Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Compiladores e Interpretes

Profesor

Allan Rodríguez

Estudiante

Estefani Valverde Marín 2021554564

Portafolio 2

Investigación

Gramáticas Libres de Contexto

1. ¿Qué es una Gramática Libre de Contexto?

Una **Gramática Libre de Contexto (GLC)** es un tipo de gramática formal que se utiliza en informática teórica y lingüística para definir los lenguajes formales. Fue introducida por **Noam Chomsky** en 1956.

Una GLC está compuesta por un cuádruple G=(V,Σ,R,S)G = (V, \Sigma, R, S) donde:

- VV es un conjunto finito de variables (no terminales)
- Σ\Sigma es un conjunto finito de símbolos terminales (alfabeto)
- RR es un conjunto finito de reglas de producción (de la forma A → α, donde A ∈ V y α ∈ (V ∪ Σ)*)
- S∈VS \in V es el símbolo inicial

3. Derivaciones

Una **derivación** es el proceso de aplicar reglas de producción para transformar el símbolo inicial en una cadena terminal.

Tipos:

- **Izquierda**: se reemplaza siempre el símbolo más a la izquierda.
- **Derecha**: se reemplaza el símbolo más a la derecha.

Ejemplo con reglas:

S → aSb | ε

Derivación de la cadena "aabb":

S ⇒ aSb ⇒ aaSbb ⇒ aabb

4. Gramática Pascal (ejemplo simplificado)

El lenguaje Pascal es conocido por su estructura clara. Su gramática define bloques, declaraciones y estructuras de control.

Ejemplo (simplificado en BNF):

<program> ::= "program" <identifier> ";" <block> "."

<block> ::= <declarations> <compound_statement>

<compound_statement> ::= "begin" <statement_list> "end"

5. Gramática C (simplificada)

El lenguaje C también puede representarse por una GLC, aunque su gramática completa es más compleja y se extiende a varias páginas.

Ejemplo (BNF simplificado):

<function> ::= <type> <identifier> "(" <params> ")" "{" <statements> "}"

<type> ::= "int" | "float" | "char"

<params> ::= <type> <identifier> | ε

6. Gramática Java (simplificada)

Java tiene una gramática libre de contexto documentada por Oracle. Define la estructura de clases, métodos, variables, etc.

Ejemplo (simplificado):

<compilationUnit> ::= {<importDeclaration>} {<typeDeclaration>}

<typeDeclaration> ::= <classDeclaration> | <interfaceDeclaration>

<classDeclaration> ::= "class" <identifier> "{" {<classBodyDeclaration>} "}"

Fuente: Java Language Specification (Oracle)

Ejercicio 1 de gramática

Gramática que permita un único main y cero o muchas funciones (main y funciones con una única expresión de asignación o creación)

