Universidad Centroamericana José Simeón Cañas

Teoria de lenguajes de programación

Catedrático: Jaime Climaco

Integrantes

Andres Felipe Cardona Duarte	00037820
Axel Jared Hernández Servellón	00145319
Moises Ezequiel Juárez Mejía	00038221
Josue Alfredo Mejia Urias	00000921

Diferencias Clave entre Lenguajes Formales y Naturales

1. Introducción

El estudio de los lenguajes se divide en dos dominios fundamentalmente distintos: los lenguajes naturales y los lenguajes formales. Los lenguajes naturales, como el español o el inglés, son sistemas de comunicación que han evolucionado orgánicamente en las comunidades humanas. Su propósito principal es la expresión y comunicación flexible de ideas (Jurafsky & Martin, 2023).

Por otro lado, los lenguajes formales, como C++, Java o la lógica matemática, son sistemas artificiales diseñados con propósitos específicos. Se caracterizan por su precisión y la eliminación de la ambigüedad, siendo la base para la computación y la programación. Su estructura está definida matemáticamente, permitiendo que una máquina (como un compilador) los procese de manera determinista (Aho et al., 2007). Este informe explora las diferencias fundamentales entre ambos, centrándose en la estructura, la ambigüedad, la recursividad y el contexto.

2. Comparativa de Características Clave

2.1. Estructura (Sintaxis)

La diferencia más notable radica en la rigidez de la sintaxis.

- **Lenguajes Formales:** Poseen una estructura sintáctica **rígida y prescriptiva**. Están definidos por un conjunto de reglas inflexibles conocidas como **gramática formal**. Como se menciona en el objetivo del proyecto, la mayoría de los lenguajes de programación se modelan exitosamente con **Gramáticas Libres de Contexto (CFG)**. En este marco, la validez de una sentencia es binaria: o cumple perfectamente con la gramática, o es rechazada. Por ejemplo, olvidar un punto y coma en C++ no es un error menor, sino una violación sintáctica que invalida el programa (Aho et al., 2007).
- **Lenguajes Naturales:** Tienen una estructura **flexible y descriptiva**. Aunque existen reglas gramaticales (sujeto, verbo, objeto), estas están repletas de excepciones, variaciones dialectales y flexibilidad de orden para denotar énfasis (ej. "El niño pateó la pelota" vs. "La pelota fue pateada por el niño"). Como indica el proyecto, los lenguajes naturales **exceden el marco de las CFG** porque su estructura no puede ser capturada completamente por este tipo de gramáticas (Jurafsky & Martin, 2023).

2.2. Ambigüedad

La gestión de la ambigüedad es, quizás, la división más clara entre ambos tipos de lenguaje.

- **Lenguajes Formales:** La ambigüedad es considerada un error de diseño y debe ser **eliminada por completo**. Un compilador debe interpretar una línea de código de una única manera. Si una gramática permite que una misma sentencia (ej. if-then-else) tenga dos o más árboles de análisis sintáctico (*parse trees*), se considera una gramática ambigua y debe ser reescrita. Los parsers tradicionales, como el LL(1) que se menciona en el proyecto, son deterministas y no pueden operar sobre gramáticas ambiguas (Aho et al., 2007).
- **Lenguajes Naturales:** La ambigüedad es **inherente y omnipresente**. Es una característica, no un error, que los humanos resuelven sin esfuerzo utilizando el contexto. La ambigüedad puede ser léxica (una palabra con varios significados, ej. "banco"), sintáctica (una frase con varias estructuras, ej. "Vi al hombre con el telescopio") o semántica. Los parsers tradicionales fallan estrepitosamente porque no tienen mecanismos para desambiguar (Jurafsky & Martin, 2023).

2.3. Recursividad

Ambos lenguajes utilizan la recursividad, pero su naturaleza y límites son diferentes.

- **Lenguajes Formales:** La recursividad es una herramienta **matemáticamente definida** en la gramática. Permite la creación de estructuras anidadas de profundidad teóricamente infinita. Por ejemplo, una regla de CFG para expresiones aritméticas puede ser Expr -> Expr + Term. Un parser descendente recursivo se basa directamente en la implementación de estas reglas recursivas (Aho et al., 2007).
- **Lenguajes Naturales:** La recursividad es una propiedad fundamental (postulada por Chomsky) que permite generar un número infinito de oraciones mediante la incrustación de cláusulas (ej. "Esta es la casa [que Juan construyó]", "Este es el queso [que estaba en la casa [que Juan construyó]]..."). Sin embargo, a diferencia de la recursividad formal, la recursividad en el lenguaje natural tiene límites cognitivos prácticos; una oración con diez niveles de anidamiento es gramaticalmente posible, pero funcionalmente incomprensible (Jurafsky & Martin, 2023).

