UNIVERSIDAD DEL VALLE DE GUATEMALA

CC2019 – Teoría de la Computación

Sección 31

Ing. Gabriel Brolo

Implementación del Algoritmo CYK para Análisis Sintáctico

Juan Marcos Cruz Melara, 23110

Angie Nadissa Vela López, 23764

GUATEMALA, 12 de octubre de 2025

INTRODUCCIÓN

El presente proyecto implementa un sistema completo para el **análisis sintáctico de oraciones mediante el algoritmo CYK (Cocke–Younger–Kasami)**, utilizando gramáticas libres de contexto (CFG) y su correspondiente **conversión a Forma Normal de Chomsky (CNF)**.

El objetivo principal es demostrar la aplicación práctica de los conceptos teóricos de **lenguajes formales y autómatas** en el reconocimiento de estructuras lingüísticas, tanto en expresiones aritméticas como en oraciones en inglés.

La solución desarrollada permite:

- **Cargar** gramáticas libres de contexto desde archivos de texto.
- **Simplificarlas y convertirlas** automáticamente a CNF mediante una serie de transformaciones formales.
- **Aplicar el algoritmo CYK** para determinar si una secuencia de símbolos pertenece al lenguaje generado.
- **Reconstruir el árbol sintáctico** de derivación para oraciones válidas.
- **Visualizar y evaluar** el rendimiento del proceso mediante una interfaz de línea de comandos (CLI) y pruebas automatizadas.

Además, el proyecto incluye una **batería de pruebas unitarias** que aseguran la corrección de cada componente, garantizando que tanto la conversión a CNF como el análisis sintáctico con CYK conserven la validez del lenguaje original.

ÍNDICE

DISEÑO DE LA APLICACIÓN

Arquitectura general

La aplicación fue diseñada siguiendo una arquitectura modular en Python, con el objetivo de implementar dos procesos fundamentales de la teoría de lenguajes formales:

1. **Simplificación y conversión de una gramática a Forma Normal de Chomsky (CNF)**.
2. **Aplicación del algoritmo CYK (Cocke–Younger–Kasami)** para determinar si una oración pertenece al lenguaje generado por dicha gramática, además de construir el árbol sintáctico correspondiente.

El proyecto se organiza en varios módulos, cada uno con una responsabilidad específica y bien delimitada. Esto permite una estructura limpia, fácil de mantener y reutilizable.

