Universidad Nacional Mayor de San Marcos

Facultad de Ingeniería de Sistemas e Informática

Tema: Evaluador y Validador Prefijo con Autómata de Pila (AP)

Curso: Teoría de la Computación

Autor(es):

- A
- Condo Tiquillahuanca Jean Phool
- A
- A

Docente: VICTOR HUGO BUSTAMANTE OLIVERA

Fecha:

1 de noviembre – 2025

INDICE

2.0 Resumen

El presente proyecto consistió en el diseño, implementación y validación de un software capaz de evaluar expresiones matemáticas en **Notación Prefija (Polaca)**. El desarrollo se llevó a cabo utilizando el lenguaje de programación C++, transitando desde un prototipo básico de interacción por consola hasta una aplicación robusta basada en procesamiento de archivos (batch processing) compatible con entornos Linux y Windows.

El núcleo del proyecto se centró en la aplicación práctica de la **Teoría de Autómatas y Lenguajes Formales**. Para ello, se estructuró la solución en tres capas fundamentales:

1. **Capa de Entrada/Salida:** Se implementó el manejo de argumentos por línea de comandos (argc, argv) para la lectura de archivos de texto plano, permitiendo la automatización de pruebas y la integración con scripts de sistema.
2. **Capa de Análisis Léxico y Sintáctico (PDA):** Se diseñó un Autómata de Pila (Pushdown Automaton) que simula el comportamiento de un diagrama de estados (JFLAP). Este módulo recorre los tokens de entrada, valida que pertenezcan al alfabeto definido y verifica la estructura gramatical mediante transiciones de estado y operaciones de apilamiento/desapilamiento simbólico.
3. **Capa de Evaluación Matemática:** Se desarrolló una estructura de datos tipo Pila (Stack) dinámica con punteros. Esta estructura realiza el cálculo numérico real, operando sobre los valores validados y respetando la jerarquía de la notación prefija (evaluación de derecha a izquierda).

El resultado final es un compilador/intérprete ligero que no solo calcula resultados, sino que genera una traza detallada de las transiciones del autómata, permitiendo visualizar la lógica interna de aceptación o rechazo de cada cadena procesada.

3.0 Introducción

En el ámbito de la teoría de lenguajes formales y el diseño de compiladores, el procesamiento de expresiones matemáticas requiere mecanismos precisos para interpretar la jerarquía y precedencia de las operaciones. La **Notación Polaca (Prefija)**, desarrollada por Jan Łukasiewicz, ofrece una alternativa eficiente a la notación infija tradicional al situar el

operador antes de sus operandos, eliminando así la necesidad de paréntesis y la ambigüedad en la precedencia.

El presente proyecto tiene como objetivo el diseño e implementación de un sistema evaluador y validador de expresiones prefijas, fundamentado en el modelo teórico del **Autómata de Pila (AP)**. A diferencia de un enfoque monolítico, y atendiendo a principios de ingeniería de software y diseño de compiladores, la solución se estructura en dos etapas claramente diferenciadas:

1. **Categorización Léxica (Tokenización):** Una fase inicial encargada de procesar la cadena de entrada para identificar, validar y clasificar los componentes léxicos. En esta etapa, se discriminan los símbolos para asignarles una categoría específica: **Operandos** (números enteros o decimales) y **Operadores** (suma, resta, multiplicación, división). Esto asegura que el evaluador reciba únicamente tokens válidos.
2. **Evaluación Sintáctica y Semántica:** Una vez categorizados los tokens, un Autómata de Pila procesa la secuencia (leída de derecha a izquierda, conforme a la naturaleza de la notación prefija). Utilizando una estructura de datos LIFO (*Last-In, First-Out*), el autómata gestiona la memoria auxiliar para resolver las subexpresiones aritméticas, reduciendo la pila hasta obtener un resultado único o rechazar la cadena por inconsistencia estructural.

Esta implementación no solo busca obtener el resultado numérico de una operación, sino demostrar la aplicación práctica de las estructuras de datos lineales y los autómatas en la validación de lenguajes libres de contexto, garantizando la robustez ante errores sintácticos (como falta de operandos) y errores matemáticos (como la división por cero).

