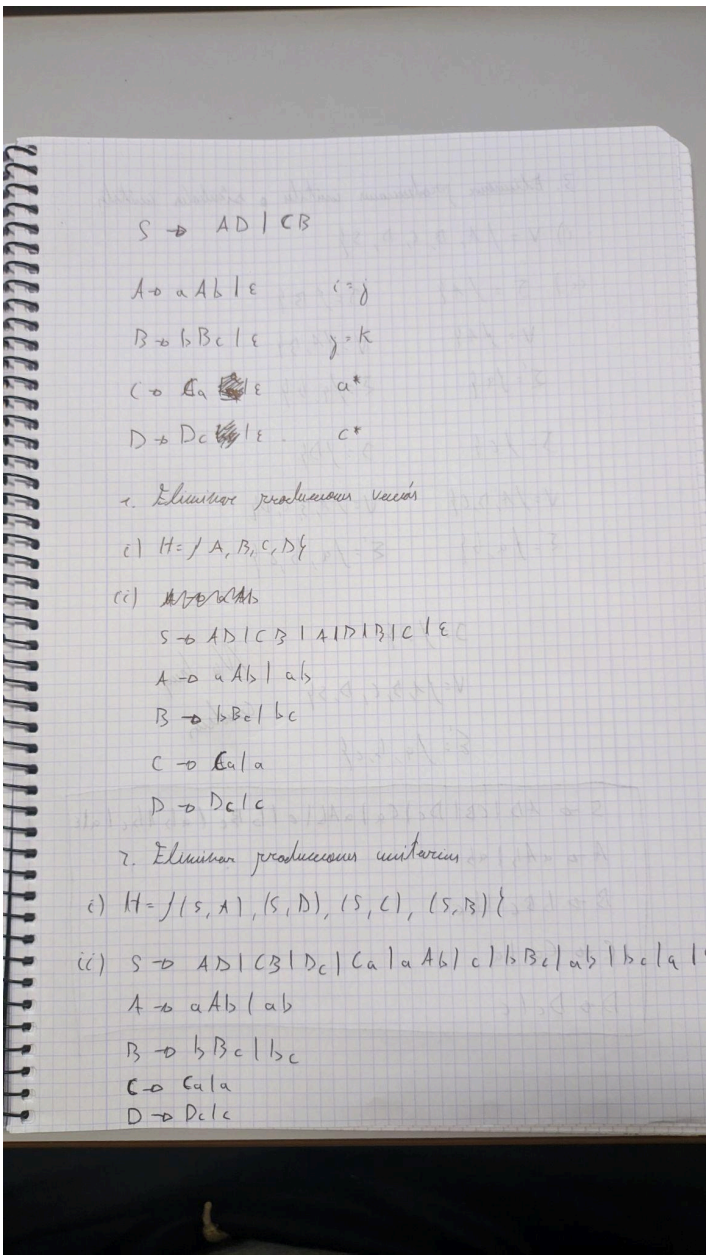
LHS		RHS
S	→	AD
S	→	CB
A	→	aAb
A	→	λ
B	→	bBc
B	→	λ
C	→	Ca
C	→	λ
D	→	Dc
D	→	λ

