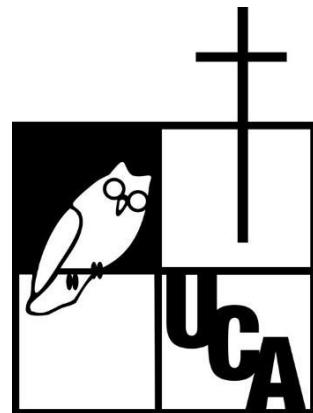


Universidad Centroamericana José Simeón Cañas
Facultad de Ingeniería y Arquitectura



Teoría de Lenguajes de Programación

Gramáticas formales y algoritmos modernos: enlace entre los lenguajes formales y el procesamiento de lenguaje natural

Fase 2 – Mini-parser para lenguaje natural limitado

Catedrático: Jaime Roberto Clímaco Navarrete

Integrantes:

Campos Umanzor Denys Eduardo - 00165221

García Castillo Rodolfo Rafael - 00082421

Leiva Cabrera Miriam Sarai - 00164021

Merlos Rodríguez Angel Gabriel - 00005118

Sibrian Rivera Ricardo José - 00173821

Fecha de entrega: Domingo 30 de noviembre del 2025

1. Resumen ejecutivo.

Este proyecto presenta el diseño e implementación de un Analizador Descendente Recursivo que acepta un subconjunto restringido y bien definido del idioma inglés. El objetivo principal es validar tanto la estructura sintáctica como la concordancia semántica (número gramatical) mediante una Gramática Libre de Contexto (CFG) optimizada para un análisis determinista LL(1).

De forma adicional, se ha diseñado y codificado la implementación de una estrategia de recuperación de errores basada en el Modo Pánico. En lugar de detener la ejecución ante el primer fallo sintáctico, el parser registra el error y sincroniza el flujo de entrada descartando tokens hasta encontrar un delimitador seguro (la coma). Esto permite el procesamiento continuo de múltiples oraciones y la generación de un reporte acumulativo de errores en una sola ejecución, evitando la terminación abrupta del programa.

Finalmente, se contrasta este enfoque con herramientas modernas de Procesamiento de Lenguaje Natural (NLP) como SpaCy. Mientras que el parser desarrollado ofrece precisión estricta dentro de sus reglas predefinidas, su naturaleza determinista limita su alcance. A diferencia de este modelo basado en reglas rígidas, los sistemas de NLP operan como modelos estadísticos y probabilísticos entrenados con grandes volúmenes de datos, lo que les otorga una flexibilidad superior y una cobertura léxica mucho más amplia para manejar la ambigüedad inherente del lenguaje natural.

2. Diseño y Especificación de la Gramática.

2.1. Definición de la Gramática.

El sistema implementa una Gramática Libre de Contexto (CFG) que ha sido factorizada para ser compatible con el análisis LL(1). Específicamente, implementa un subconjunto del idioma inglés, el cual será detallado más adelante.

Propiedades:

- **Sin Recursividad por la Izquierda:** Las reglas recursivas (como listas de adjetivos o frases preposicionales) están diseñadas mediante recursividad por la derecha o iteración, evitando ciclos infinitos en un parser descendente.
- **Factorizada por la Izquierda:** Las decisiones son deterministas basándose en el siguiente token.

2.2. Tokenización.

El analizador utiliza un vocabulario controlado donde cada palabra pertenece a una categoría gramatical que recibe del Lexer.

- DET (determinantes)
- N (sustantivos contables, incontables, colectivos, plurales)
- PRON (pronombres)
- V (verbos en presente simple, versiones SG/PL)
- AUX (modales: can, may, must)
- ADJ (adjetivos)
- PREP (preposiciones)
- COMMA (separador de oraciones)

Cada token incluye palabra original, categoría, número gramatical y posición.

