Analizador de Oraciones Simples - Versión Simplificada

Proyecto 5: Analizador de Lenguaje Natural Simple

Índice

Descripción General

Objetivo del Proyecto

Este proyecto implementa un **Analizador de Lenguaje Natural Simple** basado en la **Teoría de Autómatas y Lenguajes Formales**. El sistema es capaz de:

- Identificar si una oración cumple con una estructura gramatical definida
- Reconocer componentes: **Sujeto + Verbo + Objeto/Complemento**
- Construir árboles de derivación gramatical
- Validar oraciones mediante un Autómata Finito Determinista (AFD)

Características Principales

- **Sin dependencias externas** (solo Python estándar)
- **Alfabeto limitado** (vocabulario predefinido de ~30 palabras)
- **AFD explícito** con tabla de transiciones clara
- **Árboles de derivación** en formato ASCII
- **Análisis paso a paso** del proceso de validación
- **Código educativo** (~370 líneas, fácil de entender)

¿Por qué una Versión Simplificada?

Esta versión prioriza la **claridad conceptual** sobre la robustez:

Aspecto	Versión Completa	Versión Simplificada

Aspecto	Versión Completa	Versión Simplificada
Dependencias	spaCy, matplotlib, networkx	Ninguna
Vocabulario	Infinito (todo el español)	Limitado (30 palabras)
Análisis	NLP profesional	Búsqueda en diccionarios
Curva de aprendizaje	Alta	Baja
Enfoque	Producción	**Educación/Teoría**

Fundamentos Teóricos

1. Lenguajes Formales

Definición

Un **lenguaje formal** es un conjunto de cadenas formadas por símbolos de un alfabeto finito, que cumplen con reglas gramaticales específicas.

Componentes:

- **Alfabeto (Σ)**: Conjunto finito de símbolos
- **Cadena**: Secuencia finita de símbolos del alfabeto
- **Lenguaje (L)**: Subconjunto de todas las cadenas posibles

En nuestro proyecto:

```
Σ = {DET, N, V, PRON, PREP, ADV}  (categorías gramaticales)
L = {w | w tiene estructura SN + SV}
```

Jerarquía de Chomsky

Nuestro lenguaje pertenece a la **Clase 2: Lenguajes Independientes del Contexto (LIC)**

```
Tipo 0: Recursivamente enumerables (Máquina de Turing)
Tipo 1: Sensibles al contexto (Autómata Lineal Acotado)
Tipo 2: Independientes del contexto (Autómata de Pila) ← NUESTRO PROYECTO
Tipo 3: Regulares (Autómata Finito)
```

---

2. Gramáticas Independientes del Contexto

Definición Formal

Una **gramática independiente del contexto (GIC)** es una 4-tupla:

```
G = (V, T, P, S)

Donde:
- V: Conjunto finito de símbolos no terminales
- T: Conjunto finito de símbolos terminales (alfabeto)
- P: Conjunto finito de producciones (reglas)
- S: Símbolo inicial (axioma)
```

Características de las GIC

1. **Producciones de la forma**: $A \rightarrow \alpha$
    - A es un símbolo no terminal
    - α es una cadena de terminales y/o no terminales
2. **Independencia del contexto**: La sustitución de A no depende de lo que está antes o después
3. **Múltiples producciones**: Un símbolo puede tener varias reglas alternativas

---

3. Autómatas Finitos Deterministas (AFD)

Definición Formal

Un **AFD** es una 5-tupla:

```
M = (Q, Σ, δ, q₀, F)

Donde:
- Q: Conjunto finito de estados
- Σ: Alfabeto de entrada
- δ: Q × Σ → Q (función de transición)
- q₀ ∈ Q: Estado inicial
- F ⊆ Q: Conjunto de estados de aceptación
```

Propiedades del AFD

1. **Determinista**: Para cada par (estado, símbolo) existe exactamente una transición
2. **Finito**: Número fijo de estados
3. **Reconocedor**: Acepta o rechaza cadenas del lenguaje

Funcionamiento

1. Inicia en el estado q_0
2. Lee símbolo por símbolo de izquierda a derecha
3. Cambia de estado según la función δ

4. Si termina en un estado de F → **ACEPTA**
5. Si no → **RECHAZA**

---

4. Árboles de Derivación

Definición

Un **árbol de derivación** (o árbol sintáctico) es una representación gráfica de cómo se genera una cadena a partir de la gramática.

