1a Guía Compiladores

Nombre: Martínez Coronel Brayan Yosafat

Grupo: 3CM7 Fecha: 12/11/20

-Defina **compilador**

Programa que traduce un programa fuente en uno equivalente.

-Cuales son las dos partes de la compilación

1.- Análisis

2.- Síntesis

-Describa las 6 fases de un compilador

Análisis léxico: Divide la cadena en tokens

Análisis sintáctico: Crea un árbol sintáctico

Análisis semántico: Revisa cosas básicas de semántica en el árbol

Generador de código intermedio: Con código de 3 direcciones representa el árbol.

Optimización: Con menos recursos logra lo que el código de 3 direcciones.

Generador de código: el código optimizado se pasa a ensamblador (generalmente).

-Cuales son los 8 módulos de un compilador

Analizador léxico

Analizador sintáctico

Analizador semántico

Generador de código intermedio

Optimizador de código

Tabla de símbolos

Manejo de errores

A partir de hoc4 se usan dos etapas en hoc. ¿Cuáles son y qué hacen ?

1.- Generación de Código

2.- Ejecución

Falso o verdadero (F/V)

|  |  |
| --- | --- |
| 0.-A los terminales se les llama así porque no pueden ser sustituidos | V |
| 1.-Que una secuencia de caracteres concreta sea un token depende del lenguaje | V |
| 2.-Las cadenas que pertenecen al lenguaje generado por una gramática están hechas solo de terminales | V |
| 3.-El **análisis léxico** lee la cadena de entrada de derecha a izquierda | V |
| 4.-El **análisis léxico** construye el árbol de análisis sintáctico | F |
| 5.-La secuencia de caracteres que forma un componente léxico es el **lexema** del componente | V |
| 6.-La gramática S → **a**S |S**a** | **a** se puede analizar con un **analizador sintáctico predictivo descendente recursivo** | F |
| 7.-El tipo de **yylval** no es el mismo que el de los elementos en la pila de YACC | F |
| 8.-La única forma de indicar el tipo de los elementos en la pila de YACC es usando #define YYSTYPE | F |
| 9.-El **código intermedio** debe ser fácil de generar | V |
| 10.- Un **esquema de traducción** es una **GLC** + **reglas semánticas** | V |
| 11.- Árbol de análisis sintáctico con anotaciones es sinónimo de **árbol decorado** | V |
| 12-Análisis sintáctico **descendente** es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz | F |
| 13-Análisis sintáctico **ascendente** es donde la construcción del árbol de análisis sintáctico se inicia en las hojas y avanza hacia la raíz | V |
| *14.-*yylex() llama a yyparse() | F |
| 15.-yyparse() llama a yylex() | V |
| *16.-*yylex() retorna el **tipo** de **token** | V |
| 17.-yylval almacena el **lexema** | F |
| 18-HOC1 es una **calculadora** | V |
| 19-Las **variables** en HOC son de tipo entero | F |
| 20.-La notación **posfija** es una notación matemática libre de paréntesis y en esta notación los operadores aparecen después de los operandos | V |
| 21.-La **raíz** del árbol de análisis sintáctico se etiqueta con el símbolo inicial | V |
| 22.- Las **hojas** del árbol de análisis sintáctico se etiquetan con no terminales | F |
| 23.-En la notación **infija** la **asociatividad** y la **precedencia** se usan para determinar en qué orden hay que realizar las operaciones para **evaluar** una expresión | V |

Para que sirve el **Análisis Léxico**

|  |  |  |
| --- | --- | --- |
| a) Para generar el código en lenguaje objeto | b) Nos dice si una cadena pertenece al lenguaje generado por una gramática | ( C ) |
| c) Para dividir una cadena en tokens | d) Los compiladores no lo necesitan nunca |

El \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ comprueba que el orden en que el **analizador léxico** le va entregando los tokens es válido.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) analizador semantico | b) analizador sintáctico | c) optimizador | d) generador de codigo | ( B ) |

Es una *gramática que* tiene cuatro componentes:

1. Un conjunto de componentes léxicos*.*

1. Un conjunto de no terminales.
2. Un conjunto de producciones, en el que cada producción consta de un no terminal, llamado *lado izquierdo* de la producción, una flecha y una secuencia de componentes léxicos y no terminales, o ambos, llamado *lado derecho* de la producción.
3. .La denominación de uno de los no terminales como símbolo *inicial.*

|  |  |  |
| --- | --- | --- |
| a) Gramática Asociativa por la izquierda | b) Gramática recursiva | ( C ) |
| c) Gramática libre de contexto (GLC) | d) Gramática ambigua |  |

Cual de las sigs. opciones no es sinónimo de las otras

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) Componente léxico | b) no terminal | c) token | d) Simbolo gramatical | ( B ) |

Es una gramática donde en el lenguaje que genera existe una cadena que tiene mas de un árbol de análisis sintáctico.

|  |  |  |
| --- | --- | --- |
| *a)* Gramática recursiva por la izquierda | *b) Gramática recursiva* | ( D ) |
| c) Gramática libre de contexto | d) Gramática *ambigua* |  |

Si Una gramática contiene una regla de producción de la forma A *→* A  entonces es una

|  |  |  |
| --- | --- | --- |
| *a)* Gramática recursiva por la izquierda | *b) Gramática ambigua* | ( A ) |
| c) Gramática libre de contexto | d) ninguna de las anteriores |  |

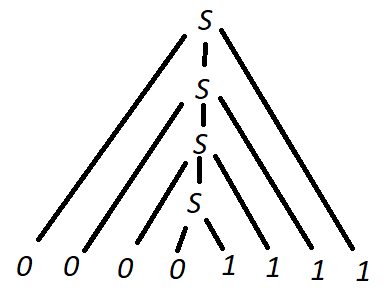
Considere la siguiente gramática

*S→* ***0*** *S* ***1*** *|* ***01***

a ) Mostrar una derivación de ***00001111***

b ) Dibuje el árbol de análisis sintáctico para la entrada ***00001111***

S -> 0S1 -> 00S11 -> 000S111 -> 00001111



Considere la siguiente gramática

S → **b**A

A → **b**B

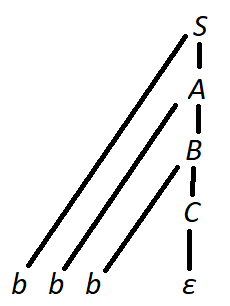
B → **b**C

C → **ε**

a ) Mostrar una derivación de **bbb**

b ) Dibuje el árbol de análisis sintáctico para la entrada **bbb**

S -> bA -> bbB -> bbbC -> bbb



Considere la siguiente gramática

*S* → *A*

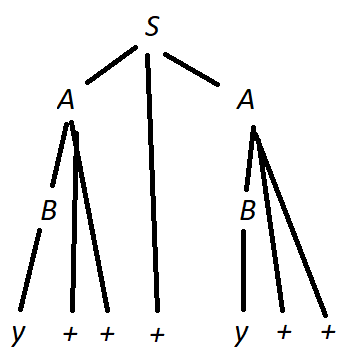
*A* → *A***+***A* | *B****++***

*B* → ***y***

a ) Mostrar una derivación de **y + + + y + +**

b ) Dibuje el árbol de análisis sintáctico para la entrada **y + + + y + +**

S -> A -> A+A -> B+++A -> B+++B++ -> y+++B++ -> y+++y++



Considere la siguiente gramática

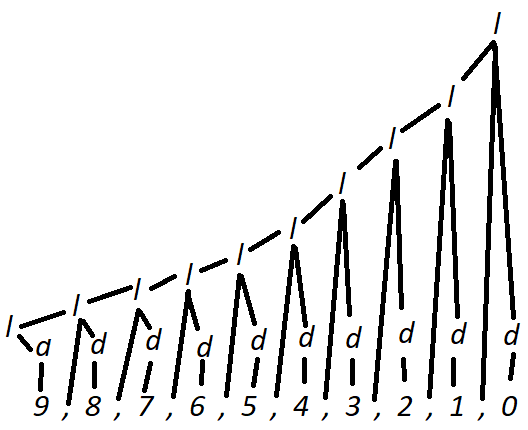
*l→ l* ***,*** *d | d*

*d→* ***0*** *|* ***1*** *|* ***2*** *|* ***3*** *|* ***4*** *|* ***5*** *|* ***6*** *|* ***7*** *|****8*** *|* ***9***