Observación: La lista completa de las palabras terminales aceptadas por el lexer se encuentran en el archivo **Diccionario.pdf**, el cual puede ser encontrado en el repositorio del proyecto.

2.3. Diseño de la Gramática.

Esta gramática define formalmente la estructura de oraciones declarativas simples en el idioma inglés. Desde una perspectiva lingüística, la estructura fundamental que valida esta gramática responde al siguiente esquema:

$$\textit{Oración (S)} = \textit{Frase Nominal NP (Sujeto)} + \textit{Frase Verbal VP (Predicado)}$$

El sistema asegura no solo el orden lineal de los elementos, sino también las relaciones de dependencia gramatical, específicamente la concordancia de número (Singular/Plural) entre el núcleo del sujeto y el verbo principal, así como las restricciones de determinación en los sustantivos.

2.4. Definición Formal de Producciones.

A continuación se presenta el conjunto de reglas de producción que definen matemáticamente el lenguaje aceptado.

1. $S \rightarrow NP\ VP$
2. $NP \rightarrow PRON$
 - | DET $AdjList\ N\ PPList$
 - | N $PPList$
3. $VP \rightarrow AuxList\ V\ VPBody$
4. $VPBody \rightarrow NP\ PPList\ | PPList\ | \epsilon$
5. $PPList \rightarrow (PREP\ NP)^*$
6. $AdjList \rightarrow (ADJ)^*$
7. $AuxList \rightarrow (AUX)^*$

- **La Oración (S):** Representa la unidad gramatical completa. Impone que toda oración debe poseer explícitamente un Sujeto (NP) y un Predicado (VP), y exige concordancia de número entre ambos.
- **La Frase Nominal (NP):** Define los argumentos (sujeto u objeto). Valida tres construcciones:
 1. **Pronominal:** Pronombres con número inherente.
 2. **Determinada:** Estructura completa donde se valida la concordancia interna, es decir, el parser verifica que el número gramatical del determinante coincida con el del sustantivo.
 3. **Sustantivo sin determinante (Bare Noun):** Permitido solo para plurales, incontables o colectivos.
- **La Frase Verbal (VP):** Constituye el predicado. Gestiona la acción y permite la presencia opcional de auxiliares.
- **Cuerpo del Predicado (VPBody):** Gestiona la transitividad. Permite que el verbo sea seguido de un Objeto Directo (NP), solo de complementos (PPList) o de nada (ϵ).
- **Estructuras Recursivas (Listas):** AuxList, AdjList y PPList permiten la expansión indefinida de auxiliares, adjetivos y complementos circunstanciales respectivamente.

Observación: En el repositorio del proyecto se encuentran los archivos ***test/valid.txt*** y ***test/invalid.txt***, en los cuales se pueden ver ejemplos de oraciones sintácticamente válidas e inválidas para entender mejor el alcance del Parser.

3. Implementación del Parser.

3.1. Arquitectura del Parser

El Parser desarrollado se clasifica como un Parser Descendente Recursivo. Esta arquitectura se caracteriza por construir el árbol de derivación sintáctica desde la raíz (símbolo inicial) hacia las hojas (tokens), implementando cada regla gramatical como una subrutina independiente.

3.2. Estructura Global del Programa

Producción Formal: Program → S (COMMA S)*

El parser no está limitado a analizar una única oración. Implementa un bucle de control superior que corresponde a una Clausura de Kleene sobre la estructura (, S).

1. **Procesamiento de Flujo Continuo:** El sistema intenta derivar una oración inicial S. Si tiene éxito (o si se recupera de un error), verifica si existe una coma (,) inmediatamente después.
2. **Iteración Indefinida:** Si encuentra la coma, la consume como un separador sintáctico y repite el proceso llamando nuevamente a la producción S. Este ciclo se mantiene while self.current() is not None.
3. **Doble Propósito de la Coma:** En este nivel, la coma actúa simultáneamente como:
 - **Delimitador de Sentencia:** Separa gramaticalmente una oración de la siguiente.
 - **Punto de Sincronización:** En la estrategia para la recuperación de errores, es el ancla segura para reiniciar el análisis si la oración anterior falló.

