UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Facultad de Ingeniería de Sistemas e Informática

Evaluador y Validador Postfijo con Autómata de Pila

Proyecto Final

Curso:	Teoria de la Computación
Docente:	Víctor Hugo Bustamante Olivera
Integrantes:	Diego Sotelo
	Alexis Gonzales
	Paolo Villavicencio
	Álvaro Salazar
Fecha:	18 de noviembre de 2025

Índice

1. Resumen

Este proyecto presenta el diseño e implementación de un evaluador y validador de expresiones aritméticas en notación posfija (RPN) utilizando un Autómata de Pila (AP) como modelo formal. Partimos de la definición teórica de un AP y de las gramáticas libres de contexto para describir el proceso de evaluación mediante operaciones de *push* y *pop* sobre la pila, estableciendo criterios formales de aceptación y rechazo de las cadenas. El sistema procesa una secuencia de *tokens* (operandos numéricos y operadores binarios $+, -, *, /$), verifica si la expresión es sintácticamente correcta y detecta errores como operandos insuficientes o división entre cero. Finalmente, se documenta la formalización del autómata, su modo de ejecución y se discute cómo este enfoque evidencia la relación entre la teoría de autómatas y la evaluación de expresiones aritméticas en contextos computacionales.

Palabras clave: Autómata de pila, Notación postfija, RPN, Evaluación de expresiones, Teoría de la computación.

Abstract

This project presents the design and implementation of an arithmetic expression evaluator and validator for postfix notation (RPN) using a Pushdown Automaton (PDA) as the formal model. We begin with the theoretical definition of a PDA and context-free grammars to describe the evaluation process through *push* and *pop* operations on the stack, establishing formal criteria for acceptance and rejection of input strings. The system processes a sequence of tokens (numeric operands and binary operators $+, -, *, /$), verifies whether the expression is syntactically correct, and detects errors such as insufficient operands or division by zero. Finally, we document the formalization of the automaton, its execution behavior, and discuss how this approach illustrates the connection between automata theory and the evaluation of arithmetic expressions in computational contexts.

Keywords: Pushdown automaton, Postfix notation, RPN, Expression evaluation, Theory of computation.

2. Introducción

2.1. Descripción del Problema

Evaluar expresiones en notación infija es un poco complicado porque hay que pensar en qué operación va primero y en cómo se agrupan los paréntesis. En cambio, la Notación Postfija (RPN) es más directa: primero van los números y después el operador. Esto permite ir resolviendo todo paso a paso usando una pila. El proceso es simple, cada vez que aparece un número, lo metemos en la pila; y cuando aparece un operador, sacamos los dos números de arriba, hacemos la operación y volvemos a guardar el resultado.

El proyecto trata justamente de eso: dado un conjunto de elementos escritos en RPN (números enteros o decimales y los operadores $+, -, *, /$), calcular cuánto vale la expresión y, al mismo tiempo, verificar si está bien escrita. La expresión será válida si al final queda exactamente un valor en la pila y si no pasan errores, como que falten números para operar o intentar dividir entre cero.

2.2. Planteamiento del Proyecto

Se implementará un simulador de Autómata de Pila (AP). La idea es que el programa reciba una lista de tokens ya identificados y vaya usando la pila de la forma básica: cuando llega un número, lo ponemos en la pila (push); cuando aparece un operador que necesita dos valores, sacamos los dos de arriba (pop), hacemos la operación y guardamos el resultado otra vez.

El AP aceptará la expresión siempre que esté bien escrita en notación postfija y que, al terminar de leer todos los tokens, la pila quede con un único valor. Si ocurre algo extraño, como que falten números para operar o que queden valores de más en la pila, entonces la expresión se rechaza. Además, el sistema debe ser capaz de detectar errores como intentar dividir entre cero.