Propiedades:

- **Raíz**: Símbolo inicial (S)
- **Nodos internos**: Símbolos no terminales
- **Hojas**: Símbolos terminales (palabras)
- **Ramas**: Aplicación de reglas de producción

Ejemplo Visual

Para la oración: "el gato come pescado"

```
        S
       / \
     SN   SV
    / \   / \
  DET  N  V  N
   |   |  |  |
  el gato come pescado
```

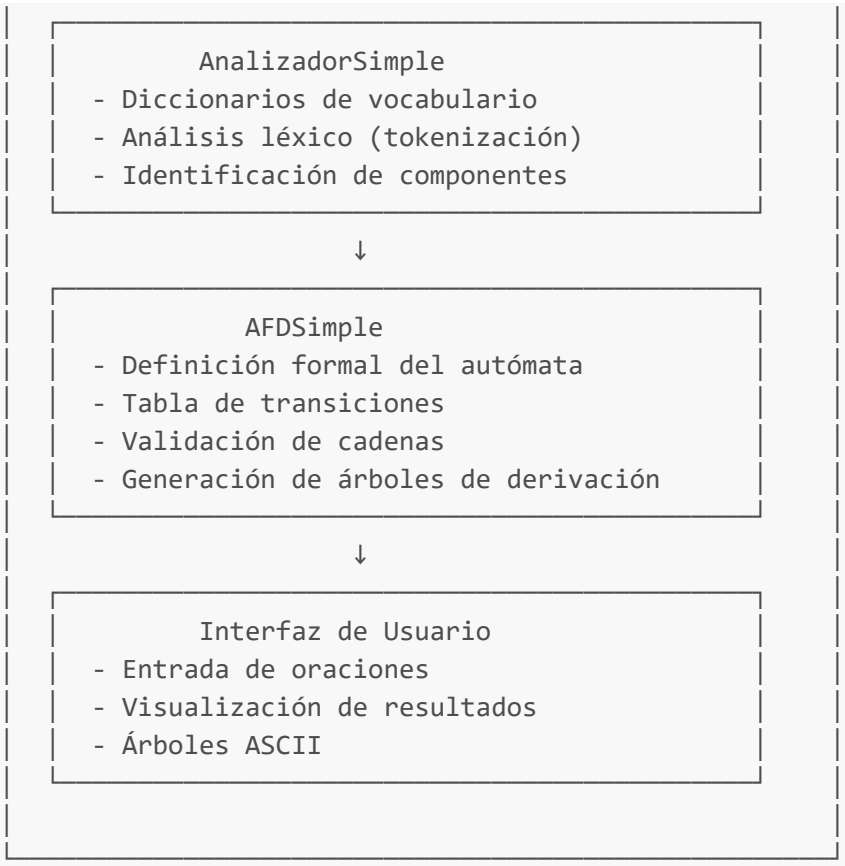
Derivación paso a paso:

```
S → SN SV                    (regla 1)
SN SV → DET N SV             (regla 2)
DET N SV → DET N V N         (regla 3)
DET N V N → el gato come pescado (sustitución terminal)
```