main-> expresión

Expresión -> expresiones*

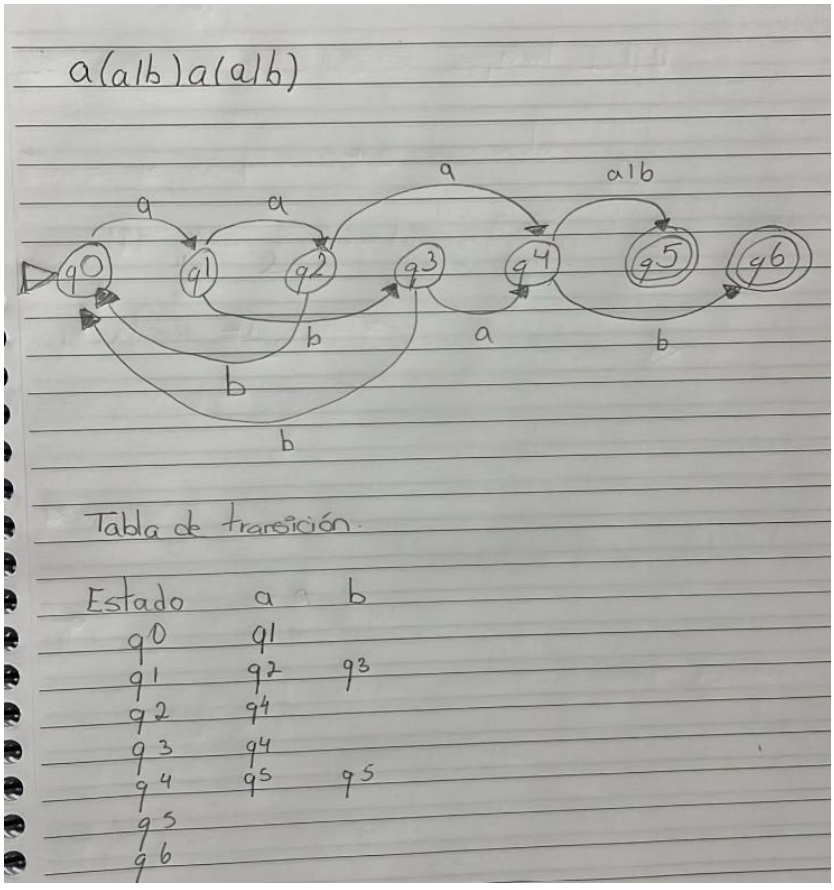
Expresión -> función

Función -> tipo id(parámetro) bloque

Función -> tipo id

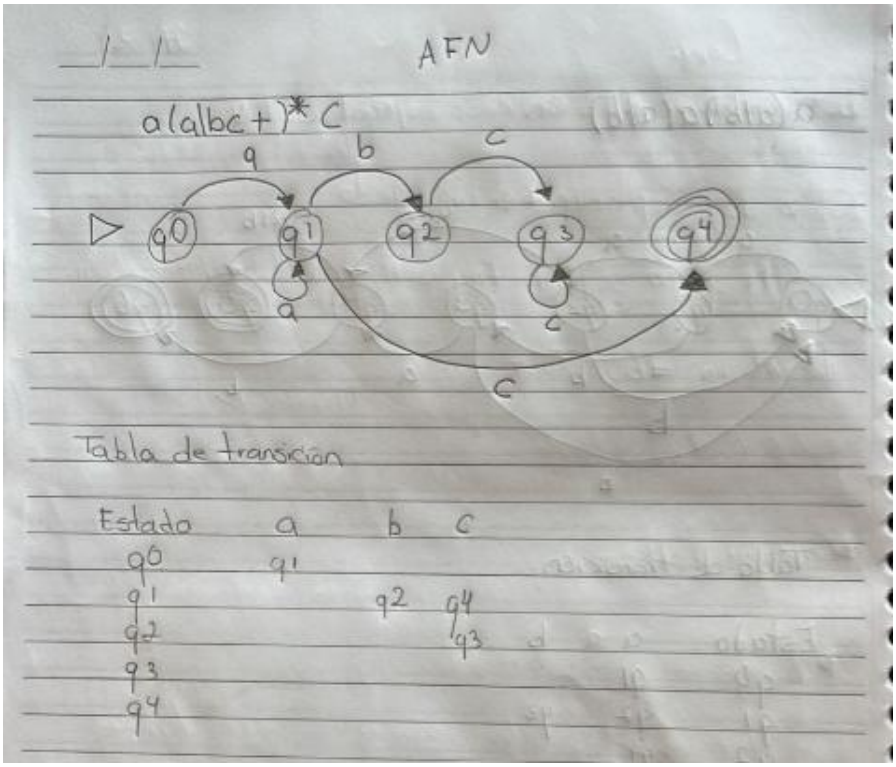
Ejercicio 2

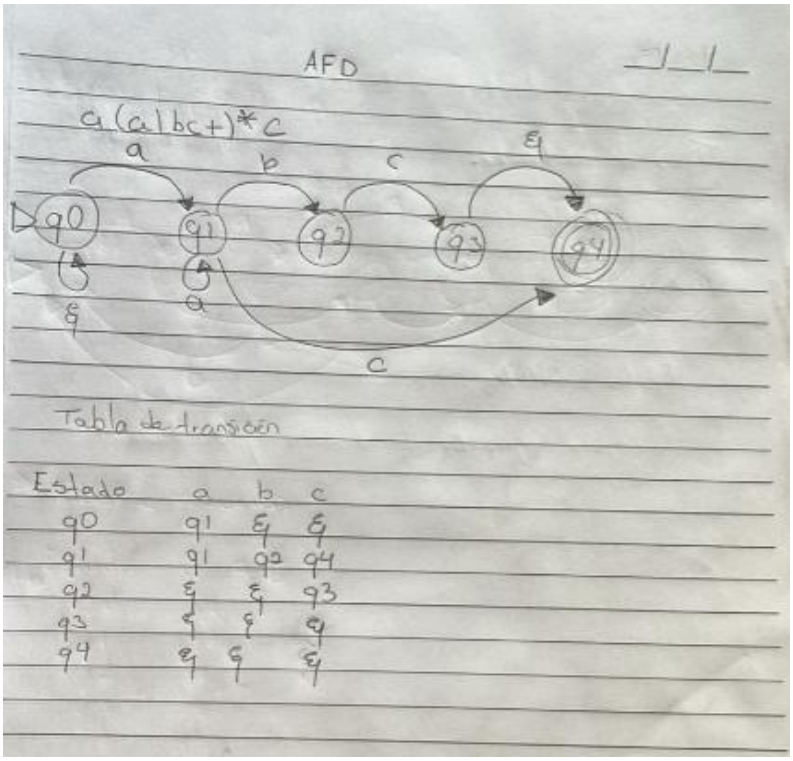
Desarrolle el AFN del lenguaje denotado por la siguiente expresión regular y su tabla de transición.