3.3. Lógica Detallada de Producciones

A continuación se describe el funcionamiento lógico de cada producción principal y cómo interactúan para validar la estructura.

-> S (Sentence - Oración)

Es el punto de entrada para una oración individual. Su lógica conecta las dos grandes estructuras de la oración:

- **Invocación del Sujeto (NP):** Llama a la subrutina de Frase Nominal. Espera recibir de retorno el atributo de número (Singular/Plural) del sujeto identificado.

- **Validación del Predicado (VP):** Llama a la subrutina de Frase Verbal, pasándole como parámetro el número obtenido del sujeto. Esto permite verificar la concordancia sujeto-verbo inmediatamente.

-> NP (Noun Phrase - Frase Nominal)

Esta es la producción más compleja debido a que maneja tres caminos de derivación distintos basándose en el token de previsualización (*lookahead*):

- **Caso A: Pronombre (PRON):** Si detecta un pronombre, lo consume y sintetiza su número gramatical inherente (ej. "they" → PL).
- **Caso B: Determinante + Sustantivo:**
 1. Consumir el determinante.
 2. Consumir opcionalmente una lista de adjetivos (AdjList).
 3. Consumir el sustantivo núcleo (N).
 4. Verifica que el determinante coincida en número con el sustantivo (ej. Error si encuentra "these boy").
 5. Consumir opcionalmente frases preposicionales (PPList).
- **Caso C: Sustantivo sin determinante:** Ocurre cuando el token es un sustantivo (N) sin determinante previo.
 1. Verifica si el sustantivo tiene permitido aparecer solo. Solo se aceptan sustantivos plurales, incontables o colectivos. Si es singular contable (ej. "car"), lanza un error sintáctico.
 2. Procesa complementos posteriores (PPList).

-> VP (Verb Phrase - Frase Verbal)

Esta producción es responsable de la acción y la concordancia principal.

1. **Procesamiento de Auxiliares:** Consumir una lista opcional de verbos auxiliares (AuxList).
2. **Núcleo Verbal:** Identifica y consume el verbo principal.
3. **Validación de Concordancia (Semántica):** Compara el número del verbo actual con el atributo heredado del Sujeto (NP). Si hay discrepancia (ej. Sujeto="PL", Verbo="SG"), rechaza la entrada, aunque la sintaxis posicional sea correcta.
4. **Cuerpo del Verbo (VPBody):** Decide qué complementos siguen al verbo.

-> VPBody (Cuerpo del Predicado)

Maneja la transitividad del verbo y los complementos circunstanciales. Utiliza el token actual para decidir entre dos caminos:

1. **Transitivo (Objeto Directo):** Si detecta el inicio de una frase nominal (DET, PRON o N), invoca recursivamente a NP para consumir el objeto directo, seguido de posibles frases preposicionales.
2. **Intransitivo / Circunstancial:** Si no detecta un objeto directo, asume que el resto son solo frases preposicionales (PPList) o el fin de la estructura (ϵ).

-> Listas (AdjList, PPList, AuxList)

Estas producciones manejan secuencias repetitivas.

El parser se mantiene en un bucle consumiendo elementos (Adjetivos, Preposiciones o Auxiliares) mientras el token actual coincide con la categoría esperada.

4. Estrategia de Recuperación de Errores.

El parser no se detiene abruptamente ante el primer error, sino que implementa una estrategia de recuperación para poder reportar múltiples fallos en un solo análisis.