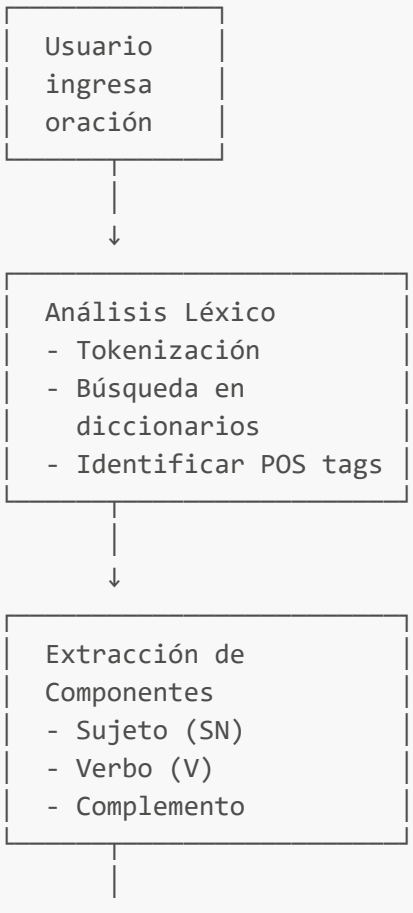
Arquitectura del Sistema

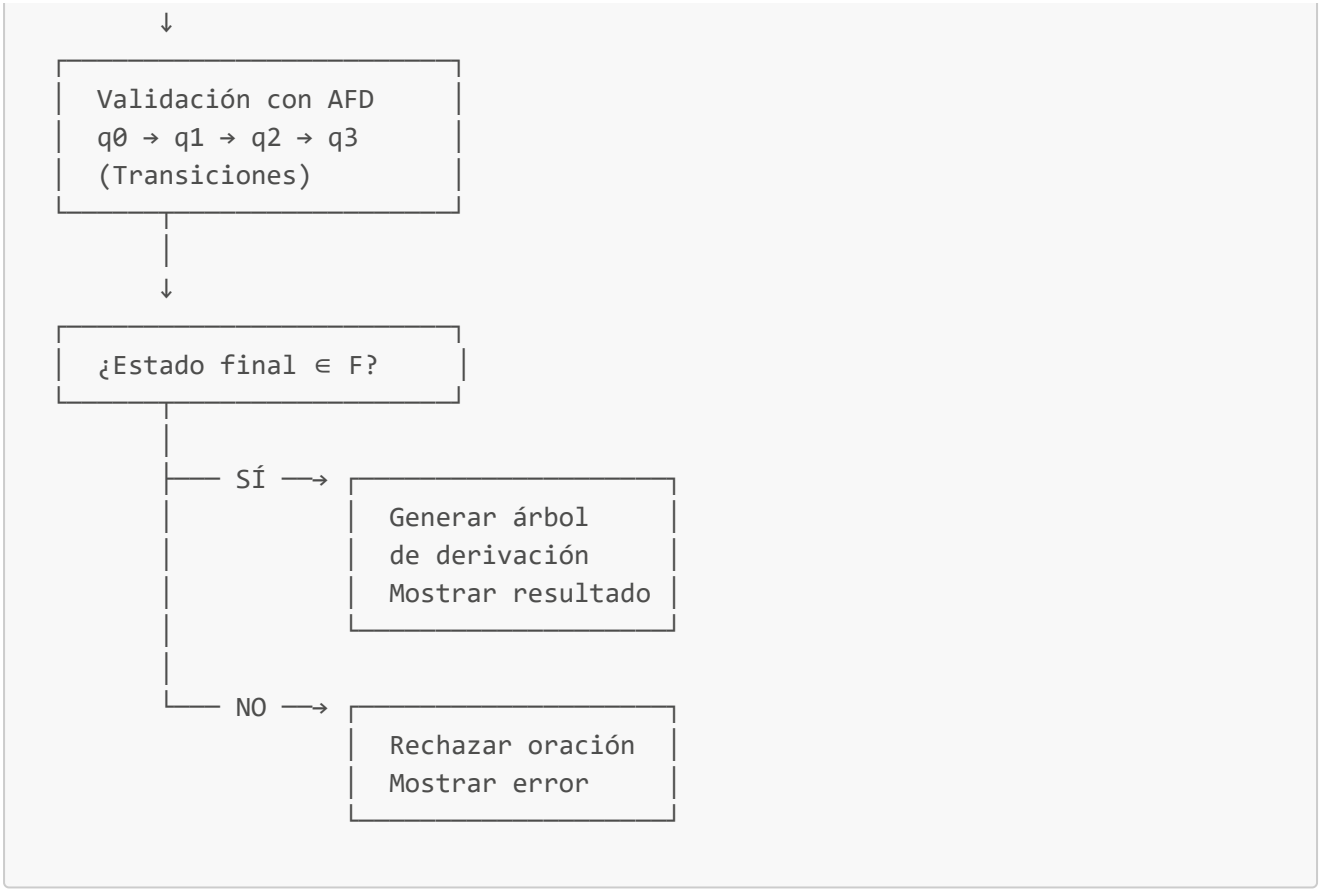
Diagrama de Componentes

```
┌──────────────────────────────────────────────────────┐
│                 version_simplificada.py              │
├──────────────────────────────────────────────────────┤
│                                                      │
```



