UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Facultad de Ingeniería de Sistemas e Informática

Evaluador y Validador Postfijo con Autómata de Pila (AP)

Proyecto Final

Curso:	Teoria de la Computación
Docente:	Víctor Hugo Bustamante Olivera
Integrantes:	Diego Sotelo
	Alexis Gonzales
	Paolo Villavicencio
	Álvaro Salazar
Fecha:	1 de diciembre 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 la corrección sintáctica y detecta errores semánticos como la división entre cero. Adicionalmente, se implementó una funcionalidad de entrada/salida de archivos: el programa lee expresiones desde un archivo de texto generado por el usuario, las evalúa y produce automáticamente dos archivos de salidas, uno con la evolución detallada de la pila y otro con el resultado calculado o con el mensaje de ERROR en caso de fallos sintácticos o aritméticos.

Palabras clave: Autómata de pila, Notación postfija, RPN, Evaluación de expresiones, Manejo de archivos, 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 syntactic correctness, and detects semantic errors such as division by zero. Additionally, a file input/output functionality was implemented: the program reads expressions from a user-generated text file, evaluates them, and automatically produces two output files—one containing the detailed stack evolution and the other displaying the calculated result or an ERROR message in case of syntactic or arithmetic errors.

Keywords: Pushdown automaton, Postfix notation, RPN, Expression evaluation, File handling, 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) que no solo funcione mediante ingreso manual, sino que también integre un módulo de gestión de archivos. El programa será capaz de aceptar un archivo de texto (.txt) creado por el usuario que contenga la expresión postfija.

El AP procesará dicha entrada utilizando las operaciones de pila (*push* y *pop*). Si la expresión es correcta, el sistema generará dos nuevos archivos de texto, uno con la evolucion de la pila y otro con el resultado numérico. En caso de detectar inconsistencias (falta de operandos, caracteres inválidos) o errores matemáticos (división entre cero), el archivo de salida reportará explícitamente el estado de fallo con el mensaje de ERROR.

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

Desarrollar un evaluador y validador de expresiones en notación postfija basado en Autómata de Pila, capaz de calcular resultados y rechazar entradas inválidas, integrando un módulo de procesamiento de archivos de texto para la automatización de la entrada y salida de datos.

Objetivos específicos

- Formalizar el modelo de Autómata de Pila para la evaluación de expresiones RPN utilizando la teoría de gramáticas libres de contexto.
- Implementar la lógica de validación sintáctica y aritmética (detección de división entre cero) mediante operaciones de pila.
- Desarrollar un módulo de persistencia que permita la lectura de expresiones desde archivos de texto externos y la generación de reportes de salida, escribiendo el resultado o el mensaje de error según corresponda.
- Detectar y reportar errores como operandos insuficientes, caracteres no válidos o pila residual mayor a uno.
- Documentar los casos de prueba y el comportamiento del sistema ante entradas correctas e incorrectas.

4. Marco teórico

4.1. Qué es un autómata de pila (AP)

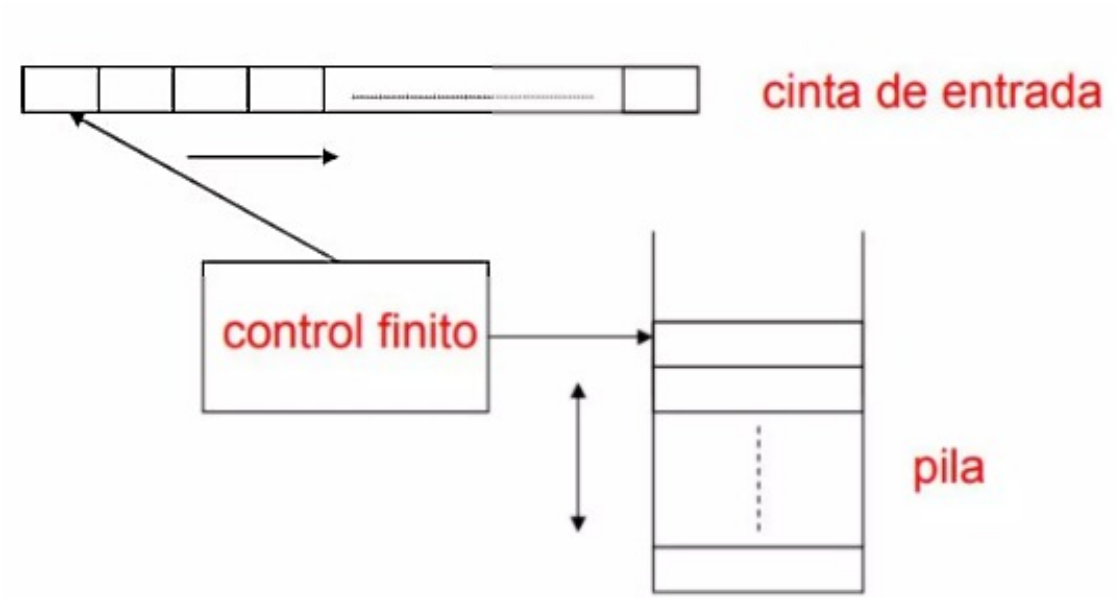
Un autómata de pila es un modelo matemático de una máquina con un estado finito y una memoria auxiliar en forma de pila (stack), que opera siguiendo una lógica de "último en entrar, primero en salir"(LIFO). Se utiliza para reconocer lenguajes libres de contexto. A diferencia de los autómatas finitos, 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 $= \langle E, A, P, \delta, q_0, Z_0, F \rangle$