- **Técnica:** Recuperación en Modo Pánico.
- **Token de Sincronización:** La Coma (,).
- **Funcionamiento:**
 1. Al detectar un error (sintáctico o de concordancia), se registra el mensaje de error en una lista acumulativa.
 2. El parser entra en estado de "pánico" y comienza a descartar tokens secuencialmente.
 3. El descarte se detiene únicamente al encontrar el token de sincronización (,) o el final del archivo.
 4. Una vez sincronizado, el parser se reinicia en un estado limpio para intentar analizar la siguiente oración de la lista.

5. Pruebas y Resultados del Parser.

Para validar el funcionamiento del parser se diseñó un pequeño banco de pruebas compuesto por oraciones gramaticalmente correctas e incorrectas en inglés, cubriendo tanto la estructura sintáctica como las restricciones de concordancia y uso de sustantivos sin determinantes.

El archivo `run_en.py` permite ejecutar el parser tanto sobre una oración puntual como sobre un archivo de texto, utilizando la siguiente línea de comandos:

```
# Modo oración única
python -m src.run_en "the young students play in the room"

# Modo archivo (una oración por línea)
python -m src.run_en -f test/valid.txt
python -m src.run_en -f test/invalid.txt
```

5.1. Caso de Prueba 1: Ejemplos válidos.

El archivo **test/valid.txt** contiene oraciones gramaticalmente correctas. Algunos ejemplos representativos son:

- boys run
- the boy eats
- the young students play in the room
- we drink water

Para este caso de Prueba, se ejecutará la frase “*the young students play in the room*”:

```
PS C:\Users\garci\Downloads\Mini_parser__en> py -m src.run_en "the young students play in the room"
✓ Parsing successful.
```

Cubriendo:

- Sujeto simple y compuesto por determinante + sustantivo.
- Sujetos en singular.
- Verbos transitivos e intransitivos.
- Presencia de adjetivos (young) y frases preposicionales (in the room, in the school).

5.2. Caso de Prueba 2: Ejemplos inválidos y manejo de errores.

El archivo **test/invalid.txt** contiene oraciones diseñadas para producir distintos tipos de errores. Algunos ejemplos por categoría son:

a) Sustantivo contable singular sin determinante (bare noun prohibido)

- book eats
- the cat plays ball

En ambos casos, el núcleo del sujeto (book, ball) está etiquetado en el léxico como nombre **singular contable (SGC)**. El parser permite sustantivos sin determinantes solo si son plurales, incontables o colectivos, por lo que se genera un error del tipo:

```
PS C:\Users\garci\Downloads\Mini_parser__en> py -m src.run_en "book eats"
✗ Errors found:
  1. Grammar Error: Countable singular noun 'book' cannot appear without a determiner.
```

b) Concordancia sujeto–verbo incorrecta

- the boys eats
- we drinks water

En estos ejemplos se mantiene la estructura sintáctica básica (NP VP), pero se fuerza una discrepancia entre el número del sujeto y el del verbo principal. El parser detecta esta incoherencia en la producción VP, donde compara el número heredado del sujeto con el número del verbo, produciendo mensajes del tipo:

```
PS C:\Users\garci\Downloads\Mini_parser_en> py -m src.run_en "the boys eats"
X Errors found:
 1. Subject-Verb Agreement Error: Subject is PL, but verb 'eats' is SG.
```

c) Discordancia determinante–sustantivo en el sujeto

- this students work
- this dogs run

Aquí el error se produce **dentro de la frase nominal (NP)**, antes incluso de llegar al verbo:

- **this** está etiquetado como determinante singular (SG).
- **students** y **dogs** están etiquetados como plurales (PL).

Al verificar la producción NP, el parser detecta la inconsistencia y lanza errores del tipo:

```
PS C:\Users\garci\Downloads\Mini_parser_en> py -m src.run_en "this students work"
X Errors found:
 1. Agreement Error: Determiner 'this' (SG) mismatch with noun 'students' (PL).
```

d) Estructuras sintácticamente incompletas o no modeladas

- boys can
- the boy in the city

En estos casos la oración se “corta” antes de completar una estructura esperada:

- **boys can** no presenta verbo principal tras el auxiliar **can**.
- **the boy in the city** se queda sin verbo (solo NP).