Flujo de Ejecución





Gramática Formal

Definición Completa

```
G = (V, T, P, S)

V = {S, SN, SV, SP}        // Símbolos no terminales
T = {DET, N, V, PRON, PREP, ADV, palabras...}  // Símbolos terminales
S = S                      // Símbolo inicial

P = {                      // Producciones
    S  → SN SV
    SN → DET N
    SN → PRON
    SN → N
    SV → V SN
    SV → V SP
    SV → V ADV
    SV → V
    SP → PREP SN
}
```

Explicación de las Producciones

Regla	Descripción	Ejemplo
`S → SN SV`	Oración = Sintagma Nominal + Sintagma Verbal	"el gato come"
`SN → DET N`	SN con determinante y sustantivo	"el gato"
`SN → PRON`	SN formado por un pronombre	"yo"
`SN → N`	SN solo con sustantivo (nombre propio)	"María"
`SV → V SN`	Verbo transitivo con objeto directo	"come pescado"
`SV → V SP`	Verbo con sintagma preposicional	"camino por el parque"
`SV → V ADV`	Verbo con adverbio	"corre rápidamente"
`SV → V`	Verbo intransitivo	"corre"
`SP → PREP SN`	Preposición + sintagma nominal	"por el parque"

Alfabeto Terminal (Vocabulario)

El lenguaje tiene un **alfabeto limitado** que incluye:

```
# Determinantes (8 palabras)
DET = {el, la, un, una, los, las, mi, tu}

# Sustantivos (14 palabras)
N = {gato, perro, niño, niña, libro, parque, pescado,
     jardín, casa, María, Juan, hermano, matemáticas, niños}

# Verbos (9 palabras)
V = {come, corre, estudia, lee, camino, juega, juegan, escribe, canta}

# Pronombres (5 palabras)
PRON = {yo, tú, él, ella, nosotros}

# Preposiciones (5 palabras)
PREP = {por, en, de, con, a}

# Adverbios (4 palabras)
ADV = {rápidamente, bien, mal, rápido}

Total: 45 palabras en el vocabulario
```

Propiedades de la Gramática

1. **Ambigüedad**: Esta gramática es **no ambigua** (cada oración válida tiene un único árbol de derivación)

2. **Recursividad**: La gramática NO es recursiva (no hay ciclos como `SN → SN CONJ SN`)

3. **Tipo**: Gramática Independiente del Contexto (Tipo 2 en Jerarquía de Chomsky)
4. **Forma Normal**: No está en Forma Normal de Chomsky (FNC) ni en Forma Normal de Greibach (FNG)

Definición del Autómata

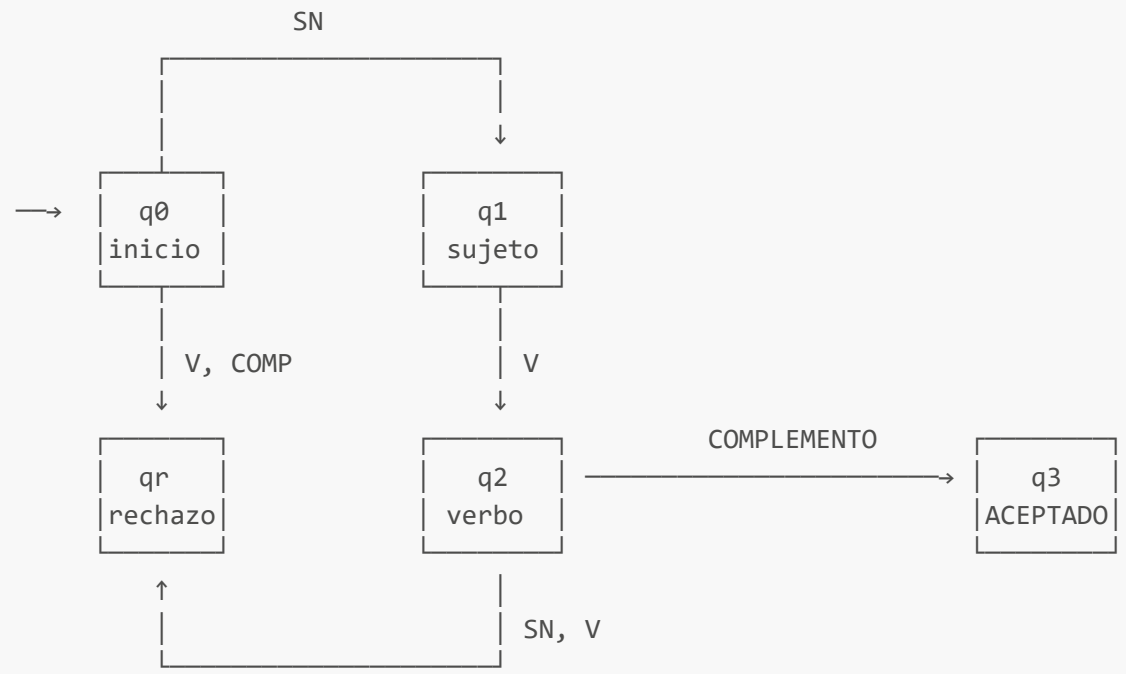
Especificación Formal del AFD

```
M = (Q, Σ, δ, q₀, F)

Q = {q0, q1, q2, q3, qr}     // 5 estados

Σ = {SN, V, COMPLEMENTO}     // Alfabeto de entrada (componentes gramaticales)

q₀ = q0                       // Estado inicial

F = {q3}                      // Estados de aceptación

δ: Función de transición definida por la tabla:
```

