Desarrollo de contenido

Unidad 1:
Expresiones gramaticales, autómatas finitos y autómatas de pila

Tecnología en Desarrollo de Software

IU Digital de Antioquia
INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

Desarrollo de contenidos
Unidad 1
Expresiones gramaticales, autómatas finitos y autómatas de pila

Nombre de la asignatura:
Autómatas, Gramáticas y Lenguajes

Nombre del Experto:
Bernardo León Hoyos Espinosa

Nombre de la ATP:
Adriana María Martínez Henao

Programa:
Tecnología en Desarrollo de Software

La historia reciente de la humanidad trata de avances, donde las máquinas reemplazan o potencian la fuerza muscular y otros, como la aviación, la radio, la televisión, la electricidad, entre otros que permiten hacer cosas que antes se consideraban ciencia ficción. Hay una máquina que reemplaza o potencia la mente humana en muchos aspectos y es la computadora. Inicialmente fueron artefactos que automatizaban máquinas y resolvían operaciones matemáticas mucho más rápido que los humanos. Eran máquinas con ejes, piñones, palancas, manivelas y demás componentes que desempeñaban funciones propias de la mente humana. Más tarde, con la ayuda de la electricidad y la electrónica, se volvieron más productivas, exactas y confiables.

Después llegaron **los algoritmos de programación**, un elemento intermedio que permite que un ser humano le diga a la máquina qué debe hacer y que la máquina le diga al ser humano cuál es el resultado de su proceso.

Los algoritmos han pasado por muchas etapas, desde una tabla con huecos que programaban el trabajo automático de un telar como lo hicieron los egipcios hace miles de años, pasando por tarjetas perforadas, usadas en los años sesenta y setenta hasta los lenguajes de programación que usamos actualmente. Los lenguajes de programación requieren de una correcta sintaxis y normas gramaticales como cualquier idioma.

Los algoritmos son elementos netamente lógicos que necesitan una forma para establecer esa comunicación entre hombre y máquina. Los autómatas son la base fundamental de los algoritmos. Los lenguajes, son los que permiten traducir las funciones lógicas del algoritmo, en instrucciones dentro de un procesador.

En este curso comprenderás, no solo cómo funciona un computador desde sus fundamentos más básicos, sino también el papel que desempeñan los lenguajes en el proceso.

Objetivos de la asignatura

Objetivos de Aprendizaje

Objetivo general

Desarrollar estructuras de grafos o autómatas, basados en lenguajes y sus normas gramaticales, que permitan construir algoritmos capaces de realizar tareas específicas en un computador.

Objetivos específicos

- Diseñar autómatas finitos que permitan aceptar o rechazar cadenas de texto, con base en el alfabeto dado y la norma gramatical estipulada.
- Agregar el concepto de memoria a un autómata gramatical, incorporando el concepto de pila (Push-Down).
- Diseñar autómatas capaces de simular cualquier algoritmo de computación, manipulando símbolos sobre una cinta de acuerdo con unas reglas establecidas y así, entender los fundamentos de las funciones dentro de un procesador.
- Construir algoritmos que reconozcan cadenas secuenciales que generen respuestas con base, tanto en la secuencia de estados anteriores, como en el estado actual. Los secuenciadores son fundamentales en algoritmos de reconocimiento de contraseñas, en automatización industrial y en el mundo de la robótica.

Pregunta orientadora

Cuando un director le da instrucciones a su colaborador, éste le entiende solamente si ambos hablan el mismo idioma. De lo contrario, se requiere de un intérprete. Lo mismo sucede con los computadores, no entienden inglés ni español, ni ningún idioma de los que hablamos los humanos. Sin embargo, hacen las tareas que les pedimos. Esto significa que existe un intérprete que se pueda entender con la persona y con la máquina.

Para eso están los lenguajes de programación, que nos permiten comunicarnos con el intérprete, pero no con el computador directamente. Te has preguntado: **¿cómo puede un**

computador entender las instrucciones que recibe de una persona y luego comunicar el resultado de su trabajo de forma que la persona lo comprenda?