En todos estos escenarios, la función **accept** detecta que el token actual no coincide con la categoría esperada (o que se llegó al fin de la entrada) y lanza un **ParseError** señalando la categoría esperada y el token encontrado.

```
PS C:\Users\garci\Downloads\Mini_parser_en> py -m src.run_en "boys can"
X Errors found:
 1. Unexpected end of input. Expected category: V.
```

5.3. Caso de Prueba 3: Ejemplo de recuperación en modo pánico.

Para ilustrar el efecto práctico del modo pánico, se probó el siguiente comando:

```
python -m src.run_en "the dogs eat apples, this dogs run, that  
student sees the teachers"
```

Esta entrada contiene **tres oraciones** separadas por comas:

1. the dogs eat apples → gramaticalmente correcta.
2. this dogs run → incorrecta (DET singular + N plural).
3. that student sees the teachers → correcta.

El parser procesa la primera oración sin problemas. Al analizar la segunda, detecta el error de concordancia determinante–sustantivo:

```
PS C:\Users\garci\Downloads\Mini_parser_en> py -m src.run_en "the dogs eat apples, this dogs run, that student sees  
the teachers"  
X Errors found:  
1. Agreement Error: Determiner 'this' (SG) mismatch with noun 'dogs' (PL).
```

Si tenemos más de una oración incorrecta (en la primera oración quitamos la “s” a [dogs](#)):

```
PS C:\Users\garci\Downloads\Mini_parser_en> py -m src.run_en "the dog eat apples, this dogs run, that student sees  
the teachers"  
X Errors found:  
1. Subject-Verb Agreement Error: Subject is SG, but verb 'eat' is PL.  
2. Agreement Error: Determiner 'this' (SG) mismatch with noun 'dogs' (PL).
```

En ese momento entra en **modo pánico**:

1. Registra el mensaje de error en una lista interna.
2. Comienza a descartar tokens uno por uno hasta encontrar el token de sincronización, que en este diseño es la **coma** (,).
3. Una vez encontrada la coma, la consume y reinicia el análisis como si comenzara una nueva oración (S) desde el token siguiente.

Gracias a esta estrategia, la tercera oración ([that student sees the teachers](#)) se analiza completamente y se valida como correcta, a pesar de que hubo un error en la oración anterior.

En un parser descendente recursivo sin recuperación en modo pánico, el comportamiento típico sería detener la ejecución al primer error y abortar el análisis del resto de la entrada. En contraste, el enfoque implementado permite:

- Reportar varios errores en una sola pasada sobre el archivo de entrada.
- Aislard oraciones defectuosas sin perder la información de las oraciones siguientes.

6. Comparación con NLP moderno.

6.1. Selección de la herramienta.

Para este análisis comparativo se seleccionó la biblioteca spaCy (específicamente el modelo en_core_news_sm para inglés).

- **Justificación:** A diferencia de nuestro mini-parser académico, spaCy es una herramienta de grado industrial que utiliza modelos estadísticos pre-entrenados sobre grandes textos, por lo que tiene la capacidad de realizar Dependency Parsing (Análisis de Dependencias) y POS Tagging (Etiquetado Gramatical) simultáneamente.

6.2. Comparación Parser vs NLP.

Esta es la diferencia fundamental entre el proyecto realizado y la herramienta moderna:

Parser: Enfoque determinista.

Nuestro mini-parser funciona bajo un paradigma de Autómatas y Gramáticas Libres de Contexto, en donde se valida la entrada basándose en un conjunto finito de reglas de producción predefinidas.

En este sentido, la oración es "Válida" (pertenece al lenguaje) o "Inválida" (error de sintaxis). No hay puntos medios. Si una palabra no está en el vocabulario o la estructura varía mínimamente de la regla, el parser rechaza la entrada.