Tabla de Transiciones

Estado Actual	Entrada	Estado Siguiente	Descripción
q0	SN	**q1**	Sujeto identificado
q0	V	**qr**	Error: falta sujeto
q0	COMPLEMENTO	**qr**	Error: estructura incompleta
q1	V	**q2**	Verbo identificado
q1	SN	**qr**	Error: dos sujetos
q1	COMPLEMENTO	**qr**	Error: falta verbo
q2	COMPLEMENTO	**q3**	Predicado completo ✓
q2	SN	**qr**	Error: estructura incorrecta
q2	V	**qr**	Error: dos verbos
q3	*	**qr**	Ya se aceptó, no más entrada
qr	*	**qr**	Estado de rechazo (trampa)

Diagrama de Estados



Descripción de Estados

Estado	Nombre	Tipo	Significado
q0	Inicial	Normal	Esperando inicio de oración
q1	Sujeto	Normal	Sintagma nominal identificado
q2	Verbo	Normal	Verbo principal encontrado
q3	Aceptación	**Final**	Oración válida completa
qr	Rechazo	Trampa	Estructura inválida

Implementación en Código

```
# Definición formal del AFD
class AFDSimple:
    def __init__(self):
        # Conjunto de estados
        self.Q = ['q0', 'q1', 'q2', 'q3', 'qr']

        # Estado inicial
        self.q0 = 'q0'

        # Estados de aceptación
        self.F = ['q3']

        # Función de transición (tabla)
        self.delta = {
            ('q0', 'SN'): 'q1',
```

```
            ('q1', 'V'):  'q2',
            ('q2', 'COMPLEMENTO'): 'q3',
            # Transiciones implícitas a qr para casos no definidos
        }
```

Propiedades del Autómata

1. **Determinismo**: Para cada par (estado, símbolo) hay exactamente una transición
2. **Completitud**: Todas las entradas no definidas van a qr (trampa)
3. **Minimalidad**: No hay estados equivalentes que se puedan fusionar
4. **Conexión**: Todos los estados son alcanzables desde q0

Instalación y Uso

Requisitos Previos

- **Python 3.7 o superior**
- **Sistema operativo**: Windows, macOS, o Linux
- **Sin dependencias externas** (solo biblioteca estándar de Python)

Instalación

```
# 1. Clonar o descargar el proyecto
cd Proyecto-AFD-Oraciones

# 2. No se requiere instalación de dependencias
# El archivo es completamente independiente

# 3. Verificar instalación de Python
python --version
# o
python3 --version
```

Ejecución del Programa

```
# Ejecutar el analizador
python version_simplificada.py

# O en algunos sistemas
python3 version_simplificada.py
```