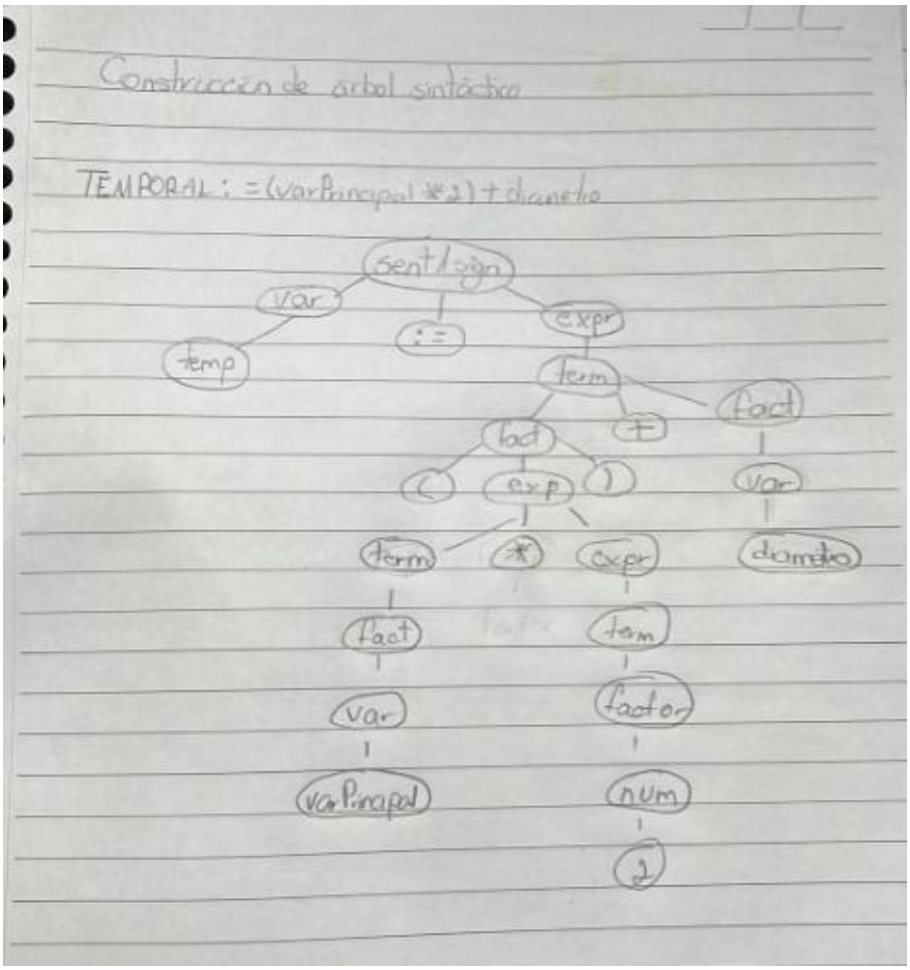
Ejercicio 3

Desarrolle el AFN, su tabla de transición, el AFD y su trabla de transición de la siguiente expresión regular

