Nombre:

Fernando David Solís reyes

Matrícula:

2017-5688

Asignatura:

Microcontroladores

Tarea:

Resumen del capítulo #9 introducción

a los lenguajes Formales

Fecha:

09/10/2025

Resumen del Capítulo IX: Introducción a los Lenguajes Formales

Los asuntos principales que abarcan a lo largo del capítulo son:
• 1. Ciencias y Lenguajes (Las Normas): El capítulo establece las definiciones de Lenguaje Formal (un grupo de símbolos y normas) y Gramática

(el conjunto de normas).en lo cual estos lenguajes se categorizan en tipos (Tipo 3 o Regular y Tipo 2 o Libre de Contexto), siendo esenciales para la creación de compiladores y traductores de código.

2. Los Aparatos Autómatas: Presenta Autómatas Finitos (AFD/AFN), esto son dispositivos conceptuales muy básicos (como las máquinas de estado que previamente citamos) empleados para identificar o crear palabras válidas de acuerdo a las normas de una gramática.

• 3. Machine Turing y Capacidad Computacional (El Límite): Finaliza con la Máquina de Turing, que es la representación teórica del modelo de máquina más potente que se conoce (el "super robot" teórico). Al analizar esta computadora y la Teoría de la Computabilidad, se examinan las fronteras de lo que las computadoras pueden y no pueden realizar.

Al haber analizado el contenido del libro este aborda la estructura y clasificación de los lenguajes utilizados para comunicarse con las computadoras, conocidos como lenguajes formales, contrastándolos con la complejidad del lenguaje natural.

Este muestre temas principales como:

1. Gramáticas y Lenguajes Formales

El lenguaje formal L(G) conjunto restringido de frases o "palabras" que se pueden generar utilizando un sistema de reglas específico que pueden formarse a partir de una Gramática G. La gramática se compone de cinco elementos clave:

Σ (Alfabeto): El conjunto de símbolos con el que se forman las palabras.

T (Terminales): Símbolos que aparecen en la palabra final.

N (No Terminales): Símbolos que se sustituyen para formar la palabra.

s (Símbolo Inicial): El punto de partida para la derivación de palabras.

c **(Composiciones o Reglas):** Las reglas que definen cómo se sustituyen los símbolos para estructurar palabras válidas en el lenguaje

Clasificación de Gramáticas (Jerarquía de Chomsky)

Las gramáticas se clasifican en tipos según la complejidad de sus reglas:

Tipo 3 (Regular): posee reglas muy sencillas de sustitución y está íntimamente relacionada con los Autómatas Finitos.

Tipo 2 (Libre de Contexto): Es una de las más utilizada para la elaboración de compiladores e intérpretes. Se representa por medio de árboles de derivación y la notación Backus-Naur (BNF).

Ejemplo de Gramática Libre de Contexto (GLC)

Vamos a crear un lenguaje muy simple para operaciones matemáticas básicas, como sumas y multiplicaciones, sin preocuparnos por el orden de las operaciones (solo para el ejemplo).

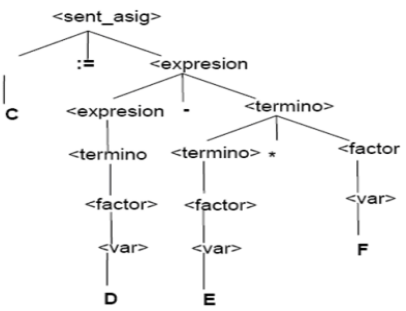
Gramática (G):

- **No Terminales (N):** `{<Expresión>, <Número>}` (Son conceptos que se van a expandir)
- **Terminales (T):** `{+, *, (, ), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}` (Son los símbolos finales que vemos)
- **Símbolo Inicial (S):** `<Expresión>` (Todo empieza con una expresión)
- **Reglas de Producción (P):**
    1. `<Expresión> ::= <Expresión> + <Expresión>` (Una expresión puede ser la suma de dos expresiones)
    2. `<Expresión> ::= <Expresión> * <Expresión>` (Una expresión puede ser la multiplicación de dos expresiones)
    3. `<Expresión> ::= ( <Expresión> )` (Una expresión puede estar entre paréntesis)
    4. `<Expresión> ::= <Número>` (Una expresión puede ser simplemente un número)
    5. `<Número> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9` (Un número es un dígito del 0 al 9)

¿Cómo se genera una palabra con esta gramática? (Derivación)

Vamos a generar la palabra "2 + 3":

1. Empezamos con el Símbolo Inicial: `<Expresión>`
2. Aplicamos la Regla 1: `<Expresión> + <Expresión>`
3. Aplicamos la Regla 4 al primer `<Expresión>`: `<Número> + <Expresión>`
4. Aplicamos la Regla 5 al `<Número>`: `2 + <Expresión>`
5. Aplicamos la Regla 4 al segundo `<Expresión>`: `2 + <Número>`
6. Aplicamos la Regla 5 al `<Número>`: `2 + 3`



Tipo 1 (Sensible al Contexto) y Tipo 0: Son las más complejas y de menor aplicación práctica en la programación diaria

2. Autómatas Finitos (AF)

Cuando se habla de los autómatas finitos son elementos teóricos generadores de palabras de un lenguaje. Son la mejor forma de representar gráficamente las gramáticas regulares. El capítulo detalla que:

- **Autómatas Finitos Determinísticos (AFD):** Donde para cada estado y cada símbolo de entrada, hay una única transición posible.
- **Autómatas Finitos No Determinísticos (AFN):** Donde puede haber múltiples transiciones para un mismo estado y símbolo.
- **Conversión:** Se explica el procedimiento para convertir un AFN a un AFD equivalente.