NLP: Enfoque Moderno estadístico y probabilístico.

Las herramientas modernas de NLP (como los modelos basados en Redes Neuronales o Transformadores) no "validan" reglas estrictas, sino que predicen la estructura más probable.

En lugar de buscar una coincidencia exacta en una tabla de parsing (como la tabla LL(1)), el modelo calcula la probabilidad $P(\text{Etiqueta} \mid \text{Contexto})$. Utiliza vectores (embeddings) para entender que "gato" y "felino" funcionan gramaticalmente igual, aunque no tenga una regla explícita para ambos.

Si la oración es ambigua (ej. "*Vi al hombre con el telescopio*"), el modelo estadístico elegirá el árbol de análisis que tenga la mayor probabilidad acumulada basada en los millones de oraciones con las que fue entrenado, en lugar de detenerse por un conflicto de reglas.

6.3. Caso de Prueba 1: Ejemplo simple aceptado por el Parser.

Oración de entrada: we drink water

Análisis del Parser:

El parser procesó esta oración exitosamente validando reglas semánticas específicas:

- Sujeto (NP):** Identificó "we" como PRON con atributo Number=PL.
- Verbo (VP):** Identificó "drink" como V (Number=PL).
- Validación:** Confirmó la concordancia Sujeto-Verbo (PL ↔ PL).
- Restricción Bare Noun:** Aceptó "water" sin determinante porque el léxico lo clasifica como Incontable (UNC), cumpliendo la regla de noun Allows_Bare.

Análisis de SpaCy:

TOKEN	POS	DEP	LEMMA	MORPH
we	PRON	nsubj	we	Case=Nom Number=Plur Person=1 PronType=Prs
drink	VERB	ROOT	drink	Tense=Pres VerbForm=Fin
water	NOUN	dobj	water	Number=Sing

SpaCy generó el siguiente árbol de dependencias:

- nsubj (Sujeto nominal):** we (Morph: Number=Plur).
- ROOT (Raíz):** drink.
- dobj (Objeto directo):** water (Morph: Number=Sing).

Conclusión del Caso:

Ambos sistemas interpretaron la estructura idénticamente.

- Diferencia Clave:** El Parser Propio aceptó la oración porque cumple la regla. SpaCy analizó la oración porque es estadísticamente probable en el idioma inglés.
- Observación:** SpaCy detecta que "water" es singular (Number=Sing), pero no impone una restricción de error si falta el determinante; simplemente lo etiqueta. Nuestro parser, en cambio, *hubiera fallado* si "water" fuera un sustantivo contable singular (ej. "we drink apple"), demostrando una validación normativa más estricta en este contexto cerrado.

6.4. Caso de Prueba 2: Ejemplo complejo aceptado por el Parser.

Oración de entrada: boys give the book to the girls

Análisis del Parser:

El sistema demostró su capacidad para manejar recursividad y complementos:

- Sujeto:** "boys" aceptado como *Bare Noun* por ser Plural (PL).
- Verbo:** "give" (PL) concuerda con "boys".
- Objeto Directo:** "the book" procesado como NP anidado.
- Complemento (PPList):** "to the girls" procesado exitosamente como una Frase Preposicional (PREP + NP) adjunta al VP.

Análisis de SpaCy:

TOKEN	POS	DEP	LEMMA	MORPH
boys	NOUN	nsubj	boy	Number=Plur
give	VERB	ROOT	give	Tense=Pres VerbForm=Fin
the	DET	det	the	Definite=Def PronType=Art
book	NOUN	dobj	book	Number=Sing
to	ADP	dative	to	
the	DET	det	the	Definite=Def PronType=Art
girls	NOUN	pobj	girl	Number=Plur

El modelo generó un árbol más amplio en matices semánticas:

- **nsubj:** boys.
- **dobj:** book.
- **dative (Dativo):** SpaCy identifica “to” no sólo como una preposición genérica, sino como un marcador de dativo(objeto indirecto).
- **pobj (Objeto de preposición):** girls.