Mapa de la asignatura

Enlace mapa editable: https://drive.google.com/file/d/1v0-yr6hMR9opHcDFbi7XyxSPFU0PxJbh/view?usp=sharing

Cronograma de actividades de la asignatura

Denominación	Evidencia de Aprendizaje	Estrategia	Forma	Ponderación
Actividad conocimientos previos	Foro.	Foro.	Asincrónica Individual.	0%
Unidad 1	EA1.Expresiones regulares y autómatas finitos. EA2. Autómatas de pila y lenguajes libres de contexto.	Resolución de problemas. Resolución de problemas.	Asincrónica Individual. AsincronicaGrupal (3)	20 % 25 %
Unidad 2	EA3. Máquinas secuenciales.	Resolución de problemas.	AsincrónicaGrupal (3).	25 %
Actividad de cierre	EA4. Máquinas de Turing. Actividad de cierre.	Resolución de problemas.	Asincrónica Grupal (3).	30 %

Unidad 1. Expresiones gramaticales, autómatas finitos y autómatas de pila

Introducción

El desarrollo de *software* implica construir algoritmos que se fundamentan en el análisis lógico-matemático. No obstante, para ello, debe emplear lenguajes de programación, que, como cualquier idioma, se caracterizan por sus propias normas de ortografía, sintaxis, semántica y gramática.

Todos los algoritmos se pueden representar con grafos, o sea, estructuras formadas por nodos y enlaces que permiten modelar cualquier problema. A estos grafos los llamamos *autómatas gramaticales* y son la fundamentación misma de la programación.

Si bien la gramática no es parte de la lógica, sus normas contribuyen a interpretar correctamente un mensaje entre un emisor y su receptor. Esto, teniendo en cuenta que los lenguajes están definidos por un conjunto de símbolos, conocido como alfabeto, y por instrucciones, que tienen un significado específico.

Así pues, en esta primera unidad, examinaremos la construcción de autómatas finitos, basados en lenguajes regulares, con el propósito de reconocer cadenas de texto. En otras palabras, aprenderemos a verificar si los caracteres forman parte de un alfabeto dado y si las instrucciones tienen algún significado en el lenguaje que se esté usando.

Entender esto será clave para nuestro aprendizaje pues hay que tener claro que una cadena puede llegar a ser rechazada, por ejemplo, porque un carácter no fue reconocido (una ñ, una tilde); además, hay cadenas escritas que no tienen ningún sentido en el idioma en el que estemos hablando, por ejemplo, decir libro, no significada nada si hablamos en inglés.

Finalizaremos la Unidad 1 con el tema 2, en el cual aprenderemos a construir autómatas de pila y a reconocer cadenas más complejas que no pueden ser reconocidas por los autómatas finitos, que son el fundamento de los arreglos o vectores en la programación de *software.*

Objetivos de aprendizaje de la Unidad 1

1. Diseñar autómatas finitos que permitan aceptar o rechazar cadenas de texto, con base en el alfabeto dado y la norma gramatical estipulada.

2. Agregar el concepto de memoria a un autómata gramatical, incorporando el concepto de pila (*Push-Down*) para construir arreglos o vectores.

Cronograma de actividades Unidad 1			
Actividad de aprendizaje*	**Evidencia de aprendizaje****	**Semana*****	**Ponderación**
AA. Actividad de conocimientos previos	EA. Foro	Semana 1	0%
AA1. Lenguajes regulares y autómatas finitos	EA1. Fase 1. Expresiones regulares y autómatas finitos	Semana 2	20%
AA2. Autómatas de pila (*Push-Down*)	EA2. Fase 2. Autómatas de pila y lenguajes libres de contexto	Semana 4	25%
Total			45%

Unidad 1. Expresiones gramaticales, autómatas finitos y autómatas de pila

Si tuviéramos que contarte en dos puntos qué aprenderemos en esta unidad, te lo resumiríamos así:

1. Aprenderemos que los lenguajes de programación tienen reglas gramaticales.

2. Conoceremos cómo hacer algoritmos con grafos llamados *autómatas*, que serán los encargados de reconocer las cadenas de texto, aceptarlas cuando cumplen con todas las normas gramaticales del lenguaje o rechazarlas cuando las infrinjan.