Interfaz de Usuario

Al ejecutar el programa verás:

```
╔═══════════════════════════════════════════════════════════════╗
║              VERSIÓN SIMPLIFICADA - AFD                       ║
║                   Analizador de Oraciones                     ║
╚═══════════════════════════════════════════════════════════════╝

Características:
  - Sin dependencias externas (no requiere spaCy)
  - Vocabulario limitado (alfabeto finito)
  - AFD explícito con tabla de transiciones
  - Árbol de derivación ASCII

----------------------------------------------------------------------
VOCABULARIO DISPONIBLE:
----------------------------------------------------------------------
Determinantes: el, la, un, una, los, las, mi, tu
Sustantivos: gato, perro, niño, niña, libro, parque, pescado, jardín...
Verbos: come, corre, estudia, lee, camino, juega, juegan, escribe...
Pronombres: yo, tú, él, ella, nosotros
Preposiciones: por, en, de, con, a

----------------------------------------------------------------------
EJEMPLOS DE ORACIONES VÁLIDAS:
----------------------------------------------------------------------
  - el gato come pescado
  - yo camino por el parque
  - María estudia matemáticas
  - los niños juegan en el jardín

----------------------------------------------------------------------
Ingresa una oración para analizar
(o escribe 'salir' para terminar)
----------------------------------------------------------------------

Oración: _
```

Ejemplos y Casos de Prueba

Caso 1: Oración Válida Simple

Entrada: `el gato come pescado`

Salida:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'el gato come pescado'
```

```
------------------------------------------------------------------------
FASE 1: Análisis léxico
------------------------------------------------------------------------

Palabras reconocidas:
  'el' → DET
  'gato' → N
  'come' → V
  'pescado' → N

------------------------------------------------------------------------
FASE 2: Transiciones del AFD
------------------------------------------------------------------------

  q0 → q1
  Razón: Sujeto identificado: 'el gato'

  q1 → q2
  Razón: Verbo identificado: 'come'

  q2 → q3
  Razón: Predicado completo: 'come pescado'

========================================================================
RESULTADO FINAL
========================================================================

Estado final: q3
¿Es estado de aceptación?: SÍ

✓ ORACIÓN ACEPTADA
  Sujeto: el gato
  Verbo: come
  Predicado: come pescado

------------------------------------------------------------------------
ÁRBOL DE DERIVACIÓN:
------------------------------------------------------------------------

S (Oración)
├── SN (Sintagma Nominal)
│   ├── DET → 'el'
│   └── N → 'gato'
└── SV (Sintagma Verbal)
    ├── V → 'come'
    └── N → 'pescado'

------------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
------------------------------------------------------------------------
1. S → SN + SV
```

```
2. SN → DET + N
3. SV → V + complemento
```

Caso 2: Oración con Pronombre

Entrada: `yo camino por el parque`

Análisis:

- **Sujeto**: "yo" (PRON)
- **Verbo**: "camino" (V)
- **Complemento**: "por el parque" (SP)

Árbol de derivación:

```
S (Oración)
├── SN (Sintagma Nominal)
│   └── PRON → 'yo'
└── SV (Sintagma Verbal)
    ├── V → 'camino'
    ├── PREP → 'por'
    ├── DET → 'el'
    └── N → 'parque'
```

Reglas aplicadas:

1. S → SN + SV
2. SN → PRON
3. SV → V + complemento

Caso 3: Oración con Nombre Propio

Entrada: `María estudia matemáticas`

Análisis:

- **Sujeto**: "María" (N - nombre propio)
- **Verbo**: "estudia" (V)
- **Objeto**: "matemáticas" (N)

Árbol de derivación:

```
S (Oración)
├── SN (Sintagma Nominal)
│   └── N → 'María'
└── SV (Sintagma Verbal)
```

```
    ├── V → 'estudia'
    └── N → 'matemáticas'
```

Reglas aplicadas:

1. S → SN + SV
2. SN → N
3. SV → V + complemento

Caso 4: Oración Inválida - Sin Sujeto

Entrada: `por el parque`

Salida:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'por el parque'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'por' → PREP
  'el' → DET
  'parque' → N

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → qr
  Razón: No se identificó sujeto

======================================================================
RESULTADO FINAL
======================================================================

Estado final: qr
¿Es estado de aceptación?: NO

✗ ORACIÓN RECHAZADA
```

Explicación: La oración no tiene sujeto ni verbo, solo un sintagma preposicional.