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
q_0: Estado inicial $q_0 \in E$.
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

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

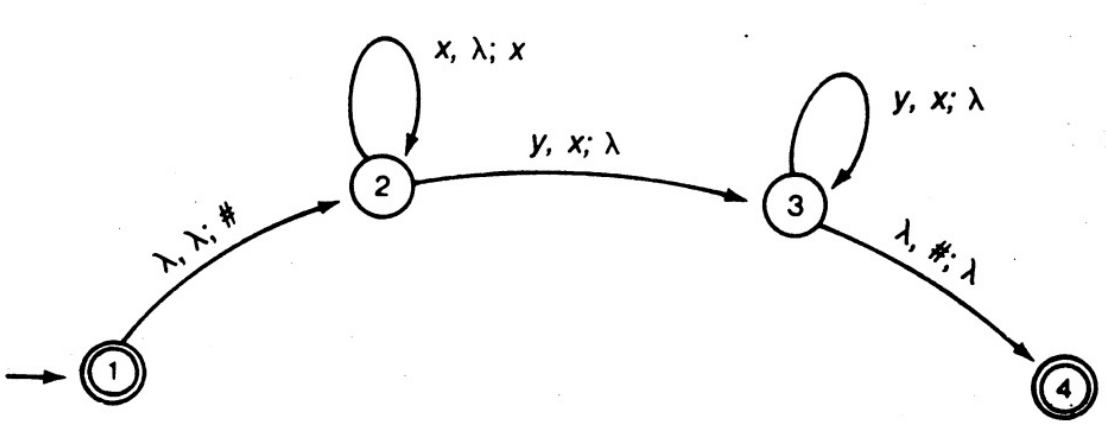
y se interpreta así:

- 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

Una expresión aritmética en notación postfija (o notación polaca inversa) es una secuencia formada por símbolos de dos tipos diferentes: operadores (para simplificar, consideraremos únicamente los operadores aritméticos binarios +, -, * y /) y operandos (para simplificar pensaremos en identificadores de una sola letra). Cada operador se escribe detrás de sus operandos.

Por ejemplo, a la expresión siguiente, escrita en la notación habitual (infija):

a*b/c

Le corresponde la siguiente expresión en notación postfija:

ab*c/

Si en la expresión infija aparecen paréntesis, estos cambian la correspondiente expresión postfija sólo si los paréntesis alteran el orden de prioridad de los operadores:

a/b+c*d-e*f, traducido a notación postfija es: **ab/cd*+ef*-**

a/(b+c)*(d-e)*f, se traduce en cambio por: **abc+/de-*f***

Tres ventajas importantes de la notación postfija frente a la convencional infija son las siguientes:

- En notación postfija nunca son necesarios los paréntesis.
- En notación postfija no es necesario definir prioridades entre operadores.
- Una expresión postfija puede evaluarse de forma muy sencilla.

¿Porqué es adecuada para un AP?

La razón por la cual la notación postfija es adecuada para un Autómata de Pila es gracias a que su estructura elimina la necesidad de manejar paréntesis y prioridades complejas, y se basa directamente en operaciones de pila (push y pop). Esto permite que el autómata modele de forma sencilla la verificación de expresiones postfijas válidas, utilizando únicamente su memoria tipo pila para controlar la forma y la consistencia de la expresión analizada.