2.4. Contexto

El manejo del contexto es el punto de ruptura final entre los compiladores clásicos y el NLP moderno.

- **Lenguajes formales:** En su mayoría se modelan como lenguajes libres de contexto (CFG). Esto significa que la regla gramatical (por ejemplo, A -> xY) se aplica independientemente de los caracteres que rodean a A. Aunque los compiladores modernos realizan "análisis semántico" (por ejemplo, comprobar si una variable ha sido declarada o si los tipos coinciden), este "contexto" es muy limitado y se trata por separado de la sintaxis, a menudo utilizando tablas de símbolos (Aho et al., 2007).
- Lenguaje natural: muy dependiente del contexto. Sin él, es imposible comprender el significado de una palabra o de una frase. Esto incluye el contexto lingüístico (lo que se ha dicho antes) y el contexto pragmático (el conocimiento sobre el mundo). Por ejemplo, la frase "Es demasiado tarde" puede tomar la forma de un comentario, una orden de irse o una queja, y está determinada únicamente por el contexto. Los limpiadores tradicionales no tienen acceso a este conocimiento y aquí es donde los modelos modernos de PNL (basados en estadísticas y aprendizaje profundo) centran su fuerza (Jurafsky y Martin, 2023).

3.2. Gramática Implementada

El parser fue diseñado para analizar una pequeña gramática formal, un subconjunto simplificado de un lenguaje imperativo tipo C. La gramática es capaz de validar las siguientes estructuras:

- **Declaración de variables:** Reconoce declaraciones de tipos `int`, `float` y **`string`**.
- **Formatos de declaración:** Permite declarar múltiples variables en una línea (ej. `int a, b;`) o inicializar una variable directamente en su declaración (ej. `float x = 10 * y;`).
- **Asignación:** Maneja la instrucción de asignación (`id = E ;`).
- **Expresiones aritméticas:** Incluye soporte para expresiones aritméticas con los operadores +, * y paréntesis `( )`.
- **Estructuras de control:** Reconoce sentencias condicionales **`if`** y bucles **`for`**. (El cuerpo de estas estructuras está limitado a ser una única instrucción de asignación).

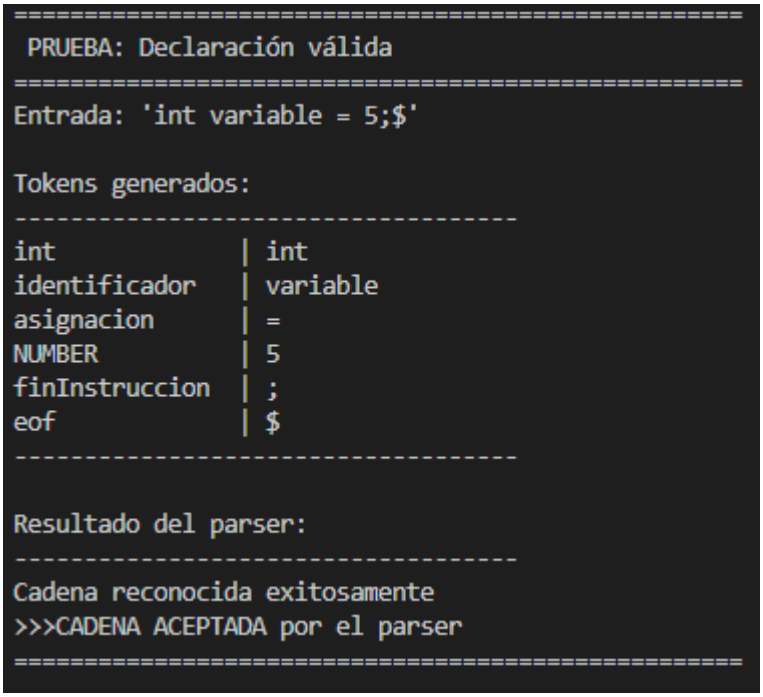
4. Resultados y análisis comparativos

En esta sección se presentan los resultados obtenidos al someter tanto al parser formal desarrollado como a la librería de NLP (spaCy) a diferentes pruebas de entrada. El objetivo es evidenciar experimentalmente las limitaciones de las gramáticas libres de contexto frente al lenguaje natural.