Conclusión del Caso

El Parser propio resolvió la estructura sintáctica linealmente ("esto es una preposición seguida de un nombre"). SpaCy fue un paso más allá identificando la función grammatical del complemento (Dativo/Receptor). Sin embargo, el resultado de nuestro parser es perfecto dentro de su alcance: validó que la estructura grammatical es correcta y que todas las concordancias de número (boys-give, the-book, the-girls) son correctas.

6.5. Caso de Prueba 3: Ejemplo rechazado por el Parser.

Oración de entrada: boys can

Este escenario evalúa el comportamiento de ambos sistemas ante una oración grammaticalmente incompleta. En la gramática normativa del inglés, los auxiliares modales (*can, must, may*) muchas veces requieren un verbo léxico principal para formar un predicado válido.

Análisis del Parser:

Resultado: FALLO (Parse Error).

1. El parser consume exitosamente el sujeto boys (NP).
2. Ingresa a la producción VP.

3. La subrutina AuxList consume el token can.
4. El parser intenta ejecutar la siguiente instrucción obligatoria: self.accept("V") (esperando el verbo principal).
5. Al no encontrar más tokens, se lanza una excepción crítica:
"Unexpected end of input. Expected category: V."

El sistema actúa como un validador estricto. Si la estructura no está completa según la definición formal $VP \rightarrow AuxList \ V \ VPBody$, la entrada es rechazada.

Análisis de SpaCy:

TOKEN	POS	DEP	LEMMA	MORPH
boys	NOUN	nsubj	boy	Number=Plur
can	AUX	ROOT	can	VerbForm=Fin

Resultado: El modelo de NLP no juzga la validez gramatical estricta, sino que intenta construir el mejor árbol de dependencias posible con los elementos disponibles.

Conclusión del Caso

Mientras el Parser dice invalida la oración porque la estructura no es correcta, SpaCy logra tokenizar correctamente las palabras.

6.6. Caso de Prueba 4: Ejemplo fuera del alcance del Parser.

Oración de entrada: How many times?

Este caso representa un problema para nuestro parser: la entrada contiene vocabulario no definido en nuestro diccionario y utiliza una estructura interrogativa que no existe en nuestra gramática ($S \rightarrow NP \ VP$).

Análisis del Parser:

Resultado: FALLO (Lexical Error).

El proceso falla en la etapa de pre-procesamiento (Tokenización), antes de iniciar el análisis sintáctico, debido a que la palabra "How" no existe en el LEXICON estático del proyecto.

Error: LexicalError: Unknown word 'how' at position 1.

Incluso si agregáramos las palabras al diccionario, el parser fallaría sintácticamente después, ya que la gramática espera un Sujeto (NP) al inicio, y no tiene reglas para manejar adverbios interrogativos (How) o signos de puntuación final (?). Este ejemplo demuestra las limitantes de nuestro parser.

Análisis de SpaCy:

TOKEN	POS	DEP	LEMMA	MORPH
How	SCONJ	advmod	how	
many	ADJ	amod	many	Degree=Pos
times	NOUN	ROOT	time	Number=Plur
?	PUNCT	punct	?	PunctType=Peri

Resultado: SpaCy demuestra su capacidad de POS Tagging. Gracias a su entrenamiento previo con millones de textos, puede inferir las categorías gramaticales y las relaciones de dependencia sin necesidad de reglas manuales explícitas.

Interpretación:

- Tagging preciso:** Identifica many como adjetivo (ADJ) y times como sustantivo plural (NOUN), a pesar de la ambigüedad de estas palabras en inglés.
- Manejo de fragmentos:** Al no haber verbo, el modelo identifica correctamente al sustantivo times como el núcleo semántico (ROOT) de la frase.
- Dependencias:** Establece que How modifica a many (advmod), y many modifica a times (amod).