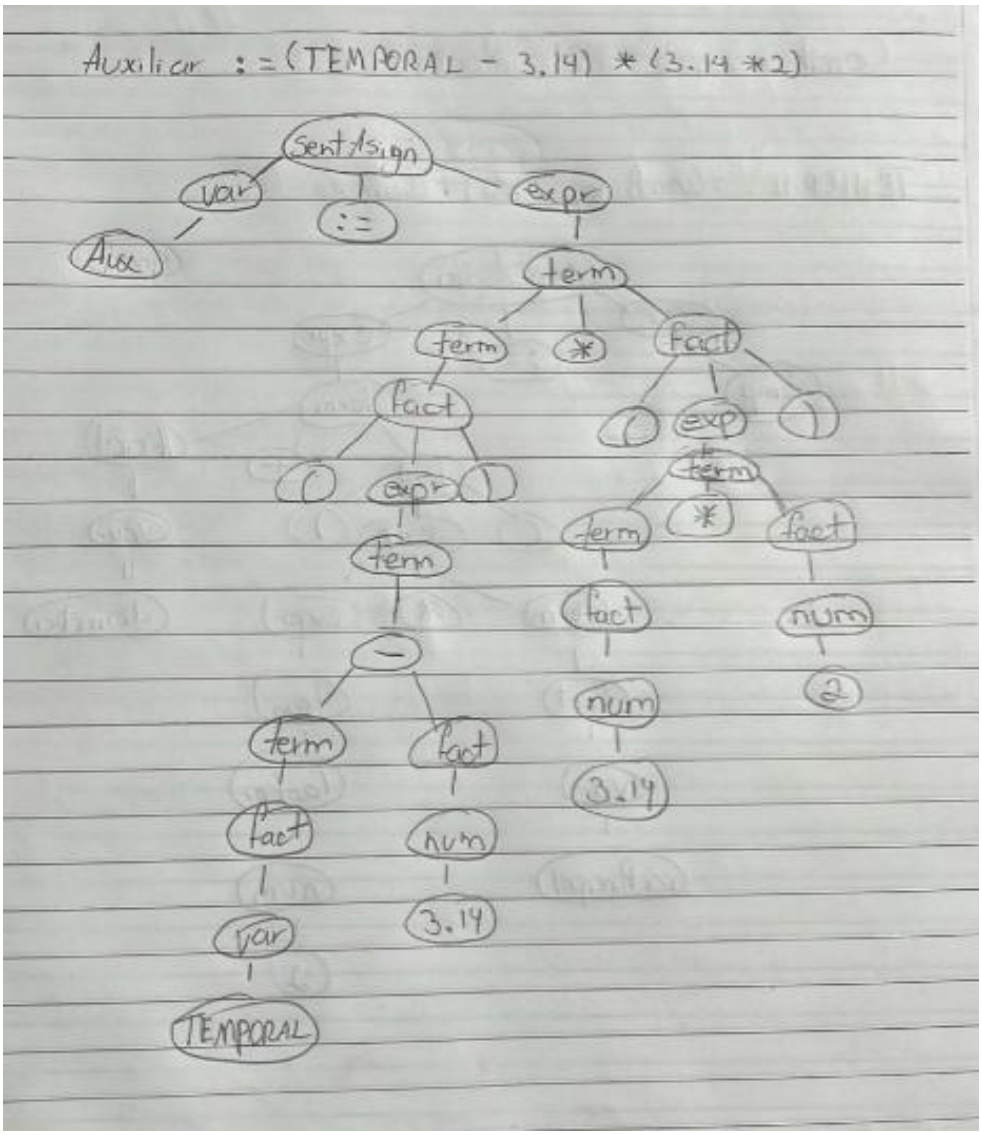




Construcción de árbol sintáctico



Auxiliar := (TEMPORAL - 3.14) * (3.14 * 2)



Gramática árbol sintáctico

[inicio [decs [decs [dec

[func 'func' [sep :] [tipo int] [sep :] [id principal] '_'

[bloque [bloque

[expr [exprUni [[exprP [exprArit [operandoArit

[invocaFunc

[[id ejecutarFuncion] '(' [parametrosInv [parametrosInv [paramInv [exprP [exprArit [operandoArit [id saludo]]]]]] ',' [paramInv [exprP [exprArit [operandoArit [literal [charN 'a']]]]]] ')']]]]]

[finExpr ;]]]]]

[expr [retorno 'return' [finExpr ;]]]]

'_']] [dec

[func]]]]

[inicio [desc [desc [desc [func 'func' [sep :] [tipo int] [sep :] [id ejecutarFuncion] '_'

[[param [sep :][tipo int] [sep :] [id dato][endExpr ;]]] [loc [sep :] [tipo int] [sep :] [id 'a'][endExpr ;]] [param [sep :][tipo char][sep :][id letra][endExpr ;]] [tipo glob [sep :][tipo string] [id mensaje][signo =][expr ["Hola $%&/#$&) mundo"]][endExpr ;]] [tipo loc [sep :][tipo int][id variable][signo =][tipo [int[valor[34]]]][signo +][id][signo ^][id funcionSecundaria][(][id procesar][(][tipo int [34]][,][id][)][)][endExpr ;]] [if][for[sep :][id variable][sep :][in][range[(][num 5][)]]]*[][print[[(][("Hola mundo"][)]]]][endExpr ;][*]]

