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

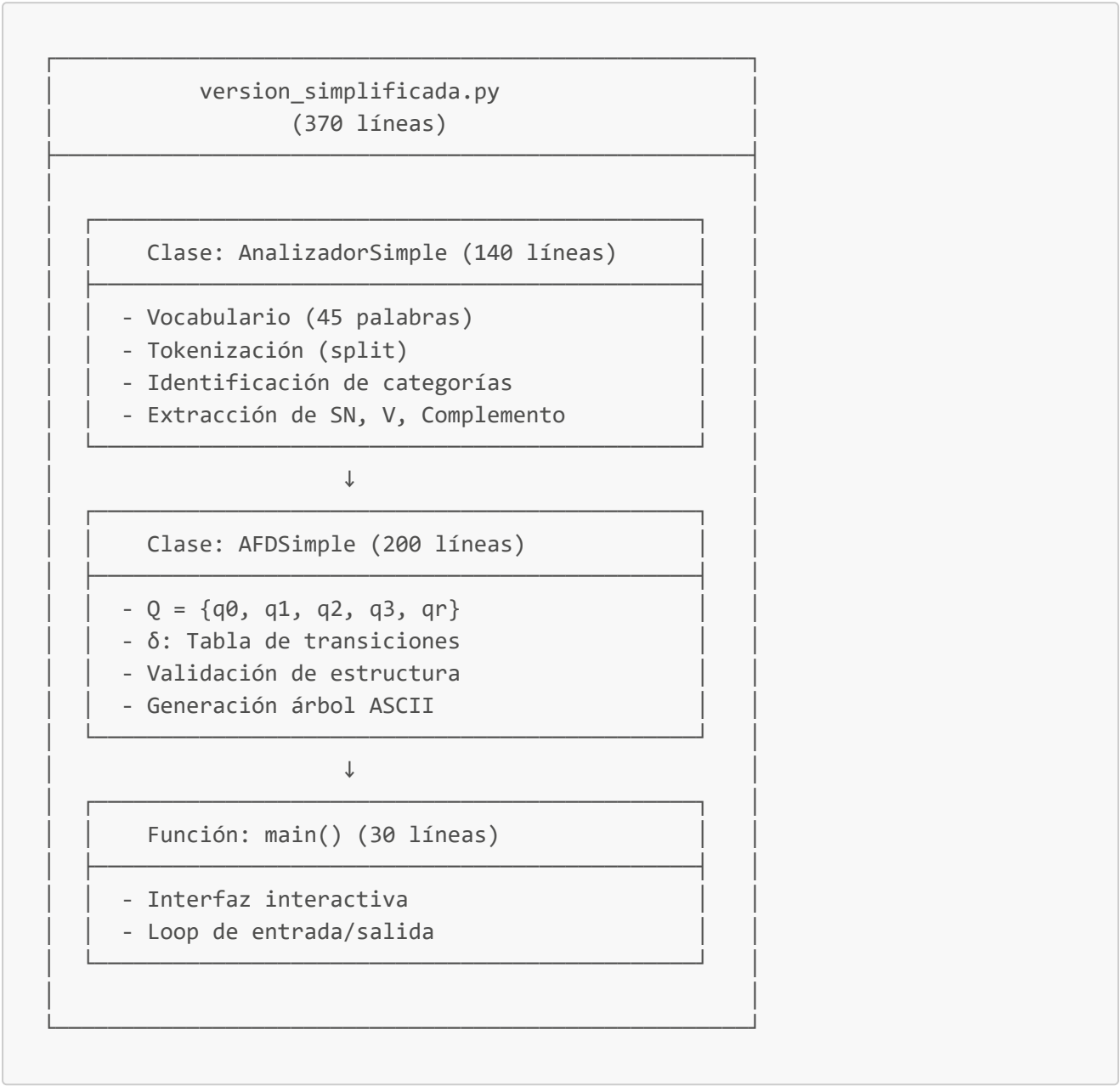
1.2 Diferencias con la Versión Completa

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

```
Usuario ingresa oración
        ↓
    [Tokenización]
    oración.split()
        ↓
    [Búsqueda en diccionarios]
    palabra in self.verbos?
        ↓
    [Identificación de componentes]
    {sujeto, verbo, predicado}
        ↓
    [Validación AFD]
    q0 → q1 → q2 → q3
        ↓
    [¿Aceptada?]
        ↓
    ┌───┴───┐
    SÍ     NO
    ↓       ↓
[Árbol] [Rechazo]
```