4.1 Pruebas

Prueba 1

- **Entrada:** int variable = 5;$
- **Resultado del parser:**



- **Análisis:** El parser procesó la entrada sin errores. Esto confirma que la implementación del autómata de pila y la tabla LL(1) funcionan correctamente para la sintaxis rígida y predecible del lenguaje formal.

Prueba 2

- **Entrada:** el perro come; $
- **Resultado:**

```
================================================
 PRUEBA: Lenguaje natural (análisis del compilador)
================================================
Entrada: 'el perro come;$'

Tokens generados:
----------------------------------
identificador   | el
identificador   | perro
identificador   | come
finInstruccion  | ;
eof             | $
----------------------------------

Resultado del parser:
----------------------------------
Error: Se esperaba 'asignacion' pero se encontro 'identificador' ('perro')
>>> ERROR: La cadena NO es válida según la gramática
================================================
```

- **Análisis:** El parser falló inmediatamente. El error documentado se debe a dos factores críticos:
    - **Léxico:** El tokenizador clasificó la palabra "el" como un identificador genérico, al no estar en su lista de palabras clave reservadas.
    - **Sintaxis:** La gramática LL(1) no posee una regla de producción que permita comenzar una sentencia con un identificador en este contexto (esperaba un tipo de dato). Esto demuestra que el parser carece de la flexibilidad para interpretar estructuras que no sigan estrictamente las reglas predefinidas, confirmando su incapacidad para procesar lenguaje natural.

Prueba 3: Con librería LNP (spaCy)

- **Entrada:** El perro come
- **Resultado:**

```
============================================================
 PRUEBA: Lenguaje natural (análisis spaCy)
============================================================
Entrada: 'el perro come;$'

=== Análisis spaCy ===
Frase analizada: 'el perro come'

el          | Lema: el          | POS: DET    | Dep: det        | Head: perro
perro       | Lema: perro       | POS: PROPN  | Dep: nsubj      | Head: come
come        | Lema: come        | POS: PROPN  | Dep: ROOT       | Head: come

Interpretación:
- spaCy interpreta la oración como válida en español.
- 'el' → determinante
- 'perro' → sustantivo
- 'come' → verbo (raíz de la oración)
=== Fin de análisis spaCy ===

>>> spaCy analizó correctamente la oración.
============================================================
```

- **Análisis:** A diferencia del parser formal, la herramienta de NLP procesó la entrada exitosamente. No solo aceptó la cadena, sino que realizó un etiquetado gramatical (POS Tagging) correcto, identificando el determinante, el sustantivo y el verbo. Esto evidencia que los modelos modernos, basados en estadística y aprendizaje automático, pueden manejar la ambigüedad y el vocabulario extenso del lenguaje natural, superando las limitaciones de las CFG.

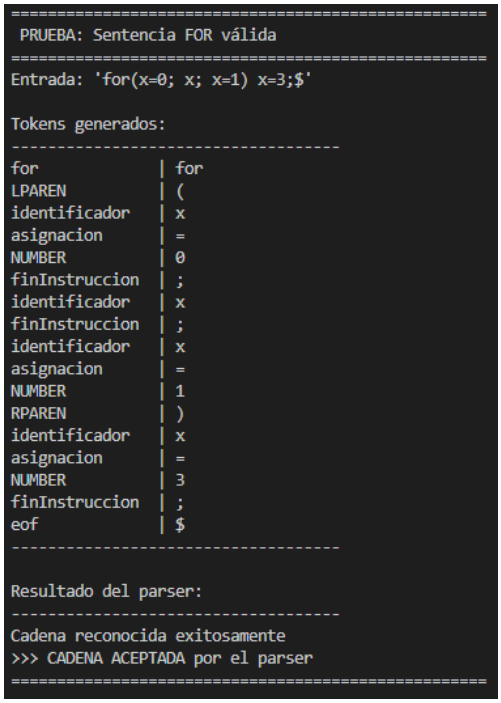
Prueba 4: Prueba de la sentencia IF valido e IF invalido