2.3. Modelo y Método

Modelo. Para este proyecto usamos un Autómata de Pila (AP) descrito de forma general como una séptupla, pero en palabras simples es solo un sistema con estados, una entrada (los números y operadores), y una pila donde vamos guardando valores. El estado inicial es donde empieza todo, y el estado de aceptación es donde el AP determina que la expresión está correcta. Mediante el proceso la pila usa operaciones básicas de *push* y *pop* para recorrer toda la expresión.

Método. El algoritmo recorre los tokens una sola vez. Cuando aparece un número, simplemente lo metemos a la pila (*push*). Cuando aparece un operador como $+, -, *$ o $/$, el programa revisa si hay por lo menos dos valores en la pila; si los hay, saca ambos (*pop*), hace la operación y vuelve a meter el resultado. Si falta algún valor para operar, se rechaza la expresión. Al terminar de leer todos los tokens, solo se acepta si queda exactamente un valor en la pila. Además, si el operador es una división y el divisor es cero, se detiene todo y se reporta un error. El proceso completo toma tiempo lineal respecto al tamaño de la expresión y usa una cantidad de memoria proporcional a lo que llegue a crecer la pila.

3. Objetivos

3.1. Objetivo general

Desarrollaremos un programa que sea capaz de evaluar y validar expresiones que estén en notación postfija, nos basaremos en un autómata de pila que sea capaz de calcular el resultado solo si la expresión es válida, en caso de que no, la expresión sería rechazada, además colocaremos los pasos y detalles del proyecto en nuestra página de github en el `README.md`

Objetivos específicos

- Explicar, desde la teoría de autómatas de pila y gramáticas libres de contexto, cómo se puede modelar el proceso de evaluar expresiones aritméticas en notación postfija (RPN).
- Crear e implementar un autómata de pila que pueda revisar si una expresión postfija es correcta y, además, calcular su resultado usando operandos numéricos y operadores básicos $(+, -, *, /)$.
- Identificar y mostrar errores comunes en las expresiones ingresadas, como falta de operandos, uso de símbolos no permitidos o intentos de división entre cero.
- Construir un módulo interactivo que permita al usuario escribir una expresión y ver, paso a paso, cómo el autómata la procesa y obtiene el resultado.
- Elaborar y aplicar varios casos de prueba para comprobar que el evaluador/validador funciona correctamente, registrando y analizando los resultados obtenidos.

4. Marco teórico

4.1. ¿Qué es un autómata de pila (AP)?

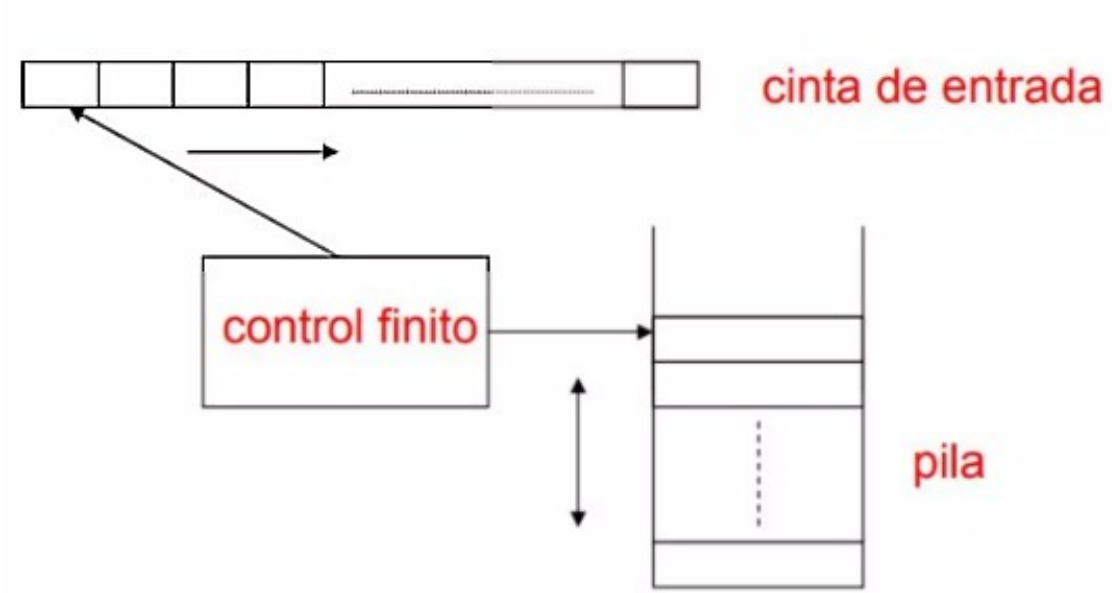
Un autómata de pila es un modelo matemático de una máquina con un estado finito y una memoria auxiliar en forma de pila (stack), que opera siguiendo una lógica de "último en entrar, primero en salir"(LIFO). Se utiliza para reconocer lenguajes libres de contexto. A diferencia de los autómatas finitos, estos pueden procesar y aceptar cadenas más complejas al tener la capacidad de almacenar y recuperar símbolos de su pila.