4.4. Gramática libre de contexto

4.4.1. Definición Formal

Una **Gramática Libre de Contexto** (GLC), o G, se define formalmente como una 4-tupla:

$$G = (V, \Sigma, P, S)$$

Donde:

- V es un conjunto finito de **variables** (o símbolos no terminales).
- Σ es un conjunto finito de **símbolos terminales** (el alfabeto).
- Se requiere que V y Σ sean disjuntos: $V \cap \Sigma = \emptyset$.
- P es un conjunto finito de **reglas de producción** (o producciones).
- S es el **símbolo inicial** (o axioma), y debe pertenecer al conjunto de variables: $S \in V$.

4.4.2. Reglas de Producción

El término "libre de contexto"se refiere a la forma de las reglas en P. Cada regla debe tener la siguiente estructura:

$$A \to \alpha$$

Donde:

- A debe ser una **única variable**, es decir, $A \in V$.
- α es una cadena de cero o más símbolos, donde cada símbolo puede ser una variable o un terminal. Formalmente, $\alpha \in (V \cup \Sigma)^*$.

El asterisco (*) denota la **clausura de Kleene**, lo que significa çero o más"símbolos. Esto incluye la cadena vacía (denotada como ϵ o λ). Una regla de la forma $A \to \epsilon$ es una **producción épsilon** (o producción vacía).

La restricción clave es que el lado izquierdo de la regla (A) es *siempre* una única variable, sin importar el contexto (los símbolos que la rodean).

4.4.3. Derivaciones

Las reglas de producción se utilizan para generar cadenas mediante un proceso llamado **derivación**.

- **Derivación en un paso ($\Rightarrow$)**
  Sean β y γ cadenas cualesquiera en $(V \cup \Sigma)^*$. Si existe una regla $A \to \alpha \in P$, entonces decimos que la cadena $\beta A \gamma$ deriva en un paso a la cadena $\beta\alpha\gamma$.

  Esto se nota como:

  $$\beta A \gamma \Rightarrow \beta\alpha\gamma$$

- **Derivación en cero o más pasos ($\Rightarrow^*$)**
  La notación $\Rightarrow^*$ representa la **clausura reflexiva y transitiva** de la derivación en un paso ($\Rightarrow$).

  - **Reflexiva:** Para cualquier cadena α, $\alpha \Rightarrow^* \alpha$ (una cadena siempre deriva a sí misma en cero pasos).
  - **Transitiva:** Si $\alpha \Rightarrow \beta$ y $\beta \Rightarrow^* \gamma$, entonces $\alpha \Rightarrow^* \gamma$.

Básicamente, $\alpha \Rightarrow^* \beta$ significa que β puede obtenerse a partir de α aplicando cero o más reglas de producción.

4.4.4. Lenguaje Generado

El **lenguaje generado** por una gramática G, denotado como $L(G)$, es el conjunto de todas las cadenas de *terminales* que pueden ser derivadas a partir del símbolo inicial S.

Formalmente:

$$L(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}$$

- $w \in \Sigma^*$ significa que w es una cadena que consiste *exclusivamente* en símbolos terminales.
- $S \Rightarrow^* w$ significa que w se puede derivar desde S en cero o más pasos.

Cualquier lenguaje que pueda ser generado por una Gramática Libre de Contexto se denomina **Lenguaje Libre de Contexto** (LLC).

5. Desarrollo del programa

En este capítulo se presenta el desarrollo conceptual del modelo de evaluación postfija, basado en el autómata de pila definido en el marco teórico. Se describe la traducción de la definición formal a una estructura operativa, conservando la semántica matemática de cada transición δ.

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:

$$\begin{aligned} Q &= \{q_0\} && \text{(conjunto de estados)} \\ \Sigma &= \{V, +, -, *, /\} && \text{(alfabeto de entrada)} \\ \Gamma &= \{X, Z_0\} && \text{(alfabeto de pila)} \\ q_0 & && \text{(estado inicial)} \\ Z_0 & && \text{(símbolo de fondo de pila)} \\ F &= \emptyset && \text{(aceptación por pila vacía)} \end{aligned}$$

El comportamiento del autómata está determinado por la función de transición $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to Q \times \Gamma^*$, que define las acciones de apilado y desapilado en función del símbolo de entrada y del contenido de la pila.