Desarrollo temático de la unidad

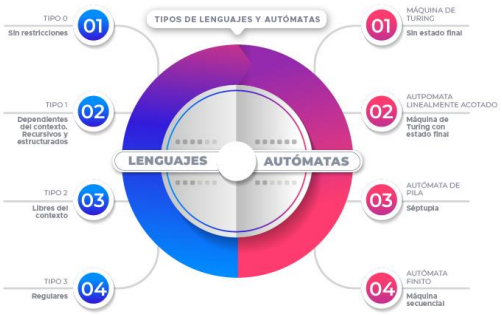
Expresiones gramaticales y autómatas finitos

Imagina que dos computadores tienen que comunicarse para definir una tarea. Imagina ahora que tú debes hacerlo con el computador. Ya tienes claro que debes hacerlo con el lenguaje de programación, ¿cierto? ¡Pues bien!, no puedes olvidar que **los lenguajes de programación tienen reglas de sintaxis y semántica más estrictas que las nuestras.** El ser humano es más flexible al comunicarse, pues utiliza modismos, regionalismos, neologismos, abreviaturas y otras variaciones que, de manera general, no distorsionan el mensaje; los computadores, por el contrario, son más rígidos y solo aceptan cadenas con una gramática previamente definida. Por ello, es importante reconocer las características de su lenguaje, con el fin de interpretar correctamente las instrucciones que enviamos y recibimos.

1. Tipos de lenguajes:

Según la jerarquía de Chomsky, los lenguajes de programación se dividen en cuatro, según sus características. El nivel más básico es el tipo 3 y el más completo es el tipo 0.

Figura 1. Tipos de lenguajes y autómatas



Birchenall & Müller (2014), definen y explican un poco los cuatro tipos de lenguaje:

- *Tipo 3. Lenguajes regulares:*

Generan cadenas básicas simples o unidas con operadores lógicos como la intersección, la unión, la negación, o la concatenación, una cantidad finita de veces. Pueden ser reconocidos por autómatas finitos deterministas. Por ejemplo, pueden aceptar cadenas como a^nb^r, que podría ser a^3b^2 o "aaabb". Se debe especificar el valor de "n" y de "r" explícitamente. Además de reconocer cadenas de texto, los autómatas finitos se usan para generar máquinas secuenciales.

- *Tipo 2. Lenguajes libres de contexto:*

Requieren una unidad de memoria para ser reconocidos, aunque esta puede usarse solamente una vez. Por ejemplo, a^nb^n, que podría ser a^2b^2 o "aabb". Aquí únicamente se especifica el valor de "n", por lo tanto, después de reconocer la subcadena a^n, se debe almacenar en una memoria la cantidad de veces que leyó la letra "a" para verificar si corresponde con la cantidad de veces que lee "b". Veamos un par de ejemplos: $a^nb^{(n+2)}$, $a^nb^{(2n)}$.

Es importante recordar que los lenguajes libres de contexto pueden ser leídos por los autómatas de pila "*Push-Down*".

- *Tipo 1. Lenguajes dependientes del contexto*:

Pueden ser leídos por un autómata acotado lineal, es decir, una máquina de Turing no determinista con una cantidad finita de posiciones. En lugar de una pila, tienen una cinta que se puede desplazar a ambos lados o quedarse inmóvil en cada paso, según el algoritmo que está corriendo.
Este autómata puede usarse como una unidad de memorias múltiples y reutilizables. El alfabeto de entrada incluye dos símbolos especiales # y $, que son las marcas fin de cinta, izquierda y derecha, respectivamente. La cabeza del autómata no puede desplazarse fuera de los límites izquierdo y derecho de la cinta, y no puede imprimir otro símbolo sobre # y $. Puede reconocer cadenas como $a^nb^{(n+1)}c^{(n+2)}d^{(n+3)}$ (Cueva *et al.*, 2003).