a ) Mostrar una derivación de ***9,8,7,6,5,4,3,2,1,0***

b ) Dibuje el árbol de análisis sintáctico para la entrada ***9,8,7,6,5,4,3,2,1,0***

l -> l,d -> l,d,d -> l,d,d,d -> l,d,d,d,d -> l,d,d,d,d,d -> l,d,d,d,d,d,d -> l,d,d,d,d,d,d,d -> l,d,d,d,d,d,d,d,d -> l,d,d,d,d,d,d,d,d,d -> d,d,d,d,d,d,d,d,d,d -> 9,d,d,d,d,d,d,d,d,d -> 9,8,d,d,d,d,d,d,d,d -> 9,8,7,d,d,d,d,d,d,d -> 9,8,7,6,d,d,d,d,d,d -> 9,8,7,6,5,d,d,d,d,d -> 9,8,7,6,5,4,d,d,d,d -> 9,8,7,6,5,4,3,d,d,d -> 9,8,7,6,5,4,3,2,d,d -> 9,8,7,6,5,4,3,2,1,d -> 9,8,7,6,5,4,3,2,1,0



Dada la gramática

T={**a**, **b**, +, -, \*, /, (,)},

N={E, T, F}

S={E}

P={ E->T | E+T | E-T

T->F | T\*F | T/F

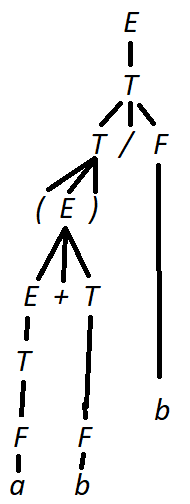
F-> **a** | **b** | (E) }

y la cadena **(a+b)/b**

a) Obtenga una derivación de dicha cadena

b) Dibuje el árbol de análisis sintáctico que corresponde a la cadena mencionada

E -> T -> F -> T/F -> (E)/F -> (E+T)/F -> (T+T)/F -> (F+T)/F -> (F+F)/F -> (a+F)/F -> (a+b)/F -> (a+b)/b



**Análisis sintáctico predictivo descendente recursivo**

Considere la siguiente gramática

S →**a** | **(** S **)**

Escriba el analizador sintáctico predictivo descendente recursivo

void S() {

if (preana == ‘(’)

parea(‘(’); S(); parea(‘)’);

else if (preana == ‘a’)

return;

else error();

}

**Ambigüedad**

Demostrar que la siguiente gramática es ambigua

S → **a**S |S**a** | **a**

usando la cadena **aa**

Como se puede obtener mediante: S -> aS -> aa, y con S -> Sa -> aa, es ambigua.

Demostrar que la siguiente gramática es ambigua

A →A **x** B | **x**

B → **x** B | **x**

usando la cadena **xxxxx**

Como A -> AxB -> AxBxB -> xxBxB -> xxxxB -> xxxxx y también se puede obtener con A -> AxB -> AxxB -> AxxxB -> xxxxB -> xxxxx. Entonces es ambigua.

Demostrar que la siguiente gramática es ambigua

S→ **a** S **b** S | **b** S **a** S | **ε**

usando la cadena **abab**

Como S -> aSbS -> aSbaSbS -> abaSbS -> ababS -> abab, y también S -> aSbS -> abSaSbS ->abaSbS -> ababS -> abab, entonces es ambigua.

Verificar si las siguientes gramáticas son ambiguas

S → S **+** S | S **-** S | **a**

S → S S **+** | S S **-** | **a**

La primera lo es: S -> S+S -> S+S-S -> … -> a+a-a, y S -> S-S -> S+S-S -> … -> a+a-a

La segunda no lo es.

**Recursividad por** la **izquierda**

Para eliminar la recursividad por la izquierda

A → A**a** | **b**

se transforma en

A → **b** | **b**R

R → **a**R | **ε**

Ahora considere las siguientes gramáticas

A→ 1 | A 0

y

S→ ( L ) | a

L→ L , S | S

Elimine la recursividad por la izquierda de dichas gramáticas.

