

Reporte Técnico: Versión Simplificada

Analizador de Oraciones con AFD - Versión Educativa

Proyecto: Analizador de Lenguaje Natural Simple
Versión: Simplificada (Sin dependencias externas)
Autores: Ricardo Méndez, Emiliano Ledesma
Fecha: Noviembre 2024

Resumen Ejecutivo

Este reporte presenta la **versión simplificada** del analizador de oraciones, diseñada específicamente con fines educativos. El sistema implementa un Autómata Finito Determinista (AFD) que valida oraciones simples en español sin dependencias externas, utilizando únicamente la biblioteca estándar de Python.

Características principales:

- 370 líneas de código educativo
- Sin dependencias (solo Python estándar)
- Vocabulario limitado de 45 palabras
- Gramática formal explícita
- AFD con 5 estados
- Árboles de derivación en ASCII

1. Introducción

1.1 Motivación del Diseño Simplificado

La versión simplificada fue diseñada para:

1. **Enfoque educativo:** Priorizar claridad conceptual sobre robustez técnica
2. **Accesibilidad:** Eliminar barreras de instalación y configuración
3. **Transparencia:** Código fácil de entender y modificar
4. **Teoría aplicada:** Demostrar directamente conceptos de autómatas y gramáticas

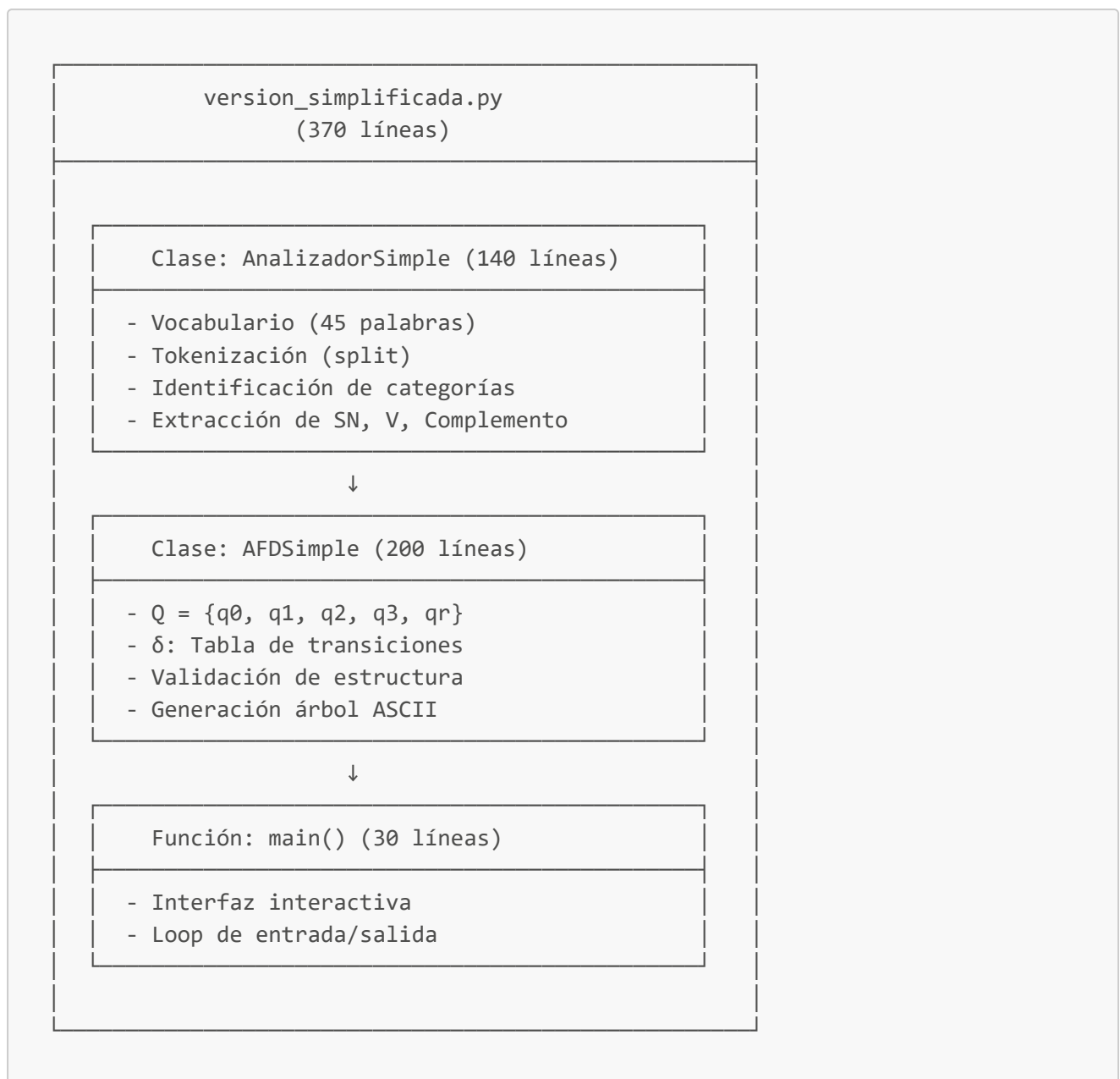
1.2 Diferencias con la Versión Completa

Aspecto	Versión Completa	Versión Simplificada
Código	1,115 líneas (4 archivos)	370 líneas (1 archivo)
Dependencias	spaCy + matplotlib + networkx	Ninguna
Instalación	5+ minutos	Instantánea
Vocabulario	Ilimitado (español completo)	45 palabras

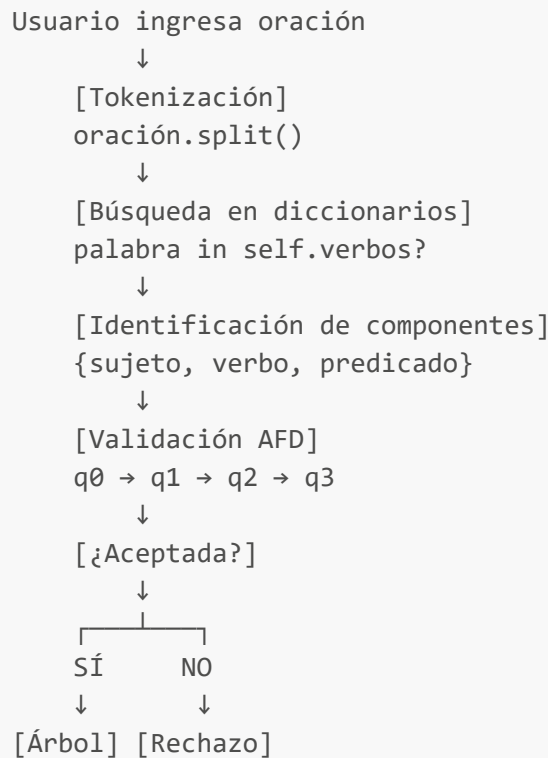
Aspecto	Versión Completa	Versión Simplificada
Análisis	NLP profesional (POS tagging)	Diccionarios simples
Visualización	PNG a color	ASCII en consola
Complejidad	Alta (múltiples módulos)	Baja (2 clases)
Enfoque	Producción	Educación

2. Arquitectura del Sistema

2.1 Visión General



2.2 Flujo de Datos



3. Componentes del Sistema

3.1 Clase AnalizadorSimple

Responsabilidades

- Mantener el vocabulario del lenguaje (alfabeto)
- Tokenizar oraciones (separar en palabras)
- Identificar categorías gramaticales por búsqueda en diccionarios
- Extraer componentes: sujeto, verbo, complemento