Conclusión del Caso

Nuestro Parser falló porque la palabra “How” no está contemplada por el lexer, pero incluso si la agregamos, nuestra gramática tiene una estructura bien definida que no acepta estructuras más complejas como una pregunta. Sin embargo, SpaCy desglosó la estructura interna de la frase interrogativa con precisión lingüística, demostrando que puede entender gramática incluso cuando falta la estructura global (Sujeto+Verbo).

7. Conclusiones.

- El proyecto evidenció que el Parser que se desarrolló es capaz de validar la sintaxis y la concordancia semántica de un subconjunto definido de un lenguaje natural con una precisión del 100%. Esto confirma que, si bien un parser determinista no puede abarcar la flexibilidad del lenguaje humano completo, es la herramienta ideal para validar lenguajes acotados donde se requiere un cumplimiento normativo estricto y sin ambigüedades.
- La implementación de la estrategia de recuperación de errores mediante el Modo Pánico resultó ser un componente fundamental de la solución, ya que si bien las pruebas funcionales se realizaron con entradas breves (tres oraciones, separadas por coma), la importancia de esta técnica escala drásticamente en escenarios de producción. En un parser diseñado para procesar archivos con miles de líneas de código o texto, la capacidad de no abortar la ejecución ante el primer error es

fundamental. El uso de un token de sincronización permitió aislar los fallos y continuar el análisis, demostrando cómo esta estrategia maximiza la utilidad del diagnóstico al reportar múltiples errores en una sola pasada, en lugar de detenerse abruptamente.

3. La comparación técnica con una herramienta NLP como SpaCy resaltó la diferencia fundamental entre un sistema determinista y uno estadístico. Mientras que el parser desarrollado garantiza que cualquier oración aceptada es gramaticalmente perfecta según las reglas definidas, es incapaz de manejar vocabulario o estructuras desconocidas. Por el contrario, los modelos de NLP basados en estadística y probabilidad sacrifican la validación estricta a favor de la robustez, logrando inferir estructuras incluso en oraciones más complejas. Esto concluye que la elección entre un parser recursivo y un modelo de NLP depende estrictamente del objetivo del sistema.

8. Anexos.

Se adjunta link de repositorio: https://github.com/rjsibrian/TLP_ProyectoFinal.git

El proyecto se organiza en dos módulos principales ubicados al mismo nivel:

(1) mini_parser_en y (2) nlp_implementation.

1. Carpeta mini_parser_en

Su contenido se clasifica de la siguiente manera:

- **Diccionario.pdf**
Documento que recopila todas las palabras terminales reconocidas por el analizador léxico.
- **Carpeta test**
Reúne los archivos utilizados para validar el desempeño del parser.
 - **valid.txt**: contiene oraciones gramaticalmente correctas que deben ser aceptadas por el analizador.
 - **invalid.txt**: contiene ejemplos diseñados para provocar errores sintácticos o semánticos.
- **Carpeta src**
Incluye el código fuente del analizador, dividido en tres componentes esenciales:
 - **en_lexicon.py**: implementa el analizador léxico responsable de segmentar la oración en tokens y asignarles la categoría correspondiente según el diccionario definido.

- **en_parser.py**: contiene la lógica central del Analizador Descendente Recursivo, la implementación de la Gramática Libre de Contexto, las verificaciones de concordancia y la estrategia de recuperación de errores.
- **run_en.py**: script de ejecución que integra el lexer y el parser para permitir el análisis de oraciones individuales o de un archivo completo mediante la línea de comandos.

2. Carpeta nlp_implementation

- **demo_spacy.py**
Script que emplea la biblioteca spaCy para realizar etiquetado morfosintáctico (POS tagging).
- **frases_input.txt**
Archivo que contiene las oraciones utilizadas como entrada para la demostración con spaCy.