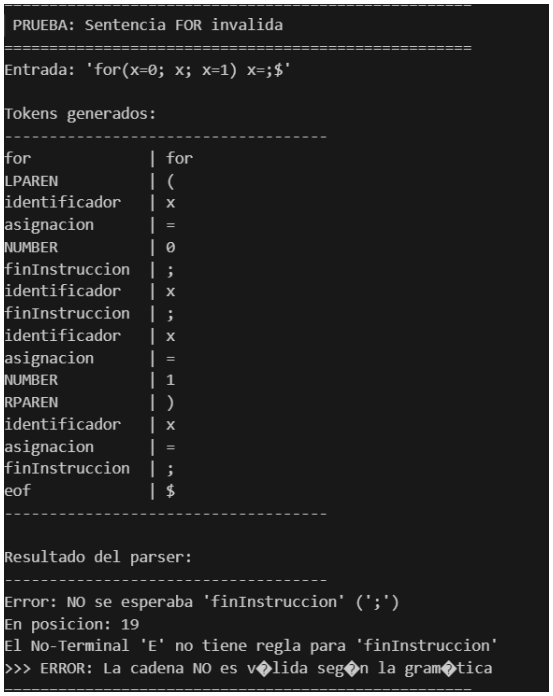
```
================================================
 PRUEBA: Sentencia IF válida
================================================
Entrada: 'if(5+3) x=1;$'

Tokens generados:
-----------------------------------
if              | if
LPAREN          | (
NUMBER          | 5
PLUS            | +
NUMBER          | 3
RPAREN          | )
identificador   | x
asignacion      | =
NUMBER          | 1
finInstruccion  | ;
eof             | $
-----------------------------------

Resultado del parser:
-----------------------------------
Cadena reconocida exitosamente
>>> CADENA ACEPTADA por el parser
================================================
```

```
================================================
 PRUEBA: Sentencia IF invalida
================================================
Entrada: 'if(5+3) x=;$'

Tokens generados:
-----------------------------------
if              | if
LPAREN          | (
NUMBER          | 5
PLUS            | +
NUMBER          | 3
RPAREN          | )
identificador   | x
asignacion      | =
finInstruccion  | ;
eof             | $
-----------------------------------

Resultado del parser:
-----------------------------------
Error: NO se esperaba 'finInstruccion' (';')
En posicion: 10
El No-Terminal 'E' no tiene regla para 'finInstruccion'
>>> ERROR: La cadena NO es v□lida seg□n la gram□tica
```

- **Análisis**. En el primer caso (if(5+3) x=1;$), la entrada se proceso de forma exitosa; el analizador léxico identifico los tokens y el analizador sintáctico valido la secuencia contra la gramática if (E) INST, aceptando la cadena porque (5+3) es una E valida y x=1; es una INST valida. En contraste, en la segunda prueba (if(5+3) x=;$), el parser fallo correctamente tal como se esperaba. El analizador sintáctico, después de procesar x =, esperaba un No-Terminal E (es decir, un NUMBER o identificador) segun la gramática. Sin embargo, recibio inesperadamente un token finInstruccion (';'). Esto provoco el error reportado: El No-Terminal 'E' no tiene regla para 'finInstruccion', demostrando que el parser es estricto y determinista, rechazando cualquier entrada que no cumpla perfectamente con la sintaxis rigida del lenguaje formal.

Prueba 5: Prueba de la sentencia FOR valido y FOR invalido





- **Análisis**: En el caso valido (for(x=0; x; x=1) x=3;$), la cadena es aceptada exitosamente. El parser valida que la estructura completa se alinea con la gramática definida para un bucle for: una inicialización (x=0;), una condición (x), una instrucción de incremento (x=1) seguidas de un cuerpo de bucle valido (x=3;). Por el contrario, en el caso invalido (for(x=0; x; x=1) x=;$), el parser falla. El error ocurre al analizar el cuerpo del bucle. después de leer x =, la gramática espera una expresión (un NUMBER o identificador que conforma un 'E'). Al encontrar un fin Instrucción (';') en su lugar, el parser reporta correctamente que el No-Terminal 'E' no tiene una regla para 'finInstruccion', demostrando su incapacidad para procesar sintaxis malformada.

5. Conclusión

Los lenguajes formales logran precisión sacrificando la flexibilidad, mientras que los lenguajes naturales logran una expresividad ilimitada al costo de la ambigüedad. El análisis sintáctico (*parsing*) de un lenguaje formal es un problema resuelto y determinista. En contraste, el procesamiento del lenguaje natural requiere una gestión compleja de la ambigüedad y el contexto, un desafío que excede las gramáticas libres de contexto y que define el campo del NLP moderno.

6. Referencias

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, techniques, and tools* (2nd ed.).

Pearson Addison-Wesley.Jurafsky, D., & Martin, J. H. (2023). *Speech and language processing* (3rd ed.). Prentice Hall.