4.0 Marco Teórico

En esta sección se establecen los fundamentos matemáticos y conceptuales necesarios para el diseño del evaluador y validador de expresiones prefijas. Se abordan las definiciones formales de lenguajes, la estructura de los Autómatas de Pila (AP) y las propiedades de la notación polaca.

4.1. Lenguajes y gramáticas formales

Definición de lenguaje formal

Un lenguaje formal L es un conjunto de cadenas de símbolos construidas sobre un alfabeto finito Σ. En el contexto de la Jerarquía de Chomsky, los lenguajes se clasifican según la complejidad de la gramática que los genera. Para el análisis de expresiones aritméticas estructuradas, nos centramos en los Lenguajes Libres de Contexto (Tipo 2), los cuales permiten estructuras anidadas y recursivas que los autómatas finitos simples no pueden reconocer debido a su falta de memoria auxiliar.

Gramáticas Libres de Contexto (GLC)

Una Gramática Libre de Contexto es una cuádrupla $G = (V, \Sigma, R, S)$, donde:

- V: Conjunto finito de variables o símbolos no terminales.
- Σ: Conjunto finito de símbolos terminales (el alfabeto de la cadena).
- R: Conjunto finito de reglas de producción de la forma $A \rightarrow \alpha$, donde $A \in V$ y $\alpha \in (V \cup \Sigma)^*$.
- S: Símbolo inicial ($S \in V$).

Producciones para expresiones aritméticas en prefijo

Para modelar una expresión prefija (donde el operador precede a los operandos), se puede definir una gramática recursiva simple. Sea Σ = {+, -, *, /, n}, donde n representa un número, las reglas de producción P serían:

$E \rightarrow + E E \mid - E E \mid * E E \mid / E E \mid n$

Esta gramática establece que una expresión (E) es un número o un operador seguido de dos expresiones válidas, capturando perfectamente la naturaleza de la notación polaca.

4.2. Autómata de Pila (AP o PDA)

Los autómatas finitos deterministas tienen limitaciones para reconocer lenguajes que requieren contar o balancear estructuras (como paréntesis o jerarquías de operadores). Para superar esto, se introduce el Autómata de Pila (Pushdown Automaton), que incorpora una memoria auxiliar de acceso LIFO (Last In, First Out).

Definición formal

Formalmente, un autómata de pila se define como una séptupla:

$AP = (Q, \Sigma, \Gamma, \delta, q0, Z0, F)$

Donde:

- Q: Conjunto finito de estados de control.
- Σ: Alfabeto de entrada.
- Γ: Alfabeto de la pila.
- δ: Función de transición $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Pfin(Q \times \Gamma^*)$.
- q0: Estado inicial.
- Z0: Símbolo inicial de la pila.
- F: Conjunto de estados de aceptación.

Funcionamiento general: Pila y transiciones

El funcionamiento del AP depende del estado actual, el símbolo leído y el símbolo en el tope de la pila. Una transición $(p, \gamma) \in \delta(q, a, X)$ significa: estando en el estado q, leyendo a y con X en el tope de la pila, pasar a p, desapilar X y apilar γ.

Aceptación por estado final o pila vacía

Un AP puede aceptar de dos formas:

- Por estado final: cuando, tras procesar la entrada, alcanza un estado de F.
- Por pila vacía: cuando, tras procesar la entrada, la pila queda vacía.

Para este proyecto, se usa aceptación por pila vacía.

4.3. Notación prefija (polaca)

Concepto

La notación prefija fue introducida por Jan Łukasiewicz. Su característica principal es que el operador precede a sus operandos. Por ejemplo: (3 + 4) se escribe como + 3 4.

Operadores y operandos

Los operandos son números o símbolos terminales, mientras que los operadores binarios requieren dos argumentos que pueden ser números o subexpresiones completas.

Reglas de formación válidas

La notación prefija no necesita paréntesis ni precedencia. La validación se facilita recorriendo la expresión de derecha a izquierda, apilando operandos y reduciendo cuando aparece un operador.