Atributos

```
self.determinantes: List[str]    # 8 palabras
    ['el', 'la', 'un', 'una', 'los', 'las', 'mi', 'tu']

self.sustantivos: List[str]      # 14 palabras
    ['gato', 'perro', 'niño', 'niña', 'libro', 'parque',
     'pescado', 'jardín', 'casa', 'María', 'Juan', 'hermano',
     'matemáticas', 'niños']

self.verbos: List[str]          # 9 palabras
    ['come', 'corre', 'estudia', 'lee', 'camino', 'juega',
     'juegan', 'escribe', 'canta']

self.pronombres: List[str]      # 5 palabras
```

```
['yo', 'tú', 'él', 'ella', 'nosotros']

self.preposiciones: List[str]    # 5 palabras
['por', 'en', 'de', 'con', 'a']

self.adverbios: List[str]        # 4 palabras
['rápidamente', 'bien', 'mal', 'rápido']
```

Método Principal: analizar_oracion()

Algoritmo:

```
1. Tokenizar: palabras = oracion.split()
2. Inicializar resultado
3. i = 0

4. Identificar SUJETO (SN):
  - Si palabra[i] es determinante:
    sujeto = determinante + sustantivo
  - Si palabra[i] es pronombre:
    sujeto = pronombre
  - Si palabra[i] es sustantivo:
    sujeto = sustantivo

5. Identificar VERBO (V):
  - Si palabra[i] está en self.verbos:
    verbo = palabra[i]

6. Identificar COMPLEMENTO:
  - complemento = resto de palabras

7. Validar: si sujeto Y verbo → válida = True
8. Retornar resultado
```

Complejidad: $O(n)$ donde n = número de palabras

3.2 Clase AFDSimple

Responsabilidades

- Definir formalmente el autómata ($Q, \Sigma, \delta, q_0, F$)
- Procesar oraciones mediante transiciones de estado
- Validar si la estructura es aceptada
- Generar árboles de derivación en formato ASCII

Definición Formal del AFD

```

# 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',          # Sujeto encontrado
    ('q1', 'V'): 'q2',          # Verbo encontrado
    ('q2', 'COMPLEMENTO'): 'q3', # Predicado completo
}
# Cualquier transición no definida → qr (rechazo)

```

Método Principal: procesar_oracion()

Algoritmo:

```

1. estado_actual = q0
2. analisis = AnalizadorSimple.analizar_oracion(oracion)

3. TRANSICIÓN 1: q0 → q1
   Si analisis.sujeto existe:
       estado_actual =  $\delta(q0, SN) = q1$ 
   Sino:
       estado_actual = qr
       RETORNAR rechazada

4. TRANSICIÓN 2: q1 → q2
   Si analisis.verbo existe:
       estado_actual =  $\delta(q1, V) = q2$ 
   Sino:
       estado_actual = qr
       RETORNAR rechazada

5. TRANSICIÓN 3: q2 → q3
   Si analisis.predicado existe:
       estado_actual =  $\delta(q2, COMPLEMENTO) = q3$ 
   Sino:
       estado_actual = qr
       RETORNAR rechazada

6. aceptada = (estado_actual ∈ F)
7. Si aceptada: generar_arbol_derivacion()
8. RETORNAR resultado

```

4. Análisis Detallado

4.1 Gramática del Lenguaje

Definición formal:

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

V = {S, SN, SV, SP}

T = {DET, N, V, PRON, PREP, ADV}
  ∪ {el, la, un, ..., gato, perro, ..., come, corre, ...}