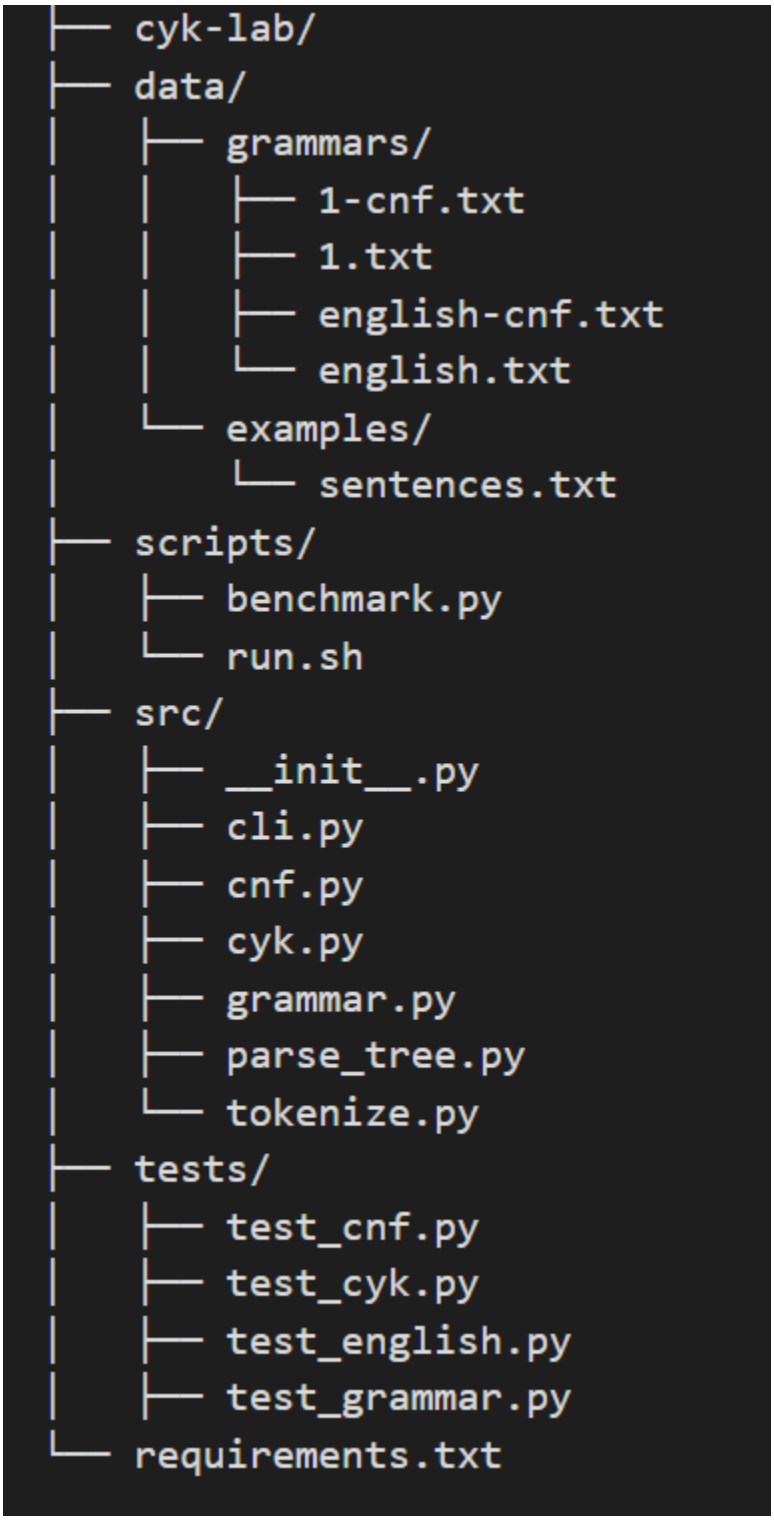
Video del funcionamiento

Link

Repositorio de GitHub

Link

Estructura del proyecto



Descripción de los módulos

1. benchmark.py

Este módulo implementa un **script de prueba y medición de rendimiento** del algoritmo CYK.
Su función principal (bench) carga una gramática, normaliza una lista de oraciones de prueba, y aplica el algoritmo CYK sobre cada una para medir el tiempo de ejecución en milisegundos.

- **Entradas:** ruta a la gramática en texto y una lista de frases.
- **Salida:** para cada oración, imprime si pertenece o no al lenguaje, junto con el tiempo de ejecución.
- **Uso:** permite evaluar el desempeño del algoritmo y verificar la corrección de la gramática.

2. run.sh

Este script proporciona una forma automatizada de **ejecutar pruebas rápidas** del sistema con ejemplos en consola.

Funcionamiento:

1. Activa el entorno virtual (.venv) del proyecto.
2. Ejecuta el análisis de una **expresión aritmética** en CNF, mostrando el árbol de derivación y el tiempo de ejecución.
3. Ejecuta el análisis de una **oración en inglés** normalizada, mostrando también su árbol sintáctico.
4. En caso de error en la segunda ejecución, el comando || true evita que el script se interrumpa.

Propósito:

- Servir como **demostración práctica del sistema completo**, combinando la CLI con el análisis sintáctico visual.
- Facilitar la validación final sin necesidad de ejecutar manualmente cada módulo.