Máquinas de Estado Finito y Máquinas de Turing

El capítulo enlaza los autómatas con conceptos de la teoría de la computación:

- **Máquinas de Estado Finito (MEF):** Se establece la equivalencia entre un autómata finito y una máquina de estado finito.

una máquina de estado finito (MEF) como un juego de reglas muy simple que un robot, un juguete o un programa de computadora usa para saber qué hacer.

Ejemplo práctico-Imagina un **semáforo** 🚦. En cualquier momento, el semáforo solo puede estar en una de estas **tres situaciones o "estados"**:

1. 🟥**Rojo** (¡Alto!)
2. 🟨 **Amarillo** (¡Cuidado!)
3. 🟩 **Verde** (¡Adelante!)

Esa **situación** en la que está la máquina es su **estado**. Entonces esta Nunca puede estar en rojo y verde al mismo tiempo, ¿verdad? Solo puede estar en **un estado a la vez**.

- **Máquinas de Turing:** Este es el modelo teórico más potente, capaz de manejar cadenas y de representar cualquier algoritmo.

La Conversión AFN a AFD

El objetivo es tomar un autómata "flexible" (AFN, donde una entrada puede llevar a varios estados) y transformarlo en uno "estricto" (AFD, donde cada entrada lleva a un único estado) que funcione exactamente igual.

1. El Problema del AFN (La Flexibilidad)

Cuando un AFN, cuando lees un símbolo, la máquina puede ir a **más de un estado a la vez**. Esto es útil para diseñar, pero complicado de implementar en un programa, ya que una computadora solo puede estar en un estado a la vez.

2. La Solución del AFD (La Agrupación)

Es importante saber qué Para hacerlo determinístico, usamos un truco: tratamos un **grupo de estados** del AFN como si fuera un **solo y nuevo estado** en el AFD.

- **Paso Clave:** Se utiliza el **Conjunto Potencia P(E)** del AFN. Este conjunto potencia incluye **todas las combinaciones posibles** de los estados originales.
- **Fórmula:** Si el AFN tiene n estados, el AFD podría tener hasta 2n estados (pero en la práctica, generalmente usa solo un subconjunto de ellos).

3. El Proceso de Mapeo (Llenar la Tabla)

La conversión se hace creando una nueva **tabla de transiciones** para el AFD:

1. **Nuevos Estados:** Cada celda o grupo de la tabla del AFN (como {q0,q1}) se convierte en un **estado único** en el nuevo AFD.
2. **Transiciones:** Para saber a dónde va un nuevo estado (por ejemplo, el estado {q0,q1}) con un símbolo de entrada ('a' o 'b'), se mira a **dónde llevan esos símbolos a *todos los estados* que están dentro del grupo original** ({q0} y {q1}) en el AFN.
3. **Unión:** El nuevo destino es la **unión** de todos los posibles estados a los que se podía llegar en el AFN original.

En el ejemplo proporcionado: Cuando el autómata está en el estado **{q0,q1}** y lee '**b**':

- Desde q0 con 'b' se va a {q0}.
- Desde q1 con 'b' se va a {q1,q2}.
- El nuevo estado de destino en el AFD es la unión de ambos: $\{q0\} \cup \{q1,q2\} = \{q0,q1,q2\}$.

En la cual, de esta manera, se crea un autómata AFD que **simula perfectamente** todas las posibilidades del AFN, pero de una manera **determinística** (con una sola transición definida para cada estado y símbolo).

Teoría de la Computabilidad

Al final del capitulose abarca, sobre la **Teoría de la Computabilidad** y la **Teoría de la Complejidad**. La primera examina los problemas que se pueden o no resolver por medio de una computadora (problemas computables), mientras que la segunda se enfoca en la eficiencia y los recursos necesarios para resolver dichos problemas.

Ejemplo Teoría de la Computabilidad (¿Se puede resolver?)

Esta teoría se pregunta si existe algún algoritmo (o sea, una serie de pasos) que pueda resolver el problema, sin importar cuánto tiempo o cuánta memoria tarde.

Ejemplo: El Problema de las Tres Reinas 👑

Problema: ¿Es posible colocar tres reinas en un tablero de ajedrez de 3x3 de tal manera que ninguna reina pueda atacar a otra?

Análisis de Computabilidad:

Pregunta Clave: ¿Existe un algoritmo que pueda, dándole un tablero y un número N de reinas, decir si existe una solución?

Respuesta (para este caso): Sí, el problema es computable.

Podemos crear un algoritmo que pruebe todas las posibles combinaciones de colocación de las tres reinas en el tablero de 3x3. Como el número de combinaciones es finito, el algoritmo siempre terminará y nos dará la respuesta ("Sí, se puede" o "No, no se puede").

Conclusión de la Computabilidad: Si el problema se puede resolver, se pasa a la siguiente fase: la Complejidad. (Si no se pudiera resolver, como el famoso "Problema de la Parada" de la Máquina de Turing, la Teoría de la Complejidad no sería necesaria)