P = {
  S → SN SV
  SN → DET N | PRON | N
  SV → V SN | V SP | V ADV | V
  SP → PREP SN
}

S = S
```

4.2 Tabla de Transiciones del AFD

Estado	Entrada SN	Entrada V	Entrada COMPLEMENTO	Otros
q0	q1 ✓	qr X	qr X	qr X
q1	qr X	q2 ✓	qr X	qr X
q2	qr X	qr X	q3 ✓	qr X
q3	qr X	qr X	qr X	qr X
qr	qr X	qr X	qr X	qr X

Leyenda: ✓ = Transición válida | X = Va a rechazo

4.3 Análisis de Complejidad

Temporal

Operación	Complejidad	Justificación
<code>oracion.split()</code>	$O(n)$	Recorre la cadena una vez
<code>palabra in diccionario</code>	$O(1)$ amortizado	Búsqueda en lista pequeña (~10 elementos)

Operación	Complejidad	Justificación
Identificar componentes	$O(n)$	Recorre lista de palabras
Transiciones AFD	$O(1)$	Máximo 3 transiciones
Generar árbol ASCII	$O(n)$	Recorre estructura una vez
Total	$O(n)$	Lineal en número de palabras

Espacial

Estructura	Espacio	Notas
Vocabulario (6 diccionarios)	$O(1)$	Constante: 45 palabras
Lista de palabras	$O(n)$	Proporcional a longitud
Estructura de análisis	$O(n)$	Lista de tuplas (cat, palabra)
Historial de estados	$O(1)$	Máximo 4 estados
Total	$O(n)$	Lineal

5. Casos de Uso

5.1 Caso Exitoso: Oración Simple

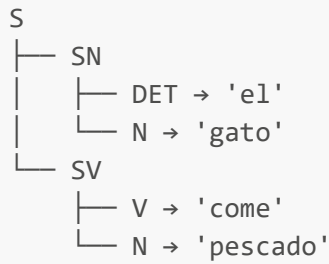
Entrada: "el gato come pescado"

Ejecución paso a paso:

- TOKENIZACIÓN:**
palabras = ['el', 'gato', 'come', 'pescado']
- ANÁLISIS LÉXICO:**
'el' → DET (determinante)
'gato' → N (sustantivo)
'come' → V (verbo)
'pescado' → N (sustantivo)
- EXTRACCIÓN DE COMPONENTES:**
sujeto = 'el gato' (DET + N)
verbo = 'come'
predicado = 'come pescado'
- VALIDACIÓN AFD:**
 $q_0 \rightarrow q_1$ (SN identificado)
 $q_1 \rightarrow q_2$ (V identificado)
 $q_2 \rightarrow q_3$ (COMPLEMENTO identificado)
- RESULTADO:**

Estado final: q3
¿Aceptada?: SÍ ($q3 \in F$)

6. ÁRBOL DE DERIVACIÓN:



5.2 Caso Fallido: Sin Verbo

Entrada: "el gato pescado"

Ejecución:

1. TOKENIZACIÓN:
palabras = ['el', 'gato', 'pescado']
2. ANÁLISIS LÉXICO:
'el' → DET
'gato' → N
'pescado' → N (NO es verbo)
3. EXTRACCIÓN:
sujeto = 'el gato'
verbo = None ⚠
predicado = None
4. VALIDACIÓN AFD:
 $q_0 \rightarrow q_1$ (SN identificado)
 $q_1 \rightarrow q_r$ (No hay verbo) ✗
5. RESULTADO:
Estado final: q_r
¿Aceptada?: NO ($q_r \notin F$)
Razón: "No se identificó verbo después del sujeto"

7. Referencias

Fundamentos Teóricos

[1] Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006).
Introduction to Automata Theory, Languages, and Computation.

[2] Sipser, M. (2012).
Introduction to the Theory of Computation.

Implementación

[3] Python Software Foundation. (2024).
Python Standard Library Documentation.
<https://docs.python.org/3/library/>

[4] Stack Overflow. (2024).
Python String Methods.
<https://stackoverflow.com/questions/tagged/python+string>

Apéndices

Apéndice A: Código Fuente Completo

Ver archivo: [version_simplificada.py](#)

Apéndice B: Diagramas

Ver archivo: [docs/README_VERSION_SIMPLIFICADA.md](#) - Sección 5

Apéndice C: Casos de Prueba

Ver archivo: [docs/README_VERSION_SIMPLIFICADA.md](#) - Sección 7

Fin del Reporte Técnico - Versión Simplificada

Versión del documento: 1.0

Fecha: Noviembre 2024

Autores: Ricardo Méndez, Emiliano Ledesma, Diego Jiménez y Abraham Velázquez