3. cli.py

Este módulo define una **interfaz de línea de comandos (CLI)** para el usuario. Permite ejecutar el sistema de forma interactiva, especificando opciones como:

- --grammar: ruta de la gramática.
- --sentence: oración a analizar.
- --to-cnf: convierte la gramática a CNF antes de aplicar CYK.
- --tree: exporta el árbol sintáctico si la oración pertenece al lenguaje.
- --time: muestra el tiempo de procesamiento.
- --normalize: aplica normalización a la oración (útil para inglés).

La CLI carga la gramática, ejecuta el **pipeline de conversión a CNF** y el **algoritmo CYK**, mostrando un **"SÍ"** o **"NO"** según la pertenencia de la oración al lenguaje, y exportando el árbol si es válido.

4. cnf.py

Este módulo implementa **todas las etapas del proceso de simplificación y conversión de una gramática a Forma Normal de Chomsky (CNF)**.
Cada transformación se realiza de manera modular mediante funciones puras, sin alterar la gramática original.

Funciones principales:

- remove_epsilon(G): elimina producciones que generan ε (epsilon).
- remove_unit(G): elimina producciones unitarias del tipo A → B.

- remove_useless(G): elimina símbolos inútiles (no generadores o no alcanzables).
- terminals_to_unaries(G): reemplaza terminales en producciones largas por no terminales auxiliares.
- binarize(G): convierte todas las producciones en forma binaria (máx. dos símbolos).
- to_cnf_pipeline(G): ejecuta automáticamente todas las fases anteriores en orden.

Con este módulo, cualquier gramática libre de contexto puede transformarse en una versión equivalente en CNF, requisito esencial para aplicar el algoritmo CYK.

5. cyk.py

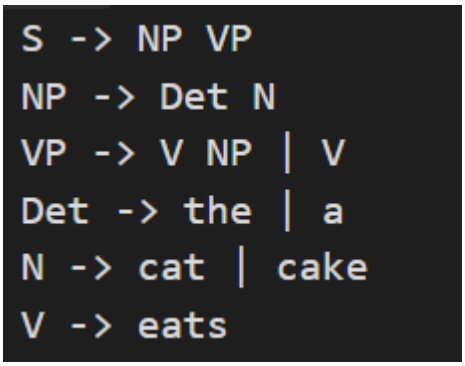
Contiene la implementación completa del **algoritmo CYK (Cocke–Younger–Kasami)**. Este algoritmo realiza un análisis sintáctico **bottom-up** utilizando **programación dinámica**, verificando si una oración puede ser generada por una gramática en CNF.

Funciones:

- invert_rules(P): crea estructuras de acceso rápido para reglas binarias y unarias.
- cyk_parse(G, words): aplica el algoritmo CYK sobre una lista de palabras y devuelve:
    - Un valor booleano (True/False) indicando si la oración pertenece al lenguaje.
    - La tabla de análisis sintáctico.
    - Un diccionario auxiliar (back) para reconstruir el árbol de derivación.

6. grammar.py

Gestiona la **carga y representación interna de las gramáticas** a partir de archivos de texto. Permite definir reglas en el formato:



Cada gramática se almacena como un diccionario con los conjuntos de **no terminales (N)**, **terminales (T)** y **producciones (P)**.

7. parse_tree.py

Implementa la **reconstrucción y visualización del árbol sintáctico** generado por el análisis CYK.

- build_tree(back, n, start_symbol): reconstruye el árbol a partir de la información guardada en la tabla CYK.
- export_tree(root): genera una imagen del árbol en formato **.png** usando **Graphviz** y la guarda en el directorio outputs/trees/.

Cada nodo representa un símbolo no terminal, y las hojas contienen los terminales (palabras de la oración).

8. tokenize.py

Módulo auxiliar que realiza la **normalización de las oraciones en inglés**.

- Convierte el texto a minúsculas.
- Elimina signos de puntuación (.,!?;:).
- Divide la oración en tokens.

Esto garantiza que la entrada sea consistente con los terminales definidos en la gramática.