- *Tipo 0. Lenguajes sin restricción:*

Puede ser leído por una máquina de Turing; es decir, el lenguaje aceptado o reconocido por una máquina de Turing, es el conjunto de palabras que hace que esta se detenga al alcanzar un estado final.
La cinta desplazable que usa la máquina de Turing puede usarse como una unidad de memorias múltiples y reutilizables. Permite cambiar de posición cualquier símbolo de la cadena, lo cual se usa para ejecutar todo tipo de algoritmos lógicos y matemáticos. A diferencia del tipo 1, la máquina de Turing puede cambiar los símbolos de los extremos y puede extenderse fuera de ellos (Cueva *et al.*, 2003).
Aunque la máquina de Turing fue creada para resolver una operación concreta, el mismo Alan Turing aseguró que puede resolver todos los problemas matemáticos que pueden expresarse mediante un algoritmo (Puig, 2012).

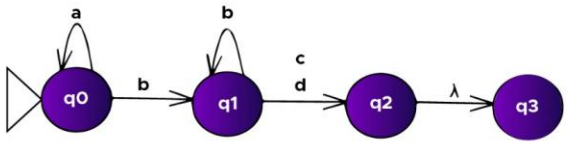
1.1. Autómata finito:

Aunque ya hemos hablado del término, aún no hemos profundizado en él. ¿Qué deberíamos saber?

Primero, es importante conocer que la misión del autómata finito es recibir una cadena construida con los símbolos de un alfabeto y determinar si dicha cadena pertenece al lenguaje que ellos reconocen.
Segundo, los autómatas finitos reconocen las cadenas de lenguajes regulares, también denominadas *cadenas tipo 3*. Además, se definen en función de cinco puntos conocidos como la quíntupla del autómata, los cuales se presentan a continuación y se explican:

Figura 2. Autómata finito 1.

Comentado [DAUM1]: La imagen requiere rediseño.



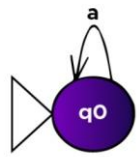
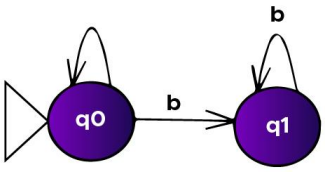
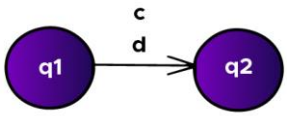
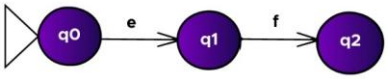
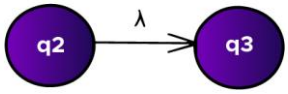
Q: conjunto finito de estados.
En el autómata finito 1, Q = q0, q1, q2 y q3
Σ: alfabeto, siempre finito y nunca vacío.
En el autómata finito 1, Σ = a, b, c, d, la letra λ indica que se puede pasar de q2 a q3 sin necesidad de leer algún carácter. Es decir, es una transición vacía.
S: estado inicial. S ∈ Q. Siempre debe existir un solo estado inicial.
En el autómata finito 1, es S = q0.
F: estado final o estados finales. F ⊆ Q.
En el autómata finito 1, solamente hay un estado final: F = q3.
Δ: transición. En ella, el autómata lee un carácter del alfabeto, ya sea para permanecer en el mismo estado o para pasar a otro.

1.2. Tipos de transiciones:

Existen unos eventos que deben ocurrir o unas condiciones que deben cumplirse para pasar de un estado a otro. Esto es lo que conocemos como **transiciones.** En los autómatas, las transiciones ocurren leyendo un símbolo de la cadena de texto, si lee el símbolo correcto, se pasa al siguiente estado. De lo contrario, rechaza la cadena por no cumplir con la gramática de la expresión.