$$\begin{aligned} \delta(q_0, \varepsilon, Z_0) &= (q_0, Z_0) && \text{Inicialización.} \\ \delta(q_0, V, Z_0) &= (q_0, XZ_0) && \text{Primer operando apilado.} \\ \delta(q_0, V, X) &= (q_0, XX) && \text{Apila operandos sucesivos.} \\ \delta(q_0, +, XX) &= (q_0, X) && \text{Suma: desapila dos, apila uno.} \\ \delta(q_0, -, XX) &= (q_0, X) && \text{Resta: desapila dos, apila uno.} \\ \delta(q_0, *, XX) &= (q_0, X) && \text{Multiplica: desapila dos, apila uno.} \\ \delta(q_0, /, XX) &= (q_0, X) && \text{Divide: desapila dos, apila uno.} \\ \delta(q_0, \varepsilon, XZ_0) &= (q_0, \varepsilon) && \text{Pila vacía: aceptación.} \end{aligned}$$

5.2. Implementación del Módulo de Archivos

Para cumplir con el requerimiento de procesamiento por lotes, se añadió un submódulo de manejo de archivos con la siguiente lógica:

1. **Lectura:** El programa solicita o recibe la ruta de un archivo de texto fuente (`entrada.txt`).
2. **Parsing:** Se lee el contenido del archivo y se tokeniza la cadena de entrada. Se valida que solo contenga caracteres permitidos (dígitos y operadores).
3. **Evaluación Controlada:**
   - Se ejecuta el autómata de pila sobre los tokens.
   - Se capturan excepciones específicas: `ZeroDivisionError` (aritmético) y errores de estructura (pila vacía al intentar operar, pila con más de un elemento al final).
4. **Escritura:**
   - Si la evaluación es exitosa: Se crea dos archivos de salida, uno llamado (`evolucion_entrada.txt`) escribiendo el paso a paso de la pila y otro archivo llamado (`resultado_entrada.txt`) donde se muestra el resultado numérico final.
   - Si ocurre un error: En el archivo de salida (`evolucion_entrada.txt`) se da el paso a paso hasta que suceda el error y el archivo (`resultado_entrada.txt`) detalla cuál es el error de la expresión (aritmética o sintáctica).

Esta implementación asegura que el programa sea robusto y capaz de manejar datos persistentes sin intervención interactiva constante.

5.3. Criterio de aceptación

El autómata acepta una expresión si, tras consumir completamente la cadena de entrada, la pila queda vacía (es decir, solo se ha eliminado el símbolo de fondo Z_0):

$$\delta^*(q_0, w, Z_0) = (q_0, \varepsilon), \quad w \in L(M)$$

Esto implica que:

- Cada operador ha encontrado suficientes operandos en la pila.
- No quedan operandos sin combinar al finalizar la lectura.

5.4. Representación del proceso

De manera general, el proceso de evaluación de una expresión postfija puede representarse como una secuencia de configuraciones del autómata:

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

Ejemplo para la expresión:

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

$$\begin{aligned}(q_0, 3\ 4\ +\ 2\ *, Z_0) &\Rightarrow (q_0, 4\ +\ 2\ *, XZ_0) \Rightarrow (q_0, +\ 2\ *, XXZ_0) \\ &\Rightarrow (q_0, 2\ *, XZ_0) \Rightarrow (q_0, *, XXZ_0) \Rightarrow (q_0, \varepsilon, \varepsilon)\end{aligned}$$

5.5. Resumen operativo

El comportamiento del autómata puede resumirse en las siguientes reglas:

- **Lectura de operando:** se apila un símbolo X.
- **Lectura de operador:** se desapilan dos símbolos X y se apila uno nuevo.
- **Lectura vacía con pila** XZ_0: se acepta la cadena.

Cuadro 1: Resumen equivalente de las transiciones del autómata

#	Entrada	Cima de pila	Acción	Nuevo contenido
1	ε	Z_0	Inicializa pila	Z_0
2	V	Z_0	Apila primer operando	XZ_0
3	V	X	Apila 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 pila	ε

Interpretación: El autómata de pila simula exactamente la lógica de evaluación de una expresión postfija: cada operando incrementa la altura de la pila y cada operador binario reduce su tamaño en uno. El estado final de aceptación representa el momento en que la pila se vacía, confirmando que la expresión fue correctamente balanceada y evaluada.