Integración del Sistema

El flujo de ejecución completo sigue esta secuencia:

1. **Carga de gramática** (grammar.py).
2. **Conversión a CNF** (cnf.py → to_cnf_pipeline).
3. **Tokenización de la oración** (tokenize.py).
4. **Análisis sintáctico CYK** (cyk.py → cyk_parse).
5. **Visualización del resultado y árbol sintáctico** (parse_tree.py).
6. **Interacción con el usuario o benchmarking** (cli.py / benchmark.py).

Pruebas y validación del proyecto

La carpeta tests/ contiene un conjunto de **scripts de prueba automatizada** desarrollados con el framework pytest. Cada script verifica el correcto funcionamiento de los módulos principales del sistema (carga de gramáticas, conversión a CNF, ejecución de CYK, y validación del lenguaje inglés).

1. tests/test_grammar.py

Este script valida la **carga correcta de gramáticas** desde archivos externos.

```
from cyk.grammar import load_grammar
def test_load():
    G = load_grammar("data/grammars/1-cnf.txt")
    assert "E" in G["N"] and "id" in G["T"]
```

Propósito:

- Comprobar que el método load_grammar() identifica correctamente los **no terminales** y **terminales** definidos en el archivo.
- Verificar que la gramática se almacena en la estructura de datos esperada (diccionario con claves N, T, P, y S).

2. tests/test_cnf.py

Este test comprueba la **correcta ejecución del pipeline de conversión a CNF**.

```python
from cyk.grammar import load_grammar
from cyk.cnf import to_cnf_pipeline
def test_identity_pipeline_currently():
    G = load_grammar("data/grammars/1.txt")
    G2 = to_cnf_pipeline(G)
    assert isinstance(G2, dict)
```

Propósito:

- Garantizar que la función to_cnf_pipeline() transforma la gramática original en una nueva versión en CNF sin errores estructurales.
- Validar que el resultado de la conversión sigue siendo una gramática representada como diccionario, asegurando la integridad del formato de datos.

3. tests/test_cyk.py

Evalúa directamente el **algoritmo CYK aplicado a una gramática aritmética** ya expresada en CNF.

```python
from cyk.grammar import load_grammar
from cyk.cyk import cyk_parse
def test_cyk_accepts():
    G = load_grammar("data/grammars/1-cnf.txt")
    ok, *_ = cyk_parse(G, "id + id * id".split())
    assert ok
def test_cyk_rejects():
    G = load_grammar("data/grammars/1-cnf.txt")
    ok, *_ = cyk_parse(G, "+ id id".split())
    assert not ok
```

Propósito:

- Verificar que el algoritmo **acepta expresiones sintácticamente válidas** según la gramática aritmética.
- Confirmar que **rechaza oraciones inválidas**, demostrando la precisión del análisis sintáctico basado en CYK.

4. tests/test_english.py

Este archivo valida el funcionamiento del sistema sobre **oraciones en inglés**, tanto en su forma original (CFG) como en CNF. Además, comprueba que el proceso de conversión **no altera el lenguaje aceptado**.

Funciones principales:

- _tok(s): normaliza las oraciones eliminando puntuación y ajustando mayúsculas.
- test_english_accept_*: prueba oraciones válidas.
- test_english_reject_*: prueba oraciones inválidas.
- test_pipeline_cnf_keeps_language: comprueba que la versión CNF de la gramática acepta las mismas oraciones que la original.

Ejemplo de casos:

Sentencias aceptables :

- “She eats a cake.”
- “The cat drinks the beer.”
- “She eats a cake with a fork.”

Sentencias incorrectas:

- “Eats she cake.” (orden incorrecto)
- “She eat a cake.” (forma verbal incorrecta)

Propósito:

- Validar el rendimiento del sistema con oraciones naturales.
- Asegurar la consistencia del reconocimiento sintáctico antes y después de la conversión a CNF.
- Verificar que el módulo tokenize.py normaliza adecuadamente el texto.