Definición de un Autómata de pila

APD $= \langle E, A, P, \delta, e_0, Z_0, F \rangle$

E: Conjunto finito de estados,
A: Alfabeto o conjunto finito de símbolos de la cinta de entrada,
P: Alfabeto o conjunto finito de símbolos de la pila. $P \cap A = \emptyset$
δ: función de transición de estados
e_0: Estado inicial $e_0 \in E$.
Z_0: Símbolo distinguido $Z_0 \in P$
F: Conjunto de estados finales o estados de aceptación. $F \subseteq E$.

Represetación gráfica:



¿Cómo una palabra es aceptada?

Para que una palabra de entrada sea aceptada en un AP se deben cumplir todas las condiciones siguientes:

1. La palabra de entrada se debe haber agotado (consumido totalmente).
2. El AP se debe encontrar en un estado final.
3. La pila debe estar vacía.

4.2. Estado de Transiciones de un AP

En un autómata de pila, la *función de transición* se define como

$$\delta : E \times (A \cup \{\varepsilon\}) \times P \to E \times P^*,$$

donde:

- E es el conjunto de estados.
- A es el alfabeto de entrada.
- P es el alfabeto de la pila.
- P^* es el conjunto de todas las cadenas (posiblemente vacías) de símbolos de pila.
- ε representa la transición sin consumir símbolo de entrada.

Una transición se suele escribir como

$$\delta(q, a, X) = (p, \gamma),$$

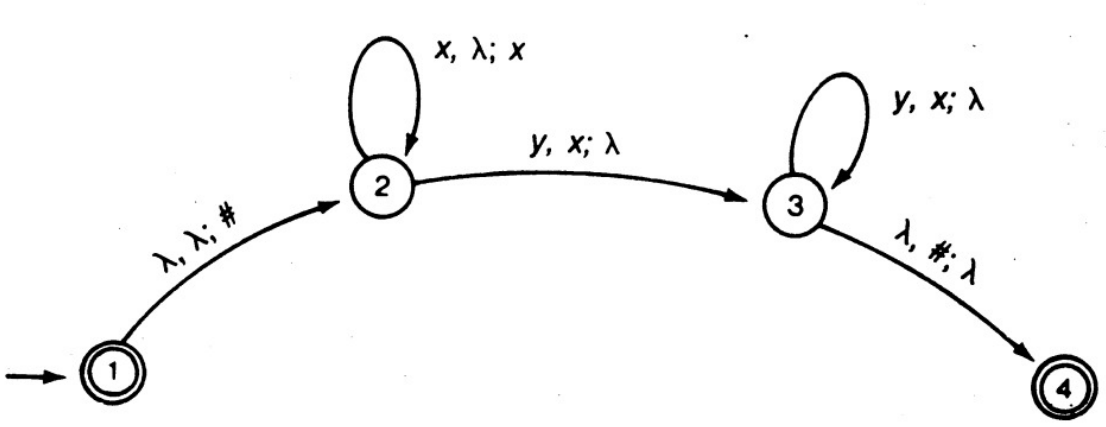
y se interpreta así:

- El autómata está en el estado q.
- Lee el símbolo a de la cinta de entrada (o ε si no consume nada).
- Observa el símbolo X en la cima de la pila.
- Cambia al estado p y reemplaza X por la cadena γ en la pila (es decir, desapila X y apila los símbolos de γ; si $\gamma = \varepsilon$, simplemente desapila X).