Escriba el analizador sintáctico predictivo descendente recursivo para dichas gramáticas

Escriba la sección de reglas de la especificación de YACC para dichas gramáticas

Para la primera:

A -> 1B

B -> 0B | **ε**

**void A(){**

**if (preana == ‘1’)**

**parea(‘1’); B();**

**else error();**

**}**

**void B() {**

**if (preana == ‘0’)**

**parea(‘0’); B();**

**}**

Para la segunda:

S→ ( L ) | a

L→ ( L1 ) | a

L1→ L1, ( L1 ) | L1 , a

void S(){

if (preana == ‘(’)

parea(‘(’); L();parea(‘)’);

else if(preana == ‘a’)

parea(‘a’);

else error();

}

void L() {

if (preana == ‘(’)

parea(‘(’); L1();parea(‘)’)

else if(preana == ‘a’)

parea(‘a’);

else error();

}

void L1() {

if (preana == ‘,’)

parea(‘,’); parea(‘(’); L1();parea(‘)’);

else if(preana == ‘,’)

parea(‘,’);parea(‘a’);

else error();

}

%%

S: ‘(’ L ‘)’

| printf(“a”)

;

L: ‘(’ L1 ‘)’

| printf(“a”)

;

L1: ‘,’ ‘(’ L1 ‘)’

|‘,’ printf(“a”)

;

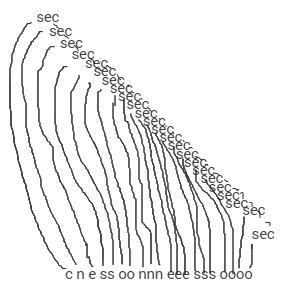
%%

**Definiciones dirigidas por la sintaxis**

|  |  |
| --- | --- |
| Producción | Regla Semántica |
| *sec →* **comienza** | *sec.x = 0*  *sec.y = 0* |
| *sec →* *sec1 instr* | *sec.x = sec1.x + instr.dx*  *sec. y = sec1.y + instr.dy* |
| *instr →* **este** | *instr.dx = 1*  *instr.dy = 0* |
| *instr →* **norte** | *instr.dx = 0*  *instr.dy = 1* |
| *instr →* **oeste** | *instr.dx = -1*  *instr.dy = 0* |
| instr → **sur** | *instr.dx = 0*  *instr.dy = -1* |

Dibuje el árbol de análisis sintácticocon anotaciones para la sig cadena

c n e ss oo nnn eee ssss oooo



Escribir la sección de reglas de la especificación de yacc para calcular la posición final del robot.

%%

sec : COMIENZA { $$.x = 0; $$.y = 0; }

| sec instr { $$.x = $1.x + $2.dx;

$$.y = $1.y + $2.dy; }

| sec TERMINA { printf("Nuevas coordenadas: %d,%d\n",

$$.x, $$.y); exit(0);}

;

instr : ESTE { $$.dx = 1; $$.dy = 0; }

| NORTE { $$.dy = 0; $$.dy = 1; }

| OESTE { $$.dx = -1; $$.dy = 0; }

| SUR { $$.dx = 0; $$.dy = -1; }

;

%%

**Esquemas de traducción**

Escriba una definición dirigida por la sintaxis para evaluar expresiones booleanas.

E → E OR E

| E AND E

| NOT E

| ( E )

| TRUE

| FALSE

Para cada esquema de traducción de las expresiones booleanas escriba la sección de reglas de la especificación de YACC

%%

E E ‘OR’ E {$$= if($1 || $2 ) }

| E ‘AND’ E{$$= if($1 &&$2 ) }

| ‘NOT’ E {$$ = if($$1 && $$ 1) 0 else 1 }

| ‘(’ E ‘)’ {$$ = $2}

| ‘TRUE’ {$$ = 1}

| ‘FALSE’ {$$ = 0}

;

%%

**Escritura de Gramaticas**

Escribir una gramática que genere todas las cadenas de longitud 4 formadas con los símbolos del alfabeto {a,b,c}