Ejemplos

Infija: A + B → Prefija: + A B

Infija: (A + B) * C → Prefija: * + A B C

4.4. Relación entre GLC y AP

Existe una equivalencia entre las Gramáticas Libres de Contexto y los Autómatas de Pila: todo lenguaje generado por una GLC puede ser reconocido por un-AP.

Cómo convertir gramáticas a autómatas de pila

La conversión se basa en simular derivaciones de la gramática: el AP usa la pila para almacenar variables pendientes. Si el tope es una variable, se aplica una producción; si es un terminal, se compara con la entrada.

Por qué las expresiones prefijas son un lenguaje libre de contexto

Las expresiones prefijas requieren emparejar operadores con el número correcto de operandos, un proceso anidado y recursivo. Esto no puede ser manejado por un autómata finito, pero sí por un AP debido a su pila LIFO.

5.0 Diseño del Autómata de Pila

El objetivo de esta sección es formalizar el funcionamiento lógico del validador de expresiones en notación prefija. Dado que la notación prefija (Polaca) se evalúa algorítmicamente leyendo la cadena de derecha a izquierda (Bottom-Up), el Autómata de Pila (AP) se ha diseñado para simular este comportamiento mediante el apilamiento de operandos y la reducción por operadores.

5.1 Gramática Propuesta

Para modelar la estructura sintáctica de las operaciones prefijas, definimos una **Gramática Libre de Contexto (GLC)**.

Sea **G = (N, Σ, P, S)** donde:

- **N (No terminales):** {S}
- **Σ (Terminales):** {a, o}
    - Donde **a** representa un **operando** (número entero o decimal).
    - Donde **o** representa un **operador** (+, -, *, /).
- **S (Símbolo inicial):** S

Producciones (P): La notación prefija se define recursivamente: una expresión es un operando, o un operador seguido de dos expresiones válidas.

S → oSS | a

Esta gramática indica que para que un operador (o) sea válido, debe ir seguido obligatoriamente de dos estructuras válidas (S), mientras que un operando (a) es válido por sí mismo.

5.2 Construcción del AP

Basado en el diagrama de estados implementado, definimos formalmente el Autómata de Pila determinista que valida esta gramática. El autómata utiliza una pila abstracta donde **X** representa un valor numérico cargado y **Z** el fondo de la pila.

Sea **M = (Q, Σ, Γ, δ, q0, Z, F)** donde:

- **Estados (Q):** {q0, q1, q2}
- **Alfabeto de entrada (Σ):** {a, o}
- **Alfabeto de la pila (Γ):** {X, Z}
- **Estado inicial:** q0
- **Símbolo inicial de pila:** Z
- **Estados finales (F):** {q2}

Función de Transición (δ): El autómata procesa la entrada en orden inverso (de derecha a izquierda respecto a la cadena original) para simular la evaluación:

1. **Inicio (Lectura del primer operando):** δ(q0, a, Z) = {(q1, XZ)} *(Se lee 'a', se deja el fondo Z y se apila X)*

2. **Acumulación (Apilar operandos):** $\delta(q1, a, X) = \{(q1, XX)\}$ *(Se lee 'a', se deja la X anterior y se apila una nueva X)*
3. **Reducción (Operación):** $\delta(q1, o, XX) = \{(q1, X)\}$ *(Se lee operador 'o', se consumen dos elementos XX de la pila y se devuelve un resultado X)*
4. **Aceptación (Fin de cadena):** $\delta(q1, \lambda, XZ) = \{(q2, Z)\}$ *(Si al terminar la cadena (lambda), la pila tiene exactamente un elemento sobre el fondo (XZ), se acepta)*

5.3 Ejecución paso a paso

A continuación, se presenta la traza de ejecución del autómata para la cadena de entrada del ejemplo: + * 5 6 3

Recordemos que el algoritmo lee el archivo en orden inverso para validar la notación prefija.