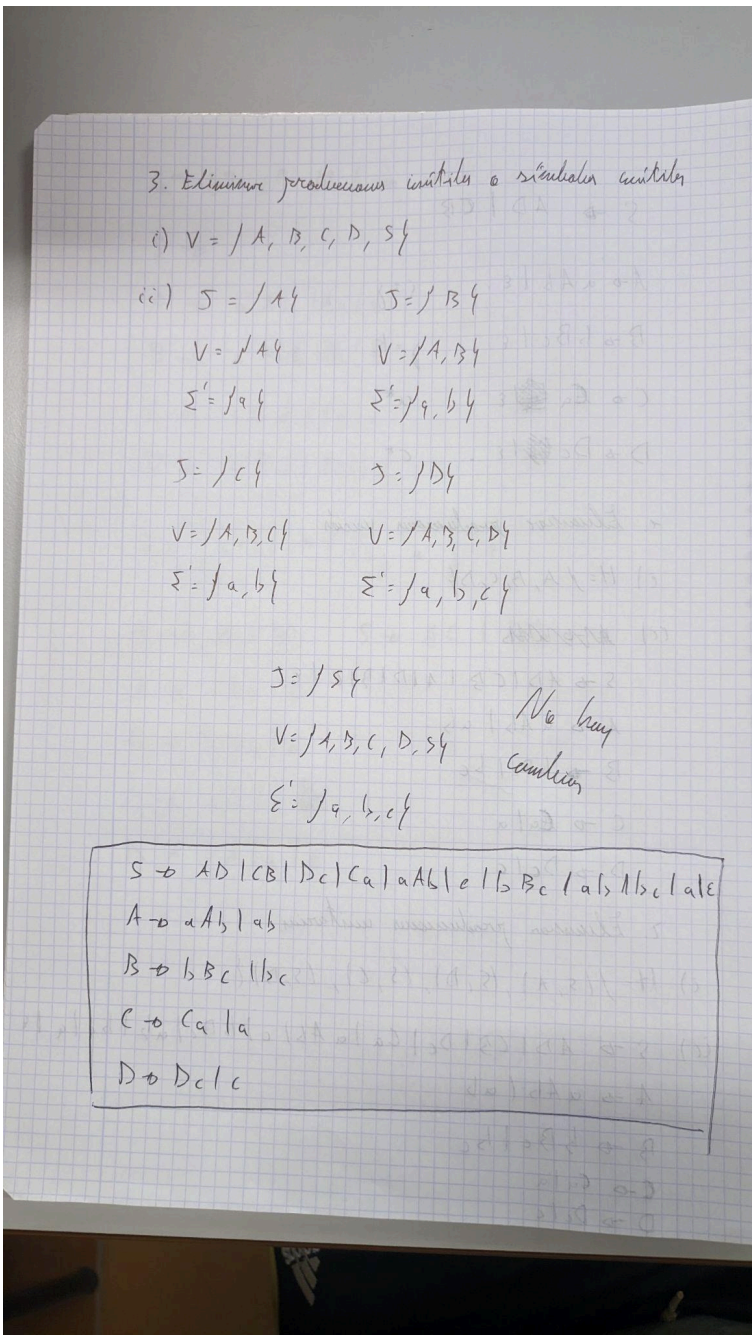


Derived λ from B. Derivations complete.

LHS		
S	→	AD
S	→	CB
S	→	Dc
S	→	Ca
S	→	aAb
S	→	c
S	→	bBc
S	→	ab
S	→	bc
S	→	a
D	→	c
C	→	a
B	→	bc
A	→	ab
C	→	Ca
D	→	Dc
B	→	bBc
A	→	aAb

Gramática simplificada. Proceso de simplificación a mano debajo.


S → AD | CB
A → aAb | ε i = j
B → bBc | ε j = k
C → Ca | ε a*
D → Dc | ε c*
1. Eliminar producciones vacías
i) H = {A, B, C, D}
ii) S → AD | CB | A | D | B | C | ε
A → aAb | ab
B → bBc | bc
C → Ca | a
D → Dc | c
2. Eliminar producciones unitarias
i) H = {(S, A), (S, D), (S, C), (S, B)}
ii) S → AD | CB | Dc | Ca | aAb | c | bBc | ab | bc | a
A → aAb | ab
B → bBc | bc
C → Ca | a
D → Dc | c



3. Eliminar producciones inútiles o símbolos inútiles

i) $V = \{A, B, C, D, S\}$

ii) $S = \{A\}$ $S = \{B\}$

$V = \{A\}$ $V = \{A, B\}$

$\Sigma' = \{a\}$ $\Sigma' = \{a, b\}$

$S = \{C\}$ $S = \{D\}$

$V = \{A, B, C\}$ $V = \{A, B, C, D\}$

$\Sigma' = \{a, b\}$ $\Sigma' = \{a, b, c\}$

$S = \{S\}$

$V = \{A, B, C, D, S\}$ No hay cambios

$\Sigma' = \{a, b, c\}$

$S \to AD \mid CB \mid Dc \mid Ca \mid aAb \mid \varepsilon \mid bBc \mid ab \mid bc \mid a$

$A \to aAb \mid ab$

$B \to bBc \mid bc$

$C \to Ca \mid a$

$D \to Dc \mid c$