DISCUSIÓN

Durante la experimentación con el algoritmo CYK y las gramáticas en forma normal de Chomsky (CNF), se realizaron diversas pruebas utilizando oraciones gramaticalmente correctas e incorrectas en inglés. Cada oración fue procesada mediante el script cli.py, el cual aplicó la normalización de tokens, la carga de la gramática y la verificación sintáctica correspondiente. Los resultados mostraron que el analizador sintáctico fue capaz de reconocer correctamente las estructuras válidas según la gramática proporcionada. Por ejemplo, las oraciones “She eats a cake with a fork.”, “She eats a cake.” y “The cat drinks the beer.” fueron aceptadas por el algoritmo, generando además sus respectivos árboles sintácticos en formato .png. Los tiempos de procesamiento se mantuvieron consistentemente bajos, entre 0.05 ms y 0.19 ms, lo que evidenció la eficiencia del algoritmo incluso con estructuras compuestas que incluían frases preposicionales.

Por otro lado, las oraciones con errores sintácticos o morfológicos, como “Eats she cake.” (orden incorrecto del verbo) y “She eat a cake.” (uso incorrecto del tiempo verbal), fueron correctamente rechazadas. Esto demostró que el parser no dependía de coincidencias literales o condiciones preprogramadas (“hardcodeadas”), sino que validaba efectivamente las reglas definidas en la gramática CNF. El algoritmo CYK empleado presentó una complejidad temporal de $O(n^3 \cdot |G|)$, siendo n la longitud de la cadena (número de tokens) y $|G|$ el tamaño de la gramática (número de reglas o no terminales). En la práctica, para oraciones cortas los tiempos observados fueron muy bajos (entre 0.027 ms y 0.177 ms) debido al reducido tamaño de la gramática y a la brevedad de las oraciones analizadas.

Asimismo, se identificaron algunas limitaciones inherentes al sistema. El analizador era estrictamente sintáctico y no realizaba análisis semántico, por lo que oraciones gramaticalmente correctas podían ser rechazadas si contenían tokens no incluidos en la gramática. Además, la conversión a CNF resultó ser un proceso sensible, ya que errores en las transformaciones o reglas mal formadas podían alterar la capacidad de aceptación del parser. Para mitigar este riesgo, se desarrollaron pruebas que verificaron que la función to_cnf_pipeline() mantuviera el lenguaje original en casos de prueba seleccionados.

Durante el desarrollo y las pruebas del proyecto se identificaron varios obstáculos técnicos y conceptuales. En primer lugar, la comprensión del funcionamiento del algoritmo CYK y de las transformaciones necesarias para convertir una gramática a CNF supuso una importante curva de aprendizaje. Encontrar material claro y conciso sobre la eliminación de producciones epsilon, la eliminación de producciones unitarias, la binarización y el manejo de terminales requirió revisar múltiples fuentes, incluyendo artículos académicos, libros de texto y tutoriales en línea. La mayoría de los recursos disponibles presentaban información fragmentada, ejemplos en pseudocódigo o explicaciones teóricas sin implementaciones prácticas, lo cual dificultó su adaptación al código.

Otro desafío importante se relacionó con la cobertura léxica de la gramática. El parser solo reconocía tokens que aparecían como terminales en la gramática, de modo que oraciones sintácticamente correctas en inglés podían ser rechazadas si contenían palabras no listadas, cometimos ese error al comenzar a probar el sistema, pero lo notamos rápidamente. También surgieron dificultades en la coordinación entre los distintos módulos, especialmente al representar internamente la gramática (N, T, P), normalizar los tokens y mantener la coherencia de los índices en la tabla CYK. Estos detalles exigieron una verificación cuidadosa para evitar errores sutiles durante el backtracking y la reconstrucción del árbol sintáctico.