Estas son las transiciones más comunes:

Tabla 1. Tipos de transiciones.

a* Significa que la letra "a" puede omitirse, o leerse una vez o varias veces. En el autómata finito 1 es la transición de q0 a q0 (ver *Figura 2*).	
b⁺ Significa que la letra "b" debe leerse al menos una vez. En el autómata finito 1 es la transición de q0 a q1, y luego de q1 a q1 cualquier cantidad de veces (ver *Figura 2*).	
c | d Se lee "c" o "d". Leyendo una letra "c" o una letra "d" pasa de q1 a q2.	
Ef Se lee "e" y "f". Antes de pasar de q0 a q2, debe leer ambas letras en el orden en que están escritas. Cada letra debe estar definida en el alfabeto. Por ejemplo: **Σ** = e, c, h, o ≠ **Σ** = e, ch, o	
λ Transición vacía. Significa que puede pasar a otro estado sin necesidad de leer algún carácter.	
Expresiones regulares que acepta el autómata finito 1: bc, bd, abc, abd, aaacb, aaabd, bbbc, abbbc, abbbd, aaabbbc, aaabbbd,etc.	**Expresiones regulares que rechaza el autómata finito 1:** ac, ad, bac, cba, dba, aabb, aadd, aaacbbb, aaadbbb, abccc, bdd, etc.

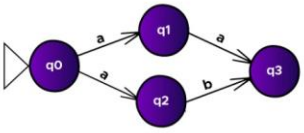
1.4. Autómatas finitos deterministas (AFD) y no deterministas (AFND):

Según las transiciones y el número de estados al que se direccionan los autómatas, tendremos dos tipos:

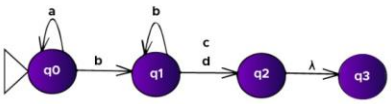
- Autómatas finitos deterministas.
- Autómatas finitos no deterministas.

¿Cuál es su diferencia? Un autómata finito **es determinista** si, para cada transición, hay solamente un estado siguiente. Por ejemplo, el autómata finito 1 de la *Figura 2* es un AFD. Un autómata finito es **no determinista** si, para alguna de las transiciones, hay más de un estado siguiente, es decir, la decisión es indeterminada.

Tabla 2. Ejemplos de autómatas no deterministas.



Con la letra "a", se puede salir de q0 para q1 o para q2.

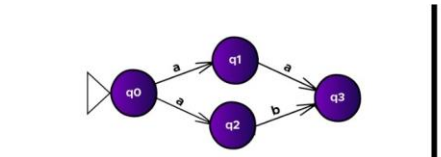
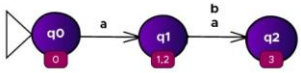
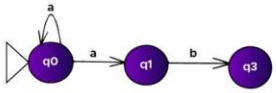
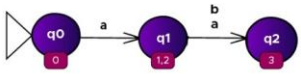
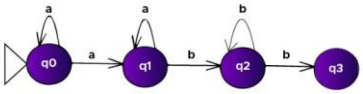
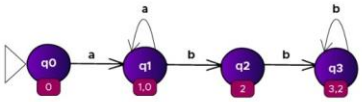


Con la letra "a" puede permanecer en el estado q0 o puede salir hacia q1.

Convertir un AFND en AFD:

Pero hay algo que hay que tener claro: **un autómata finito no determinista se puede convertir en determinista**. Para ello, ambos deben aceptar las mismas cadenas. No obstante, es probable que cambie la cantidad de nodos o de transiciones.

Tabla 3. Convertir AFND en AFD.

AFND	AFD convertido con el simulador JFLAP
	
	
	

¡Ten en cuenta!

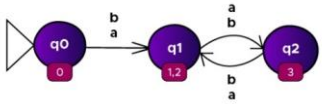
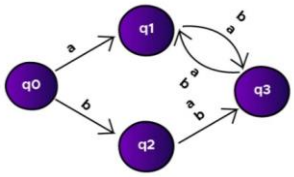
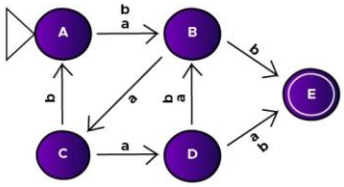
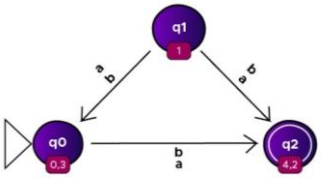
A lo largo del curso usaremos el simulador de código abierto JFLAP, el cual contiene herramientas gráficas de mucha utilidad para aprender conceptos básicos de lenguajes formales y teoría de autómatas.

Si deseas descargarlo y conocer más sobre su funcionamiento, ¡hazlo aquí!