De manera general, una transición puede:

- **Apilar** símbolos (cuando γ contiene más símbolos que X).
- **Desapilar** símbolos (cuando $\gamma = \varepsilon$).
- **Reemplazar** el símbolo de la cima por otros (cuando γ tiene uno o varios símbolos distintos de X).
- **No consumir entrada** (cuando $a = \varepsilon$), modificando solo el estado y/o la pila.

Ejemplo de diagrama de transiciones:



Estado actual	Símbolo entrada	Símbolo pila	Nuevo estado	Operación en la pila
1	λ	λ	2	apilar #
2	x	λ	2	apilar x
2	y	x	3	desapilar x
3	y	x	3	desapilar x
3	λ	#	4	desapilar #

4.3. Qué es la notación postfija (RPN) y por qué es adecuada para un AP

La notación postfija, también conocida como notación polaca inversa (RPN), es una forma de escribir expresiones aritméticas en la que los operadores aparecen después de sus operandos. En lugar de usar la notación infija tradicional (la que usamos normalmente), donde los operadores van entre los valores, en postfija todo sigue un orden lineal sin necesidad de paréntesis.

Por ejemplo, la expresión infija:

a * b / c

se escribe en postfija como:

ab*c/

Si hubiera paréntesis en la forma infija, en postfija sólo afectan cuando realmente cambian el orden de evaluación. Por ejemplo:

a/b + cd - ef se convierte en **ab/cd*+ef*-**

pero

a/(b+c) * (d-e) * f pasa a ser **abc+/de-f**

Una de las principales ventajas de la notación postfija es que no necesita paréntesis ni reglas de prioridad. Todo se resuelve únicamente siguiendo el orden en el que aparecen los símbolos. Esto hace que el proceso de evaluación sea muy directo.

Algunas ventajas claves:

- No requiere paréntesis en ningún caso.
- No obliga a definir niveles de prioridad entre operadores.
- Es muy fácil de evaluar secuencialmente.

¿Por qué es adecuada para un AP?

La notación postfija encaja muy bien con un Autómata de Pila porque su estructura es básicamente compatible con el comportamiento de una pila. Como no hay paréntesis ni reglas de precedencia que manejar, el AP puede centrarse únicamente en verificar la forma correcta de la expresión usando operaciones simples de *push* y *pop*. En otras palabras, la memoria tipo pila del autómata alcanza para controlar toda la lógica necesaria para validar que la expresión postfija esté bien construida, sin requerir mecanismos adicionales.

4.4. Gramática libre de contexto

5. Desarrollo del programa

En el desarrollo del programa se explica cómo el modelo implementado logra evaluar expresiones en notación postfija usando un autómata de pila. Básicamente, se muestra cómo la definición formal del autómata se convierte en un proceso que realmente funciona, siguiendo la lógica de cada transición . Así se puede ver, de manera sencilla, cómo cada regla teórica termina siendo una acción concreta sobre la pila.

5.1. Formalización del autómata

El autómata de pila diseñado se expresa mediante la séptupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$$

donde:

$Q = \{q_1\}$ (conjunto de estados)

$\Sigma = \{V, +, -, *, /\}$ (alfabeto de entrada)

$\Gamma = \{X, Z_0\}$ (alfabeto de pila)

$q_0 = q_1$ (estado inicial)

Z_0 símbolo de fondo de pila

$F = \emptyset$ (aceptación por pila vacía)