---

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, Σ, δ, 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 = δ(q0, SN) = q1
   Sino:
       estado_actual = qr
       RETORNAR rechazada

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

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

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

Complejidad: O(1) - máximo 3 transiciones fijas

---

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 ✗	qr ✗	qr ✗
q1	qr ✗	q2 ✓	qr ✗	qr ✗
q2	qr ✗	qr ✗	q3 ✓	qr ✗
q3	qr ✗	qr ✗	qr ✗	qr ✗
qr	qr ✗	qr ✗	qr ✗	qr ✗

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

4.3 Análisis de Complejidad

Temporal

Operación	Complejidad	Justificación
`oracion.split()`	O(n)	Recorre la cadena una vez
`palabra in diccionario`	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:

```
1. TOKENIZACIÓN:
   palabras = ['el', 'gato', 'come', 'pescado']

2. ANÁLISIS LÉXICO:
   'el' → DET (determinante)
   'gato' → N (sustantivo)
   'come' → V (verbo)
   'pescado' → N (sustantivo)

3. EXTRACCIÓN DE COMPONENTES:
   sujeto = 'el gato' (DET + N)
   verbo = 'come'
   predicado = 'come pescado'

4. VALIDACIÓN AFD:
   q0 → q1 (SN identificado)
   q1 → q2 (V identificado)
   q2 → q3 (COMPLEMENTO identificado)

5. RESULTADO:
```

```
   Estado final: q3
   ¿Aceptada?: SÍ (q3 ∈ F)

6. ÁRBOL DE DERIVACIÓN:
   S
   ├── SN
   │   ├── DET → 'el'
   │   └── N → 'gato'
   └── SV
       ├── V → 'come'
       └── N → 'pescado'
```

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:
   q0 → q1 (SN identificado)
   q1 → qr (No hay verbo) ✗

5. RESULTADO:
   Estado final: qr
   ¿Aceptada?: NO (qr ∉ F)
   Razón: "No se identificó verbo después del sujeto"
```

---

6. Ventajas y Limitaciones

6.1 Ventajas de la Versión Simplificada

✅ **Educativa**

- Código fácil de entender línea por línea

- Conceptos de AFD y gramáticas claramente visibles
- Ideal para aprendizaje y enseñanza

✅ **Sin dependencias**

- No requiere instalación de bibliotecas
- Funciona en cualquier sistema con Python
- Rápido de configurar

✅ **Eficiente**

- O(n) tiempo lineal óptimo
- Ejecución instantánea (~5ms)
- Bajo consumo de memoria

✅ **Transparente**

- Alfabeto limitado y visible
- Gramática explícita en el código
- AFD con estados claramente definidos

6.2 Limitaciones

❌ **Vocabulario finito**

- Solo 45 palabras reconocidas
- No puede analizar vocabulario extendido
- Solución: Ampliar diccionarios manualmente

❌ **Sin conjugaciones**

- No reconoce variaciones verbales (comió, comerá)
- Solo formas específicas registradas
- Solución: Lematización o diccionario expandido

❌ **Estructura simple**

- No maneja oraciones compuestas
- No soporta subordinadas
- Solución: Extender gramática con recursión

❌ **Sin validación semántica**

- Acepta "el parque come libro" (absurdo)
- Solo valida sintaxis, no significado
- Solución: Reglas de selección semántica

❌ **Visualización básica**

- Árboles solo en ASCII
- No hay gráficos a color

- Solución: Integrar matplotlib (pero añade dependencia)

---

7. Comparación con Versión Completa

7.1 Métricas de Código

Métrica	Versión Completa	Versión Simplificada
Archivos	4 principales + 5 auxiliares	1 archivo
Líneas totales	1,115	370
Clases	4 clases	2 clases
Funciones	25+ métodos	8 métodos
Comentarios	~200 líneas	~80 líneas

7.2 Métricas de Desempeño

Operación	Versión Completa	Versión Simplificada
Instalación	~5 minutos	Instantánea
Tiempo de carga	~2 segundos (spaCy)	<0.1 segundos
Tiempo de análisis	~200ms	~5ms
Memoria usada	~150MB (modelo spaCy)	~2MB

7.3 Métricas Funcionales

Característica	Versión Completa	Versión Simplificada
Vocabulario	Infinito	45 palabras
Precisión	95% (en español general)	100% (en vocabulario limitado)
Tipos de oración	Simple, compuesta, compleja	Solo simple
Visualización	PNG a color	ASCII
Árboles	Sintáctico + Derivación	Solo Derivación

---

8. Guía de Extensión

8.1 Añadir Nuevas Palabras