- *Minimizar un autómata finito:*

Un autómata finito determinista con transiciones redundantes se puede minimizar de modo que acepte las mismas cadenas que el original:

Tabla 4. Minimizar autómatas.

Autómata finito	Autómata finito minimizado con el simulador JFLAP
	
La expresión regular es: [(a|b)(a|b)]+	
	
El autómata original tiene dos estados finales. La expresión regular es: (a|b)(a|b) [(a|b)(a|b)(a|b)]*	

1.5. Entrenamiento con autómatas finitos

Hemos definido algunos elementos básicos, ¿qué te parece si ahora tratamos de aplicarlos en un ejercicio? ¡Presta mucha atención al enunciado y al desarrollo para que entiendas mejor como es el funcionamiento y cuál es la aplicación de los autómatas:

1. Dada la expresión gramatical regular: ***01*(001*)****, resolver los siguientes puntos:

a. Construir el autómata finito que pueda reconocer la expresión gramatical.
b. Construir la quíntupla del autómata.
c. Construir la tabla de transiciones.

d. Comprobar mínimo 5 cadenas que cumplan la gramática y sean aceptadas por el autómata y 5 cadenas que no cumplan la gramática y sean rechazadas por el autómata.

Solución:

Para comenzar, es preciso entender cada símbolo de la expresión. Para este ejemplo hemos usado el simulador JFLAP (JFLAP VERSIÓN 7.1):

- El primer asterisco indica que el primer "1" se puede omitir o repetir cuantas veces se quiera, pero solo el "1".
- El segundo asterisco indica que el "1" dentro del paréntesis también se puede omitir o repetir cuantas veces se quiera, pero solo el "1".
- El tercer asterisco indica que todo lo que hay dentro del paréntesis se puede omitir o repetir cuantas veces se quiera, o sea, "*001**".

A continuación, escribiremos algunas cadenas que cumplan con la gramática establecida para que el autómata las acepte. Veamos:

- 0
- 01
- 000
- 011111
- 0000000
- 0100
- 010000
- 0001
- 0001001001
- 01001110011100111
- 01111001001001001
- 01001
- 0001001001
- 0001110011100111

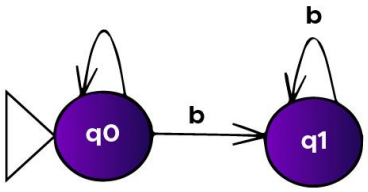
En tercer lugar, escribiremos algunas cadenas que no cumplan la gramática para que el autómata las rechace. Así:

- 1
- 0101
- 10110
- 01001110
- 101010
- 011101110111
- 001
- 0001110
- 01001001010
- 11

Ahora, desarrollemos los puntos solicitados:

a. Se construye el autómata correspondiente a la expresión regular. Como se muestra en la siguiente figura:

Figura 3. Autómata finito, 01*(001*)*



b. Para este autómata, la quíntupla correspondería a:

Q = {q0, q1}
Σ = {0, 1}
S = q0
F = {q1}
Δ = {(q0, 0, q1), (q1, 0 q0), (q1, 1, q1)}

c. Las transiciones también se pueden expresar en una tabla, de la siguiente forma:

Tabla 5. Transición autómata finito, 01*(001*)*

Transiciones	q0	q1
q0	-	0
q1	0	1

d. Después de tener el autómata finito, podemos verificar que se acepten las cadenas que cumplen con la gramática.