Entrada original: + * 5 6 3

Orden de lectura del AP: 3 → 6 → 5 → * → +

Paso	**Token Leído**	**Tipo (Σ)**	**Estado**	**Pila**	**Transición Aplicada**	**Descripción**
0	-	-	q0	Z	-	Estado Inicial.
1	**3**	a	q0	XZ	q0 → q1 (a, Z; XZ)	Lee primer operando, inicia la pila.
2	**6**	a	q1	XXZ	q1 → q1 (a, X; XX)	Lee operando, apila X.
3	**5**	a	q1	XXXZ	q1 → q1 (a, X; XX)	Lee operando, apila X.
4	*****	o	q1	XXZ	q1 → q1 (o, XX; X)	Lee operador. Desapila 2, apila 1 (Resultado de 5*6).
5	+	o	q1	XZ	q1 → q1 (o, XX; X)	Lee operador. Desapila 2, apila 1 (Resultado de 30+3).

6	**Fin**	λ	q1	Z	q1 → q2 (λ, XZ; Z)	La pila tiene exactamente 1 elemento válido. **ACEPTACIÓN.**

6.0 Implementación del Evaluador/Validador

Esta sección detalla la construcción del software encargado de analizar léxicamente, validar sintácticamente y evaluar matemáticamente las expresiones en notación prefija. La solución fue desarrollada en lenguaje C++ estándar, priorizando la portabilidad y el uso de estructuras de datos dinámicas.

6.1 Modelo lógico / algoritmo

El funcionamiento del programa se basa en un procesamiento por lotes (batch processing) que sigue un flujo secuencial dividido en tres etapas principales: Carga, Validación Léxica y Evaluación Sintáctica/Semántica.

A diferencia de la evaluación infija tradicional, la notación prefija se procesa algorítmicamente recorriendo la expresión de derecha a izquierda. Dado que el archivo de entrada se lee línea por línea (de arriba a abajo), el algoritmo almacena primero los tokens y luego los procesa en orden inverso.

Algoritmo General (Pseudocódigo):

1. **Inicio:** Validar argumentos de entrada (archivos .txt).
2. **Fase 1: Lectura de archivo**
    a. Abrir archivo de entrada.
    b. **Mientras** existan líneas no vacías:
        i. Leer línea.
        ii. Almacenar token en un array.
3. **Fase 2: Evaluación (Recorrido Inverso)**
    a. Inicializar una **Pila** vacía.
    b. Recorrer el Vector desde la última posición hasta la 0:
        i. **Si** es Operando (a): Apilar valor.
        ii. **Si** es Operador (o):
            1. Verificar si Pila tiene 2 elementos. (Si no, Error de Sintaxis).
            2. Desapilar Operando A.

3. Desapilar Operando B.
4. Calcular Resultado = A op B.
5. Apilar Resultado.

iii. Si no es Operando ni Operador, se termina el programa. Es un error léxico.

4. **Cierre:**
   a. Verificar si Pila tiene exactamente 1 elemento final.
   b. **Si** es así: Procede a terminar de vaciar la pila e imprimir el resultado. La cadena es lógicamente aceptada.
   c. **Si no**: No hay más transiciones y la pila no se vacía. No se imprime el resultado.

6.2 Implementación

La implementación se realizó en **C++** utilizando el paradigma estructurado con soporte de tipos de datos abstractos (Structs). No se utilizaron librerías de terceros para la gestión de la pila, implementándose esta de forma manual con punteros para gestionar la memoria dinámicamente.

Componentes clave del código:

- **Gestión de Archivos (<fstream>):** El programa define los nombres de los archivos de entrada y salida, para asegurar consistencia en el código. Los archivos de entrada y salido son nombrados como “documento.txt” y “resultado.txt”, respectivamente
- **Estructura Pila:** Se diseñó un struct que maneja nodos enlazados dinámicamente con el fin construir una Pila. Cada Nodo contiene un valor double y un puntero al siguiente nodo. En el struct Pila se usa un atributo puntero para siempre referenciar al último elemento apilado, útil para los métodos de la estructura. Métodos implementados para la Pila: apilar(), desapilar(), verTope() y verPila().
- **Array de Tokens:** Se utiliza un array para almacenar temporalmente la lectura del archivo, permitiendo el acceso necesario para recorrer la entrada de atrás hacia adelante (necesario para el análisis prefijo).