El funcionamiento completo está definido por la función de transición:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$$

que básicamente indica qué apila o qué desapila según el símbolo leído y lo que haya en la pila:

$$\begin{aligned}
\delta(q_1, \varepsilon, Z_0) &= (q_1, Z_0) && \text{Inicialización.} \\
\delta(q_1, V, Z_0) &= (q_1, XZ_0) && \text{Primer operando apilado.} \\
\delta(q_1, V, X) &= (q_1, XX) && \text{Apila operandos sucesivos.} \\
\delta(q_1, +, XX) &= (q_1, X) && \text{Suma: desapila dos, apila uno.} \\
\delta(q_1, -, XX) &= (q_1, X) && \text{Resta: desapila dos, apila uno.} \\
\delta(q_1, *, XX) &= (q_1, X) && \text{Multiplica: desapila dos, apila uno.} \\
\delta(q_1, /, XX) &= (q_1, X) && \text{Divide: desapila dos, apila uno.} \\
\delta(q_1, \varepsilon, XZ_0) &= (q_1, \varepsilon) && \text{Pila vacía: aceptación.}
\end{aligned}$$

5.2. Criterio de aceptación

El autómata acepta una expresión siempre y cuando, al terminar de leer toda la entrada, la pila quede completamente vacía (es decir, ya no quede ni el símbolo Z_0):

$$\delta^*(q_1, w, Z_0) = (q_1, \varepsilon), \quad w \in L(M)$$

Esto asegura que:

- Todos los operadores encontraron sus operandos.
- No quedaron operandos sin usar al finalizar.

5.3. Representación del proceso

Para mostrar cómo funciona todo el mecanismo, el proceso de evaluación se describe como una secuencia de configuraciones del autómata:

$$(q, \text{entrada}, \text{pila}) \Rightarrow^* (q', \varepsilon, \varepsilon)$$

Por ejemplo, para evaluar la expresión:

$$3\ 4\ +\ 2\ *$$

el autómata avanza así:

$$\begin{aligned}
(q_1, 3\ 4\ +\ 2\ *, Z_0) &\Rightarrow (q_1, 4\ +\ 2\ *, XZ_0) \Rightarrow (q_1, +\ 2\ *, XXZ_0) \\
&\Rightarrow (q_1, 2\ *, XZ_0) \Rightarrow (q_1, *, XXZ_0) \Rightarrow (q_1, \varepsilon, \varepsilon)
\end{aligned}$$

que muestra claramente cómo la pila va creciendo con operandos y reduciéndose cada vez que aparece un operador.

5.4. Resumen operativo

El comportamiento del autómata se resume de forma sencilla así:

- **Si se lee un operando:** se apila un símbolo X.
- **Si se lee un operador:** se desapilan dos X y se vuelve a apilar uno.
- **Si se llega al final con pila XZ_0:** la cadena se acepta.

Cuadro 1: Resumen equivalente de las transiciones del autómata

#	Entrada	Cima de pila	Acción	Nuevo contenido
1	ε	Z_0	Inicializa la pila	Z_0
2	V	Z_0	Guarda primer operando	XZ_0
3	V	X	Guarda nuevo operando	XX
4	$+$	XX	Suma (pop 2, push 1)	X
5	$-$	XX	Resta (pop 2, push 1)	X
6	$*$	XX	Multiplicación (pop 2, push 1)	X
7	$/$	XX	División (pop 2, push 1)	X
8	ε	XZ_0	Vacía la pila	ε

Interpretación: El autómata reproduce la lógica típica de las expresiones postfijas: cada operando hace crecer la pila y cada operador binario la reduce. Cuando la pila queda vacía al final, significa que la expresión estaba correctamente formada y pudo evaluarse sin problemas.

5.5. Análisis del funcionamiento

Finalmente, los resultados muestran que el programa se comporta justo como se esperaba del autómata de pila que se diseñó. Cada transición δ termina convirtiéndose directamente en una acción de apilar o desapilar, y el criterio de aceptación funciona tal como fue pensado: al final, la pila tiene que quedar vacía para asegurar que la expresión se procesó correctamente.

6. Resultados

7. Conclusiones