S -> letra letra letra letra

letra -> ‘a’ | ‘b’ | ‘c’

|  |  |  |  |
| --- | --- | --- | --- |
| **Escribir una gramática que sirva para generar las siguientes cadenas** | | | |
| **Especie perro**  **Edad 1**  **Sexo macho**  **Tamaño grande**  **Colores negro , blanco**  **Soy rápido , activo, alegre**  **Aficiones correr, comer** | **Especie gato**  **Edad 2**  **Sexo macho**  **Tamaño mediano**  **Colores negro , blanco , café**  **Soy tranquilo , sociable**  **Aficiones dormir, parrandear, comer** | **Especie perro**  **Edad 2**  **Sexo hembra**  **Tamaño pequeño**  **Colores canela , gris**  **Soy fuerte , alegre, activo.**  **Aficiones aullar** | **Especie gato**  **Edad 2**  **Sexo macho**  **Tamaño grande**  **Colores blanco**  **Soy listo , obediente**  **Aficiones jugar, haraganear** |

S -> especie edad sexo tamaño color ser afición

especie -> ‘Especie perro’ | ‘Especie gato’

edad -> ‘Edad 1’ | ‘Edad 2’

sexo -> ‘Sexo macho’ | ‘Sexo hembra’

tamaño -> ‘Tamaño grande’ | ‘Tamaño mediano’ | ‘Tamaño chico’

color -> ‘Colores ’ c

c -> c ‘,’ c | ‘negro’ | ‘blanco’ | ‘café’ | ‘canela’ | ‘gris’

afición -> ‘Aficiones’ adjetivo

adjetivo -> adjetivo ‘,’ adjetivo | ‘rápido’| ‘activo’ | ‘alegre’ | ‘tranquilo’ | ‘sociable’ | ‘fuerte’ | ‘listo’ | ‘obediente’

afición -> ‘Aficiones’ tipo2

tipo2 -> tipo2 ‘,’ tipo2 | ‘correr’ | ‘comer’ | ‘parrandear’ | ‘aullar’ | ‘jugar’ | ‘haraganear’ | ‘dormir’

|  |  |  |  |
| --- | --- | --- | --- |
| 12.-Escribir una gramática que sirva para generar las siguientes cadenas | | | |
| Etiquetado Nerd  Nivel Junior  Sexo Hombre  Lenguajes Java , C , Prolog  Aficiones programar, videogames, comics, hackear, googlear | Etiquetado Geek  Nivel Senior  Sexo Mujer  Lenguajes Pascal , Prolog , SQL  Aficiones chatear, videogames, programar | Etiquetado Nerd  Nivel Junior  Sexo Mujer  Lenguajes PHP , Perl, Java  Aficiones hackear, googlear, gotcha, dormir | Etiquetado Freak  Nivel Senior  Sexo Hombre  Lenguajes Ensamblador, C  Aficiones gotcha, dormir, chatear, comics |

S -> etiqueta nivel sexo lenguaje afición

etiqueta -> ‘Etiquetado Nerd’ | ‘Etiquetado Geek’ | ‘Etiquetado Freak’

nivel -> ‘Nivel Junior’ | ‘Nivel Senior’

sexo -> ‘Sexo Hombre’ | ‘Sexo Mujer’

lenguaje -> ‘Lenguajes ’ tipo

tipo -> tipo ‘,’ tipo | ‘Java’ | ‘C’ | ‘Prolog’ | ‘SQL’ | ‘Pascal’ | ‘PHP’ | ‘Perl’ | ‘Ensamblador’

afición -> ‘Aficiones’ tipo2

tipo2 -> tipo2 ‘,’ tipo2 | ‘programar’| ‘videogames’ | ‘hackear’ | ‘googlear’ | ‘chatear’ | ‘gotcha’ | ‘dormir’ | ‘comics’

**YACC**

.-Para que sirve $$

Simboliza que es la parte izquierda de la producción

.-Dentro de una acción gramatical $n se refiere al enésimo símbolo en el lado derecho de la producción

1.-Los %% se usan para indicar

|  |  |  |
| --- | --- | --- |
| a)inicio de la sección de declaraciones | b)inicio de la sección de reglas | ( b ) |
| c)precedencia de los operadores | d)fin del código de soporte |

2.-%token sirve para indicar

|  |  |  |
| --- | --- | --- |
| a)inicio de la sección de declaraciones | d)los no terminales de la gramática | ( d ) |
| c)precedencia de los operadores | d)los terminales de la gramática |