6.3 Pruebas

Para verificar la robustez del validador, se diseñaron cuatro casos de prueba que cubren los escenarios de éxito, error sintáctico, error léxico y error matemático.

Caso 1: Expresión Prefija Válida

- **Entrada:** + * 5 6 3 (Equivalente a (5 * 6) + 3)
- **Comportamiento esperado:** El autómata acepta la sintaxis y la pila calcula 30 + 3.
- **Archivo de Salida:**

3	Z	q0	q1	XZ	XZ
6	X	q1	q1	XX	XXZ
5	X	q1	q1	XX	XXXZ
*	XX	q1	q1	X	XXZ
+	XX	q1	q1	X	XZ
λ	XZ	q1	q2	Z	Z
λ	Z	q2	q2	λ	λ

33

Caso 2: Error Sintáctico (Faltan Operandos)

- **Entrada:** + 5 (El operador suma requiere dos números, solo hay uno)
- **Comportamiento esperado:** Al intentar operar el +, la pila solo tiene un elemento. El PDA detecta transición inválida. El programa se detiene y la pila no se vacía
- **Archivo de Salida:** No procede porque no hay suficientes X en el

5	Z	q0	q1	XZ	XZ

Caso 3: Error Sintáctico (Sobran Operandos)

- **Entrada:** + 5 5 5 (Se suman 5+5, pero sobra un 5 en la pila al final)
- **Comportamiento esperado:** El autómata finaliza, pero la pila termina con más de un elemento (XZZ en lugar de XZ).
- **Salida del Programa:**

5	Z	q0	q1	XZ	XZ
5	X	q1	q1	XX	XXZ
5	X	q1	q1	XX	XXXZ
+	XX	q1	q1	X	XXZ

Caso 4: Error Matemático

- **Entrada:** / 10 0 (División por cero)
- **Comportamiento esperado:** Sintaxis válida, pero error en tiempo de ejecución controlado durante la evaluación. El programa se detiene y la pila no se vacía.

- **Salida del Programa:**

0	Z	q0	q1	XZ	XZ
10	X	q1	q1	XX	XXZ

7. Análisis y Discusión

En esta sección se examinan las propiedades teóricas del lenguaje procesado, la eficiencia del autómata implementado y la justificación de la arquitectura elegida frente a otras alternativas de análisis sintáctico.

7.1. ¿Por qué las expresiones prefijas constituyen un Lenguaje Libre de Contexto?

Un lenguaje se considera **Libre de Contexto** (Context-Free Language, CFL) si puede ser generado por una Gramática Libre de Contexto (GLC) y reconocido por un Autómata de Pila. Las expresiones aritméticas en notación prefija no pueden ser reconocidas por un Autómata Finito Determinista (AFD) simple debido a su capacidad de **anidamiento recursivo arbitrario**.

Por ejemplo, en la expresión + * + 2 3 4 5, el operador + inicial debe "esperar" a que se resuelva toda la subexpresión * + 2 3 4 antes de poder operar con 5. Un AFD, al carecer de memoria auxiliar ilimitada, no puede rastrear la profundidad de estas operaciones pendientes. La estructura gramatical subyacente E -> Op E E demuestra que la sustitución de variables es independiente del contexto que las rodea, cumpliendo con la definición formal establecida en la teoría de autómatas [1].

7.2. Suficiencia del Autómata de Pila (AP)

El Autómata de Pila es el modelo computacional teóricamente necesario y suficiente para este problema. Según la jerarquía de Chomsky, la adición de una pila a un control finito otorga al autómata la capacidad de manejar estructuras jerárquicas [1].

En nuestra implementación, la pila actúa como el mecanismo de control de flujo que gestiona la **recursividad** inherente de la notación polaca:

- **Almacenamiento temporal:** Permite guardar operandos (n) que aún no pueden ser operados.
- Reducción: Permite recuperar estos operandos en el orden inverso (LIFO) tan pronto como un operador (Op) lo requiere.