Caso 5: Oración Inválida - Sin Verbo

Entrada: `el gato pescado`

Análisis:

- Palabras: [el, gato, pescado]
- Estructura detectada: DET + N + N
- **Problema:** Falta el verbo

Salida:

```
  q0 → q1
  Razón: Sujeto identificado: 'el gato'

  q1 → qr
  Razón: No se identificó verbo

✗ ORACIÓN RECHAZADA
```

Caso 6: Palabra No Reconocida

Entrada: `el dinosaurio come pescado`

Análisis:

- Palabra "dinosaurio" NO está en el vocabulario
- El analizador la marca como `?` (desconocida)

Salida:

```
Palabras reconocidas:
  'el' → DET
  'dinosaurio' → ?
  'come' → V
  'pescado' → N

✗ ORACIÓN RECHAZADA
```

Explicación: El alfabeto es limitado, solo reconoce las palabras predefinidas.

Tabla de Casos de Prueba

#	Oración	Válida	Razón
1	`el gato come pescado`	✅	DET+N+V+N (estructura correcta)

#	Oración	Válida	Razón
2	`yo camino por el parque`	✅	PRON+V+PREP+DET+N
3	`María estudia matemáticas`	✅	N+V+N (nombre propio)
4	`los niños juegan`	✅	DET+N+V (verbo intransitivo)
5	`por el parque`	❌	Sin sujeto ni verbo
6	`el gato pescado`	❌	Sin verbo
7	`come pescado`	❌	Sin sujeto explícito
8	`el dinosaurio come`	❌	Palabra no en vocabulario

Análisis de Complejidad

Complejidad Temporal

Análisis Léxico

```
def analizar_oracion(self, oracion: str):
    palabras = oracion.split()  # O(n)

    for palabra in palabras:    # O(n)
        if palabra in self.sustantivos:  # O(1) con hash
            # procesar...
```

Complejidad: `O(n)` donde `n` es el número de palabras

Transiciones del AFD

```
# Máximo 3 transiciones fijas:
# q0 → q1 → q2 → q3
```

Complejidad: `O(1)` (constante)

Construcción del Árbol

```
def _imprimir_arbol_derivacion(self, analisis):
    # Recorre estructura una sola vez
    for elemento in analisis['estructura']:  # O(n)
        print(...)
```

Complejidad: `O(n)`

Complejidad Total: `O(n) + O(1) + O(n) = O(n)` **lineal**

Complejidad Espacial

Almacenamiento de Vocabulario

```
self.determinantes = [...]  # 8 palabras
self.sustantivos = [...]    # 14 palabras
self.verbos = [...]         # 9 palabras
# Total: ~45 palabras
```

Espacio: `O(1)` (constante, vocabulario fijo)

Estructura de Análisis

```
resultado = {
    'estructura': [(cat, palabra) for palabra in oracion]  # O(n)
}
```

Espacio: `O(n)` para almacenar la estructura

Complejidad Espacial Total: `O(n)` **lineal**

Comparación de Rendimiento

Operación	Tiempo	Espacio	Notas
Tokenización	O(n)	O(n)	Split de cadena
Búsqueda en diccionario	O(1)	O(1)	Hash lookup
Transiciones AFD	O(1)	O(1)	Máximo 3 pasos
Generación árbol	O(n)	O(n)	Una pasada
Total	**O(n)**	**O(n)**	Óptimo

Limitaciones del Modelo

1. Vocabulario Finito

Problema: Solo reconoce 45 palabras predefinidas

Ejemplo:

```
✅ "el gato come pescado"     (todas las palabras están)
❌ "el elefante come plátanos" (palabras no reconocidas)
```

Solución teórica: Ampliar diccionarios o usar morfología

---

2. Sin Análisis Semántico

Problema: No valida significado, solo estructura

Ejemplo:

```
✅ "el parque come libro"   (gramaticalmente correcto)
   (pero semánticamente absurdo)
```

Solución teórica: Agregar reglas de selección semántica

---

3. Conjugaciones Verbales

Problema: No reconoce variaciones de verbos

Ejemplo:

```
✅ "el gato come"     (forma registrada)
❌ "el gato comió"    (pretérito no registrado)
❌ "el gato comerá"   (futuro no registrado)
```

Solución teórica: Lematización o diccionario expandido

---

4. Oraciones Compuestas

Problema: No maneja coordinación ni subordinación

Ejemplo:

```
❌ "el gato come y el perro corre"  (coordinada)
❌ "yo camino porque hace sol"      (subordinada)
```