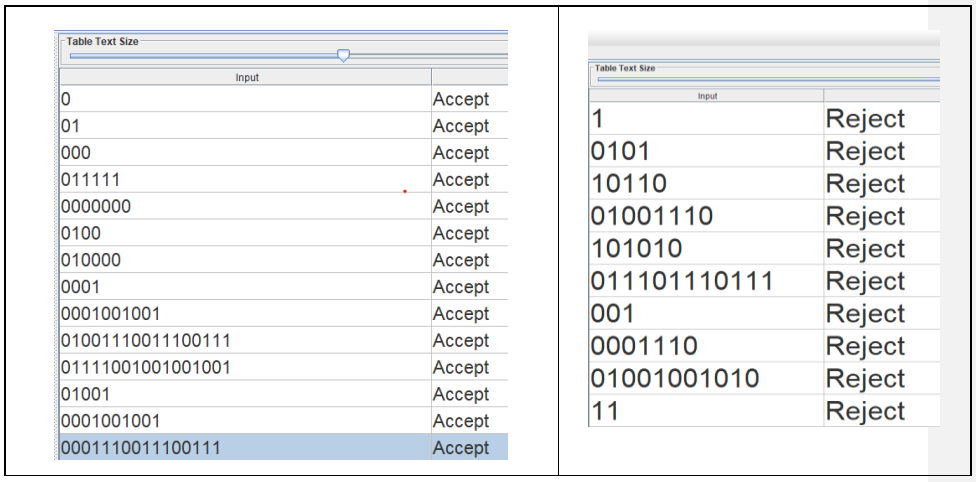
Usando la opción de autómatas finitos, que se encuentra en la ubicación que muestra la siguiente figura:

Figura 4. Ruta para simulación múltiple.

En la tabla se puede apreciar el resultado de la simulación:

Tabla 6. Resultado de la simulación múltiple.

El resultado de las cadenas que debe aceptar:	Las cadenas que debe rechazar:



Input	
0	Accept
01	Accept
000	Accept
011111	Accept
0000000	Accept
0100	Accept
010000	Accept
0001	Accept
0001001001	Accept
01001110011100111	Accept
01111001001001001	Accept
01001	Accept
0001001001	Accept
0001110011100111	Accept

Input	
1	Reject
0101	Reject
10110	Reject
01001110	Reject
101010	Reject
011101110111	Reject
001	Reject
0001110	Reject
01001001010	Reject
11	Reject

1.6 Autómatas finitos con operadores lógicos:

Tanto la teoría de conjuntos como las tablas de verdad del álgebra booleana son necesarias para resolver autómatas con más de una condición. En estos casos aparecen elementos como la teoría de conjuntos, que enseña la base de los operadores lógicos. En la tabla se comparten los operadores lógicos más empleados:

Tabla 6. Operadores lógicos más empleados.

En español	Conjuntos	Álgebra	Programación
Intersección / Y	∩	∧	&&
Unión / O	U	v	||
Negación	A’	≠ ~	!
Comparadores	≤, ≥, <, >		

En ocasiones, los autómatas deben reconocer cadenas que cumplan dos o más condiciones conectadas con operadores lógicos. Además, estos cambian radicalmente dependiendo del operador lógico que los enlaza.

En el próximo tema estudiaremos el **autómata de pila,** cuya principal característica es su unidad de memoria, que permite llevar la cuenta de ciertas operaciones, algo que no es posible con los autómatas finitos.

2. Autómatas de pila y lenguajes libres de contexto

Los lenguajes independientes del contexto o libres del contexto (LC), sobrepasan las limitaciones de memoria de los lenguajes regulares, pues almacenan una cierta cantidad de información, siempre que sea accedida en forma de pila.

Según Navarro (2017), son el mecanismo formal para expresar la gramática de los lenguajes de programación o semiestructurados. Por ejemplo, se consideran gramáticas libres del contexto: la notación "Backus-Naur form", muy popular para describir reglas sintácticas, y los DTD (*Document Type Definition*) que indican el formato propio de los documentos XML.

A esto añade que:

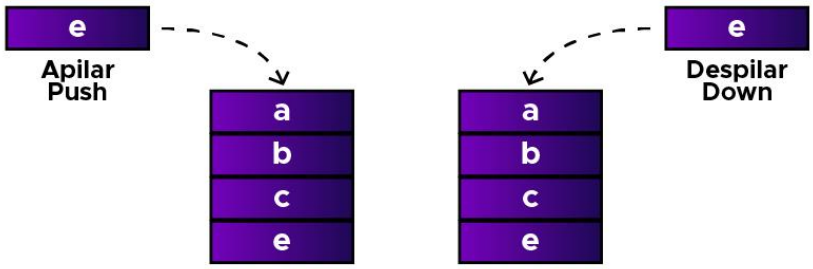
> El estudio de este tipo de lenguajes deriva la construcción semiautomática de "*parsers*" (reconocedores) eficientes, los cuales son esenciales en la construcción de compiladores e intérpretes, así como para procesar textos semiestructurados. Una herramienta conocida para esta construcción semiautomática es lex/yacc en C/Unix, y sus distintas versiones, para otros ambientes. Estas herramientas reciben esencialmente una especificación de un lenguaje LC y producen un programa que escanea tal lenguaje.