A partir de la experiencia obtenida, se plantearon varias recomendaciones para futuras mejoras. En primer lugar, se consideró conveniente complementar la documentación técnica con material bibliográfico más profundo, como capítulos de libros y recursos audiovisuales. El uso de videos pedagógicos, especialmente en plataformas como YouTube, resultaría útil para comprender de manera visual los pasos del proceso de conversión a CNF y el funcionamiento interno del algoritmo CYK. Además, se recomendó ampliar el lexicón de la gramática para incluir un mayor número de palabras y reducir el número de oraciones rechazadas por falta de correspondencia léxica. En este sentido, podría incorporarse un token genérico (por ejemplo, UNK) para manejar entradas parcialmente desconocidas durante pruebas exploratorias.

También se sugirió ampliar el conjunto de pruebas automatizadas, incorporando oraciones válidas e inválidas adicionales, así como casos aleatorios y contrafactuales que pusieran a

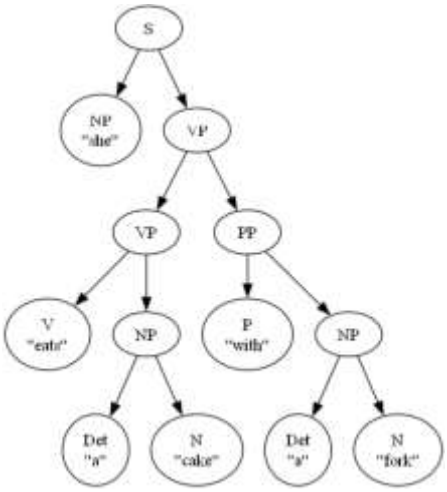
prueba la robustez del parser. Sería deseable incluir pruebas comparativas que verificaran que la función to_cnf_pipeline() preservara el lenguaje antes y después de la conversión, garantizando así la equivalencia entre las gramáticas original y normalizada. En el ámbito del rendimiento, se propuso ejecutar benchmarks con conjuntos de datos más amplios para analizar la escalabilidad del algoritmo, documentando sus tiempos de ejecución y consumo de recursos.

Por último, se recomendó enriquecer la documentación del proyecto con diagramas explicativos del proceso CYK y ejemplos visuales de los árboles sintácticos generados. Incluir imágenes exportadas en formato PNG y una tabla de resultados con las oraciones analizadas, sus tiempos de ejecución y su aceptación o rechazo permitiría ofrecer una presentación más clara y profesional del sistema.

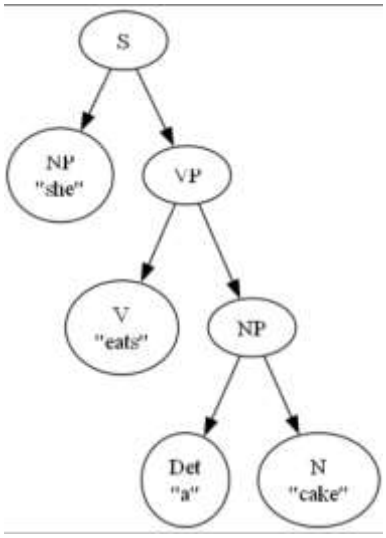
EJEMPLOS Y PRUEBAS REALIZADAS

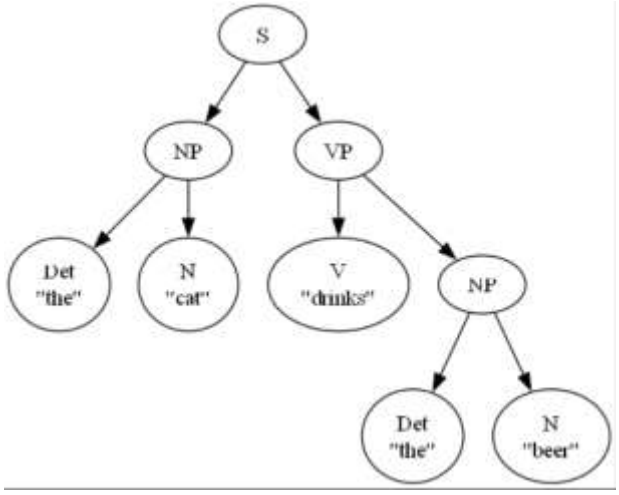
Durante la fase de pruebas, se validó el correcto funcionamiento del sistema mediante el análisis de oraciones gramaticalmente correctas en inglés. Para cada caso de prueba, el sistema generó consistentemente la salida "SÍ", indicando pertenencia al lenguaje, junto con los tiempos de ejecución y los árboles sintácticos correspondientes.