Solución teórica: Extender gramática con conectores

---

5. Orden Flexible

Problema: El español permite flexibilidad de orden

Ejemplo:

```
✅ "el gato come pescado"   (SVO - aceptado)
❌ "pescado come el gato"   (OVS - no aceptado)
   (aunque es válido en español coloquial)
```

Solución teórica: Gramática más permisiva con análisis de dependencias

---

Escalabilidad

Vocabulario

- **Actual:** 45 palabras
- **Escalable a:** ~500 palabras manteniendo O(1) en búsqueda
- **Límite práctico:** Sin límite con estructuras hash

Longitud de Oraciones

- **Actual:** Óptimo para oraciones de 3-10 palabras
- **Escalable a:** Cualquier longitud (complejidad lineal)
- **Límite práctico:** Memoria disponible

Extensiones Posibles

1. Añadir Más Categorías Gramaticales

```
# Nuevas categorías
self.adjetivos = ['grande', 'pequeño', 'rojo', 'bonito']
self.conjunciones = ['y', 'o', 'pero']

# Nueva regla
SN → DET ADJ N  # "el gato bonito"
```

2. Soporte para Oraciones Compuestas

```
# Nueva gramática
S → S CONJ S     # Coordinación
S → S CONJ_SUB S # Subordinación
```

3. Análisis de Concordancia

```
# Validar concordancia número/género
def validar_concordancia(det, sustantivo):
    if det == 'el' and sustantivo not in sustantivos_masc:
        return False
```

4. Modo Interactivo con Sugerencias

```
# Sugerir correcciones
if palabra not in vocabulario:
    sugerencia = buscar_similar(palabra, vocabulario)
    print(f"¿Quisiste decir '{sugerencia}'?")
```

Referencias

Libros Fundamentales

1. **Hopcroft, J. E., Motwani, R., & Ullman, J. D.** (2006).
   Introduction to Automata Theory, Languages, and Computation (3rd ed.).
   Pearson Education.
   ISBN: 978-0321455369

2. **Sipser, M.** (2012).
   Introduction to the Theory of Computation (3rd ed.).
   Cengage Learning.
   ISBN: 978-1133187790

3. **Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D.** (2006).
   Compilers: Principles, Techniques, and Tools (2nd ed.).
   Pearson Education.
   ISBN: 978-0321486813

Artículos y Recursos Online

- **Teoría de Autómatas**
  https://en.wikipedia.org/wiki/Automata_theory

- **Gramáticas Independientes del Contexto**
  https://en.wikipedia.org/wiki/Context-free_grammar

- **Jerarquía de Chomsky**
  https://es.wikipedia.org/wiki/Jerarqu%C3%ADa_de_Chomsky

- **Árboles de Derivación**
  https://www.tutorialspoint.com/automata_theory/derivation_tree.htm

Cursos Recomendados

- **Stanford CS143**: Compilers
  https://web.stanford.edu/class/cs143/
- **MIT 6.045J**: Automata, Computability, and Complexity
  https://ocw.mit.edu/courses/6-045j-automata-computability-and-complexity-spring-2011/

Conclusiones

Logros del Proyecto

Implementación correcta de un AFD funcional
Gramática independiente del contexto bien definida
Árboles de derivación generados correctamente
Análisis léxico basado en vocabulario limitado
Código educativo claro y comprensible

Aplicación de Conceptos Teóricos

Este proyecto demuestra exitosamente:

1. **Teoría de Lenguajes Formales**
   - Definición de alfabeto finito
   - Construcción de gramática tipo 2 (LIC)
   - Derivaciones y árboles sintácticos
2. **Teoría de Autómatas**
   - AFD con 5 estados
   - Función de transición determinista
   - Estados de aceptación y rechazo
3. **Análisis Sintáctico**
   - Reconocimiento de estructura SVO
   - Validación de cadenas del lenguaje
   - Construcción de representaciones arbóreas

Valor Educativo

- **Claridad conceptual** sobre complejidad técnica
- **Enfoque pedagógico** en teoría de computación
- **Código legible** para aprendizaje
- **Base sólida** para extensiones futuras

Autores

Ricardo Méndez
Emiliano Ledesma
Diego Jiménez
Abraham Velázquez

Última actualización: Noviembre 2024
Versión: 1.0 (Versión Simplificada)