3.-Como le indica el analizador léxico (yylex) al analizador sintáctico (yyparse) que ya no hay mas tokens en la entrada

|  |  |  |
| --- | --- | --- |
| a) retornando cero | b) retornando -1 | ( a ) |
| c) almacenando -1 en yylval | d) almacenando 0 en yylval |

4.-Una acción gramatical debe ir entre

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) comillas | b) paréntesis | c) corchetes | d) llaves | ( d ) |

5.-Considere la producción

S : S ' a' S 'b'

$4 a cual de los miembros del lado derecho de la producción se refiere?

|  |  |  |
| --- | --- | --- |
| a)la 'a' | b)la 1er S | ( d ) |
| c)la segunda S | d)la 'b' |

Si el codigo de yylex es el siguiente

int yylex() { return getchar(); }

de cuantos caracteres son los tokens

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| a) 0 | b) 1 | c) 2 | d) la cantidad de caracteres del token varia | ( b ) |

Considere la siguiente gramática (los terminales se indican en negritas)

L-> L**,** D | D

D-> **0** | **1**

Escriba la sección de reglas de la especificación de yacc para dicha gramática

%%

L -> L ‘,’ D

| D

;

D -> ‘0’

| ‘1’

;

%%

Escriba la especificación de yacc para la gramática

S → U | V

U → T**a**U | T**a**T

V → T**b**V | T**b**T

T → **a**T**b**T | **b**T**a**T | **ε**

%%

S -> U

| V

;

U -> T ‘**a’** U

| T ‘**a’** T

;

V -> T ‘**b’** V

| T ‘**b’** T

;

T → ‘**a’** T ‘**b’** T

| ‘**b’** T ‘**a’** T

|

**;**

%%

Escriba las acciones gramaticales para que imprima el numero de b's en la cadena de entrada

%{

/\*escriba el tipo de los elementos en la pila de yacc \*/

#define YYSTYPE double

%}

%%

S : ‘(’ B ‘)’ {$$ = %2; printf(“%d”, $$);}

;

B : ‘(’ B ‘)’ { $$ = $2;}

| D { $$=$1; }

;

D : { $$ = 0; }

| ‘b’ D { $$ = 1 + D; }

;

%%

Considere la siguiente gramática (los terminales se indican en negritas)

lista->lista **,** figura | figura

figura-> triangulo | cuadrilatero

triangulo-> **lado lado lado**

cuadrilatero-> **lado lado lado lado**

Escriba la sección de reglas de la especificación de yacc para dicha gramática y las acciones semánticas respectivas para que se imprima si un triangulo es equilátero y si un cuadrilátero es un cuadrado

%%

lista -> lista **,** figura {}

| figura {}

;

figura-> triangulo {}

| cuadrilatero {}

;

triangulo -> **lado lado lado {printf(“Es un triángulo equilátero\n”);}**

cuadrilatero -> **lado lado lado lado {printf(“Es un cuadrado\n”);}**

%%

**Análisis Sintáctico Predictivo no Recursivo**

-Para las siguientes GLC construya la tabla Análisis Sintáctico Predictivo no Recursivo

-Use dicho análisis para analizar las cadenas propuestas:

-Muestre el contenido de la pila, la entrada y la acción a realizar

**Problema 1.-**Considere la gramática para generar paréntesis anidados

|  |  |
| --- | --- |
| 1) A → **(** A **)** | 2) A → **a** |

Cadenas propuestas:

**( a )**

**( ( a ) )**

**( ( ( a ) ) )**

**( ( ( ( a ) ) ) )**

**Problema 2.-**Considere la siguiente gramática :

|  |  |  |  |
| --- | --- | --- | --- |
| 1) S → **a** | 2) S → **(** S R | 3) R → **,** S R | 4) R → **)** |

Cadenas propuestas:

**( a )**

**(a , a)**

**(a , a, a)**

**(a , a, a, a )**

**Problema 3.-**Considere la siguiente gramática :

|  |  |  |  |
| --- | --- | --- | --- |
| 1) S → A**a**A**b** | 2) S → B**b**B**a** | 3) A → **ε** | 4) B → **ε** |

Cadenas propuestas:

**ab** y **ba**

**Problema 4.-**Considere la siguiente gramática :

S → A

A → **ε**

A → **bb**A

Cadena propuesta:

**bbbb**