]]]]]

Lectura y resumen

Gramática libre de contexto

Definición

Una gramática libre de contexto se define como un conjunto de reglas que especifican cómo formar cadenas válidas en un lenguaje. Está compuesta por:

- **Terminales**: Son los símbolos más básicos (como if, else, +, *, (,), etc.), es decir, los que aparecen directamente en el lenguaje.
- **No terminales**: Representan conjuntos de cadenas y se usan para definir la estructura del lenguaje. Ejemplos: expr, stmt, E, T, F.
- **Símbolo inicial**: Es el no terminal desde el cual comienzan las derivaciones. Por convención, se coloca primero en la lista de reglas.
- **Producciones**: Reglas de la forma A → α donde A es un no terminal y α una secuencia de símbolos terminales y/o no terminales.

Estas gramáticas permiten generar estructuras como expresiones, sentencias condicionales, bucles, etc., de forma sistemática y jerárquica.

Notación

Los siguientes símbolos se consideran *terminales*:

- (a) Letras minúsculas al inicio del alfabeto, como *a*, *b*, *c*.
- (b) Símbolos de operadores como +, *, y similares.
- (c) Signos de puntuación como paréntesis, comas, etc.
- (d) Los dígitos del 0 al 9.
- (e) Cadenas en **negrita** como id o if, donde cada una representa un solo símbolo terminal.

Los siguientes símbolos se consideran *no terminales*:

- (a) Letras mayúsculas al inicio del alfabeto, como *A*, *B*, *C*.
- (b) La letra *S*, que generalmente se utiliza como símbolo inicial.
- (c) Nombres en minúscula y en cursiva como *expr* o *stmt*.

Derivaciones

Una derivación es el proceso mediante el cual se generan cadenas a partir del símbolo inicial, utilizando las reglas de producción.

Hay dos tipos principales:

- **Derivación por la izquierda (leftmost)**: siempre se reemplaza el no terminal más a la izquierda.
- **Derivación por la derecha (rightmost)**: se reemplaza el más a la derecha.

Gramáticas en Pascal y C

El documento no entra a detalle con ejemplos específicos de Pascal o C, se menciona cómo las GLC permiten representar formalmente estructuras comunes de estos lenguajes, como sentencias condicionales (if, else) o expresiones matemáticas.

(Arriba de este documento en la investigación moral se menciona acerca de la gramática pascal, C y java).

Investigación de técnicas de automatización de autómatas

1. Minimización de Estados

La minimización busca reducir la cantidad de estados de un autómata determinista (AFD), manteniendo exactamente el mismo lenguaje reconocido.

¿Por qué se minimiza?

- Para optimizar el rendimiento del autómata.
- Para reducir recursos computacionales.
- Para facilitar el análisis y comprensión del modelo.

¿Cómo se hace?

1. Eliminar estados no alcanzables desde el estado inicial.
2. Agrupar estados equivalentes (ver siguiente punto).
3. Construir un nuevo autómata con esos grupos como nuevos estados.

2. Combinación de Estados Equivalentes

Esta técnica es parte central de la minimización. Dos estados se consideran equivalentes si, para toda cadena de entrada, ambos conducen a aceptar o rechazar en los mismos casos.

Criterio: Si no hay forma de distinguir los estados por ninguna secuencia de símbolos, entonces se pueden fusionar en un solo estado.

Ejemplo: Si los estados q1 y q2 siempre se comportan igual ante cualquier entrada, se pueden reemplazar por un nuevo estado común.

Esto se aplica mediante métodos como:

- Tabla de distinción (o partición de pares).
- Algoritmo de Moore o de Hopcroft, que sistematizan el proceso.

3.Eliminación de Estados Inaccesibles:

Se eliminan los estados a los que no se puede llegar desde el estado inicial. Aunque no afectan el lenguaje, sí afectan el tamaño y claridad del autómata.

4.Determinización:

Convierte un autómata no determinista (AFND) en uno determinista (AFD). Se realiza mediante el algoritmo de subconjuntos. Es útil porque los AFD son más fáciles de implementar computacionalmente.

5.Autómata Completo:

A veces es necesario completar un AFD agregando un estado “trampa” o “pozo” que absorbe transiciones no definidas. Esto garantiza que cada símbolo de entrada tenga una transición definida desde cualquier estado.