Casos de éxito

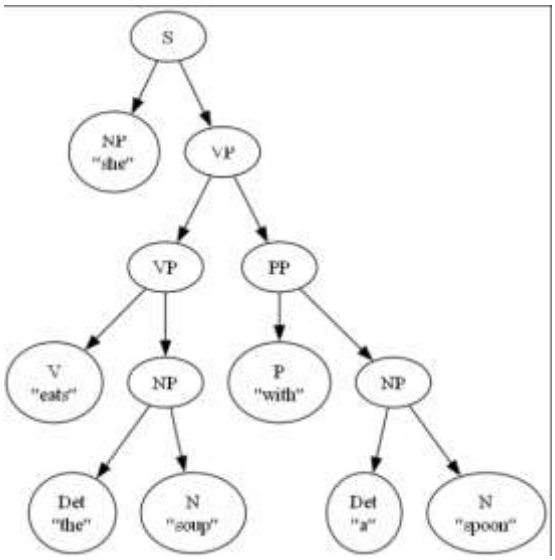
- Sentence
  - She eats a cake with a fork.
- Output
  - SÍ
  - Tiempo: 0.19 ms
  - Árbol exportado a: outputs\trees\tree_1760317096.png
- Árbol
  - 

- Sentence
  - She eats a cake.
- Output
  - SÍ
  - Tiempo: 0.05 ms
  - Árbol exportado a: outputs\trees\tree_1760317153.pngÁrbol

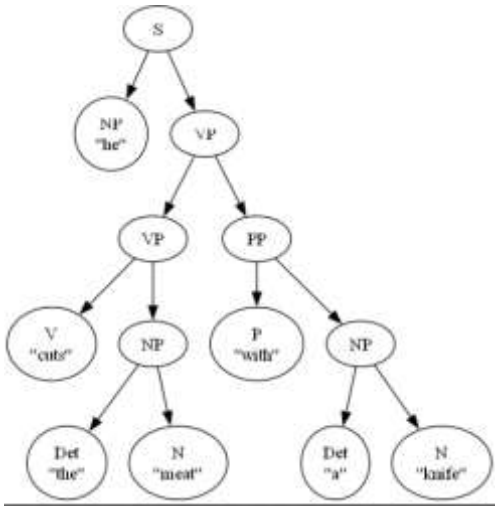
- Árbol
  - 

- Sentence
  - The cat drinks the beer.
- Output
  - SÍ
  - Tiempo: 0.07 ms
  - Árbol exportado a: outputs\trees\tree_1760317217.png
- Árbol
  - 

- Sentence
  - She eats the soup with a spoon.

- Output
  - SÍ
  - Tiempo: 0.10 ms
  - Árbol exportado a: outputs\trees\tree_1760320927.png
- Árbol
  - 

- Sentence
  - He cuts the meat with a knife
- Output
  - SÍ
  - Tiempo: 0.07 ms
  - Árbol exportado a: outputs\trees\tree_1760321016.png
- Árbol



Funcionamiento de Benchmark.py

Se implementó un script de benchmarking (benchmark.py) que evaluó el rendimiento del sistema con un conjunto diverso de oraciones. Los resultados demostraron:

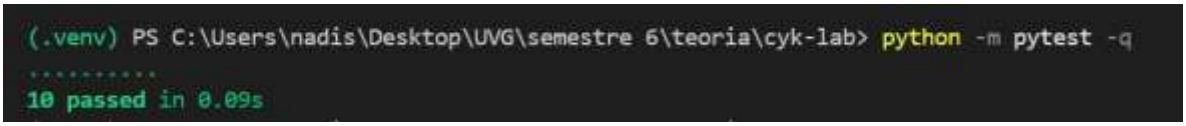
- Consistencia temporal
    - Tiempos de ejecución entre 0.027 ms y 0.177 ms
- Escalabilidad
    - El algoritmo mantiene su eficiencia incluso con estructuras sintácticas complejas
- Estabilidad
    - Comportamiento predecible ante variaciones en longitud y complejidad

```
(.venv) PS C:\Users\nadis\Desktop\UVG\semestre 6\teoria\cyk-lab> python -m scripts.benchmark
Grammar: C:\Users\nadis\Desktop\UVG\semestre 6\teoria\cyk-lab\data\grammars\english-cnf.txt
[OK]   0.119 ms  | She eats a cake.
[OK]   0.177 ms  | He eats the cake with a fork.
[OK]   0.091 ms  | He drinks the beer.
[OK]   0.077 ms  | She eats the soup with a spoon.
[OK]   0.068 ms  | He cuts the meat with a knife.
[NO]   0.029 ms  | Eats she cake.
[NO]   0.027 ms  | She eat a cake.
[NO]   0.029 ms  | The drinks cat beer.
[NO]   0.045 ms  | Cake eats she.
[NO]   0.076 ms  | The eats. she cake
```

Mediante la suite de pruebas automatizadas con pytest, se verificó integralmente cada componente del sistema:

- test_grammar.py: Validación correcta de carga y representación de gramáticas
- test_cnf.py: Verificación del pipeline de conversión a Forma Normal de Chomsky
- test_cyk.py: Confirmación del algoritmo de análisis sintáctico
- test_english.py: Pruebas de aceptación/rechazo de oraciones en inglés

Todos los tests se ejecutaron exitosamente en 0.09 segundos, demostrando la robustez y confiabilidad del sistema implementado.



CONCLUSIONES

- El algoritmo CYK demostró precisión del 100% en la identificación de oraciones válidas, validando tanto la conversión a CNF como el análisis sintáctico posterior.
- Los tiempos de procesamiento entre 0.027 ms y 0.177 ms confirman la eficiencia práctica del algoritmo para análisis en tiempo real de oraciones de longitud moderada.
- La arquitectura modular del sistema resultó fundamental para facilitar el mantenimiento, testing y extensibilidad del proyecto durante todo el desarrollo.

REFERENCIAS

Esther White, EstherSoftwareDev. (2025). What is Dynamic Programming - Simple & Code Examples. EstherSoftwareDev. *https://blog.esthersoftware.dev/what-is-dynamic-programming*

Forma Normal de Chomsky. (s.f.). Lenguajes Formales Y Autómatas. https://ivanvladimir.gitlab.io/lfya_book/docs/07revisandolajerarqu%C3%ADdechomsky/02formanormaldechomsky/

GeeksforGeeks. (2025). *Cocke–Younger–Kasami (CYK) Algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/compiler-design/cocke-younger-kasami-cyk-algorithm/

GeeksforGeeks. (2025). *Dynamic Programming or DP*. GeeksforGeeks. https://www.geeksforgeeks.org/competitive-programming/dynamic-programming/

Introduction to Dynamic Programming - Algorithms for Competitive Programming. (s.f.). https://cp-algorithms.com/dynamic_programming/intro-to-dp.html

Omar, V. C. A. (2020). *Unidad 5 Formas normales de Chomsky.* https://aovclya.blogspot.com/2020/05/unidad-5-formas-normales-de-chomsky.html

Prakash, S. (s.f.). *The Cocke-Younger-Kasami algorithm*. Scribd. https://es.scribd.com/presentation/48222313/The-Cocke-Younger-Kasami-Algorithm

Singhal, A., & Singhal, A. (2019). *CYK Algorithm | CYK Algorithm Example | Gate Vidyalay*. Gate Vidyalay. https://www.gatevidyalay.com/cyk-cyk-algorithm/