Dado que la validación depende exclusivamente de verificar el balance entre operadores y operandos (similar al problema de paréntesis balanceados), el AP cubre todos los casos posibles sin necesidad de la potencia de una Máquina de Turing.

7.3. Ventajas de la notación prefija en el diseño de Parsers

La elección de la notación polaca (prefija) simplifica drásticamente la fase de análisis sintáctico (parsing) en comparación con la notación infija tradicional.

1. **Eliminación de la Ambigüedad:** Como señala Trejos Buriticá [2], la notación polaca elimina la necesidad de paréntesis y reglas de precedencia (como la jerarquía de multiplicación sobre suma). El orden de ejecución está dictado unívocamente por la posición de los tokens.
2. **Eficiencia en la Evaluación:** Permite una evaluación lineal directa. Al no existir ambigüedad, el autómata no necesita realizar *backtracking* (retroceso) ni construir árboles de sintaxis complejos en memoria; puede calcular o validar el resultado en una sola pasada.

7.4. Complejidad Temporal del Proceso

El algoritmo implementado basado en el AP presenta una complejidad temporal lineal, denotada como **O(n)** donde n es el número de tokens (símbolos) en la cadena de entrada.

- **Lectura:** Cada token de la cadena se lee exactamente una vez.
- **Operaciones de Pila:** En cada paso, se realizan operaciones de push o pop. Dado que cada operando entra a la pila una vez y sale una vez, el costo amortizado de las operaciones de pila es constante O(1).
- **Conclusión:** El tiempo de ejecución crece proporcionalmente a la longitud de la expresión, lo cual es óptimo para validadores sintácticos.

7.5. Relación con otros modelos de análisis (LR y Recursión)

E l enfoque utilizado en este proyecto (evaluación de derecha a izquierda con pila) es funcionalmente equivalente a un **parser predictivo** o a un esquema de **recursión descendente**.

- Si leemos la notación prefija de izquierda a derecha (+ * 3 4 5), la estructura natural sugiere un parser recursivo descendente (Top-Down), donde el operador llama a funciones para obtener sus dos operandos.
- Sin embargo, al invertir la lectura (o leer de derecha a izquierda), transformamos el problema en uno que puede ser resuelto por un autómata **Bottom-Up** simple: los operandos (hojas del árbol) se apilan y se reducen hacia la raíz (el operador principal). Esta dualidad demuestra la flexibilidad de los Autómatas de Pila para implementar diferentes estrategias de análisis sintáctico sobre lenguajes libres de contexto [1].

8.0. Conclusiones

Tras finalizar el desarrollo e implementación del evaluador de notación prefija, se obtienen las siguientes conclusiones:

1. **Eficacia de los Autómatas de Pila en el Parsing:** Se demostró que los Autómatas de Pila (PDA) son la herramienta computacional idónea para validar Gramáticas Libres de Contexto (GLC), como las expresiones matemáticas. A diferencia de un Autómata Finito Determinista (DFA) simple que no tiene memoria, el PDA utiliza la pila para "recordar" operandos pendientes, asegurando que cada operador tenga sus dos argumentos correspondientes antes de proceder.
2. **Ventajas de la Notación Prefija:** La implementación evidenció por qué la notación prefija (y postfija) es preferida en computación sobre la notación infija tradicional. Al eliminar la necesidad de paréntesis y reglas de precedencia complejas, el algoritmo de evaluación se simplifica linealmente, reduciendo la complejidad computacional y el uso de memoria.
3. **Importancia de las Estructuras de Datos Dinámicas:** La construcción manual de la estructura Pila (utilizando struct y punteros) permitió un control total sobre la gestión de memoria. Esto valida la importancia de comprender las estructuras de datos fundamentales para manipular flujos de información LIFO (Last In, First Out), esenciales en la construcción de compiladores.

9.0 Referencias Bibliográficas Utilizadas

- **[1]** Universidad de Huelva. *Tema 4: Autómatas de pila*. Departamento de Tecnologías de la Información.

- **[2]** Trejos Buriticá, O. I., & Morales González, A. I. (2013). *La notación polaca como estrategia para aproximarse al pensamiento funcional y al pensamiento matemático*. Universidad de Manizales.