5.6. Análisis del funcionamiento

Los resultados muestran que el programa reproduce fielmente la semántica del autómata de pila definido teóricamente. Cada transición δ se traduce directamente en una operación de apilado o desapilado, y las condiciones de aceptación coinciden con las definidas formalmente: una única celda en la pila al finalizar la lectura.

6. Resultados

La implementación del autómata de pila permitió evaluar y validar correctamente un conjunto representativo de expresiones aritméticas en notación postfija, incluyendo casos simples y expresiones de mayor longitud. En las pruebas realizadas, el sistema mostró consistencia entre los resultados obtenidos y los valores esperados calculados manualmente o con herramientas de referencia, lo que respalda la corrección del modelo formal y de la lógica de evaluación. Además, la detección de errores como operandos insuficientes, símbolos no válidos o intentos de división entre cero se ejecutó de forma adecuada, generando mensajes claros para el usuario.

6.1. Pruebas de Archivo

Se crearon diversos archivos de texto de prueba para validar la robustez del sistema:

Cuadro 2: Resultados de pruebas con archivos de texto

Contenido Entrada (.txt)	**Contenido Salida (.txt)**	**Estado**
`3 4 + 2 *`	`14`	**Éxito**
`10 2 /`	`5`	**Éxito**
`5 0 /`	`ERROR`	**Incorrecto (Div/0)**
`3 +`	`ERROR`	**Incorrecto (Sintaxis)**
`3 4 5 +`	`ERROR`	**Incorrecto (Pila >1)**

La implementación detectó correctamente la división entre cero y las expresiones mal formadas, generando el mensaje de error estandarizado en el archivo de salida. Esto confirma que la integración del manejo de archivos no alteró la lógica core del autómata, sino que extendió su utilidad.

7. Conclusiones

Se concluye que el uso de un autómata de pila como base para la evaluación de expresiones aritméticas en notación postfija es una estrategia válida y efectiva, tanto desde el punto de vista teórico como práctico. La implementación desarrollada demuestra que los conceptos de gramáticas libres de contexto y autómatas de pila pueden materializarse en un sistema funcional capaz de validar expresiones, calcular resultados y gestionar errores de manera controlada.

Asimismo, la integración del procesamiento de archivos de texto, dotó al sistema de una capacidad semiprofesional, permitiendo al usuario validar expresiones complejas guardadas previamente y obtener reportes de error automatizados. Se verificó que el sistema es capaz de distinguir entre errores sintácticos (estructura RPN incorrecta) y semánticos (indefiniciones matemáticas), garantizando la integridad de los resultados exportados.

8. Referencias bibliográficas

- Autómatas de pila. (2008). *Universidad Nacional del Centro de la Provincia de Buenos Aires*. https://users.exa.unicen.edu.ar/catedras/ccomp1/Apunte4.pdf
- Borbón Alpízar, A. (2006). ¿Cómo evaluar expresiones matemáticas en el computador? *Revista Digital: Matemática, Educación e Internet*, *7*(2), 1–23. https://www.redalyc.org/pdf/6079/607972904002.pdf
- Coordinación de Ciencias Computacionales, INAOE. (s.f.). Gramáticas libres de contexto y lenguajes *Apuntes del Curso Propedéutico: Teoría de Autómatas y Lenguajes Formales* https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso_Propedeutico/Automatas/05_Automatas_GramaticasLibresContextoLenguajes/CAPTUL1.PDF
- Ejemplos de aplicación de pilas. (s. f.). *Universidad de Zaragoza*. https://webdiis.unizar.es/asignaturas/EDA/varios/pilas_colas/ejemplos_aplicacion_pilas.pdf
- Gutiérrez Giraldi, O., & Martínez Moreno, M. (2025). Algoritmo de conversión de una gramática libre de contexto a un autómata de pila. *XIKUA Boletín Científico de la Escuela Superior de Tlahuelilpan*, *13*(25), 31–38. https://doi.org/10.29057/xikua.v13i25.13821
- Juárez Fuentes, J. (s. f.). Autómata de pila. *Universidad Tecnológica de la Mixteca*. https://www.utm.mx/~jjf/tc/3.1%20AUTOMATA_DE_PILA.pdf
- Rincón, L. (s. f.). Autómatas de pila y lenguajes independientes del contexto. *Alfaomega*. https://libroweb.alfaomega.com.mx/book/685/free/ovas_statics/cap9/Automatas%20de%20Pila%20y%20Lenguajes%20independientes%20del%20contexto.%20Rincon,%20Luis.pdf