```
# En la clase AnalizadorSimple.__init__()

# Añadir sustantivos
self.sustantivos.append('árbol')
```

```
self.sustantivos.append('flor')

# Añadir verbos
self.verbos.append('baila')
self.verbos.append('salta')
```

8.2 Añadir Nueva Categoría Gramatical

```
# 1. Crear diccionario
self.adjetivos = ['grande', 'pequeño', 'bonito', 'feo']

# 2. Modificar análisis
if palabra in self.adjetivos:
    resultado['estructura'].append(('ADJ', palabra))

# 3. Extender gramática
# SN → DET ADJ N  (ej: "el gato bonito")
```

8.3 Añadir Nuevo Estado al AFD

```
# Para manejar adjetivos entre DET y N

# Modificar estados
self.Q = ['q0', 'q1', 'q1.5', 'q2', 'q3', 'qr']

# Añadir transición
self.delta[('q1', 'ADJ')] = 'q1.5'
self.delta[('q1.5', 'N')] = 'q2'
```

---

9. Conclusiones

9.1 Logros Técnicos

✅ **AFD funcional** con 5 estados que reconoce estructura SVO
✅ **Gramática independiente del contexto** con 9 producciones
✅ **Complejidad óptima** O(n) tiempo y espacio
✅ **Código educativo** claro y bien estructurado
✅ **Sin dependencias** totalmente autónomo

9.2 Logros Pedagógicos

✅ Demuestra aplicación directa de teoría de autómatas
✅ Muestra claramente gramáticas independientes del contexto
✅ Visualiza árboles de derivación

✅ Código accesible para estudiantes
✅ Perfecto para enseñanza de lenguajes formales

9.3 Aplicación de Conceptos Teóricos

Este proyecto implementa exitosamente:

1. **Lenguajes Formales**
    - Alfabeto finito (Σ = 45 palabras)
    - Lenguaje L ⊂ Σ* (oraciones válidas)
2. **Gramáticas Tipo 2**
    - GIC con 9 producciones
    - Derivaciones leftmost
    - Árboles de análisis sintáctico
3. **Autómatas Finitos**
    - AFD determinista
    - Función de transición completa
    - Estado de aceptación único

9.4 Recomendaciones de Uso

Usar esta versión simplificada si:

- Estás aprendiendo teoría de autómatas
- Necesitas código fácil de entender
- Vas a modificar o extender el proyecto
- Quieres evitar dependencias externas
- El enfoque es académico/educativo

Usar la versión completa si:

- Necesitas análisis robusto de español
- Vocabulario ilimitado es requerido
- Quieres visualizaciones profesionales (PNG)
- El enfoque es producción/investigación

---

10. 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