Los lenguajes independientes del contexto se emplean en diferentes campos y con diversos propósitos por su utilidad en cuanto a cadenas, por ejemplo, en biología computacional y bioinformática sirven para trabajar con secuencias de ADN o proteínas.

2.1. Autómatas de pila:

Son conocidos como PDA (*Push Down Automation*) y tienen un poder adicional de operación debido a sus unidades de memoria, la cual se conoce como pila, porque permite agregar datos en el tope de la lista y extraerlos únicamente desde allí. Veamos un ejemplo para entender cómo se apilan y extraen los datos:

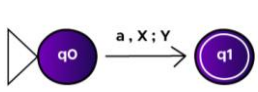
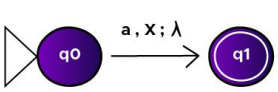
Figura 5. Autómata de pila.



Esta forma de agregar o quitar datos del tope de una lista obedece a un sistema denominado LIFO (*LAST IN, FIRST OUT*), que significa: el último que entra es el primero que sale. Esto da origen a lo que se conoce como "arreglo o vector en programación".
Además, estos autómatas reconocen cadenas de lenguajes tipo 2, es decir, lenguajes libres de contexto. Un autómata de pila, a diferencia de los finitos, tiene tres símbolos en cada transición:

- El primero es el símbolo que lee en la cadena de texto, igual que en un autómata finito.
- Después de una coma, está el símbolo que encuentra en el tope de la pila.
- Luego de un punto y coma, está el símbolo que deja en el tope de la pila.

En la siguiente tabla vemos la representación de los tres escenarios posibles:
Tabla 7. Tope de pila.

Cambiar dato en el tope de la pila	Quitar dato del tope de la pila	Agregar dato al tope de la pila
		
Para pasar de q0 a q1, debe leer una "a" en la cadena de texto y encontrar una "X" en el tope de la pila. Entonces, cambia la "X" por la "Y".	Para pasar de q0 a q1, debe leer una "a" en la cadena de texto y encontrar una "X" en el tope de la pila. Entonces, quita la "X".	Para pasar de q0 a q1, debe leer una "a" en la cadena de texto y adicionar una "Y" sin importar qué hay en el tope de la pila.

	El símbolo λ significa: no adicionar. Así, la pila queda con un dato menos.	La pila queda con un dato más.

Un autómata de pila se define en función de los siguientes siete puntos, conocidos como la ***séptupla del autómata***:

Q: conjunto finito de estados. En los ejemplos de la *Tabla 7*, puedes observar dos estados: Q = q0 y q1.

Σ: alfabeto de la cadena de texto, siempre debe ser finito y no puede ser vacío. En los ejemplos de la *Tabla 7* encontrarás que Σ = a. También acepta transiciones vacías, como en los autómatas finitos.

S: estado inicial, S ∈ Q; siempre debe existir un solo estado inicial. En los ejemplos de la *Tabla 7:* S = q0.

F: estado final o estados finales, F ⊆ Q. En los ejemplos de la *Tabla 7*, solamente hay un estado final: F = q1.

T: (Tau) alfabeto de la pila. Siempre debe ser finito y no puede ser vacío.

Δ: transición.; en ella, el autómata lee un carácter del alfabeto y el tope de la pila, y asigna el nuevo valor del tope de la pila, ya sea para permanecer en el mismo estado o para pasar a otro.

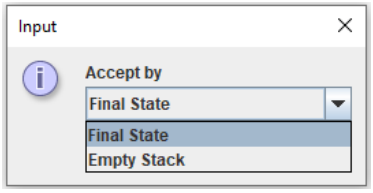
Z_0: indica que la pila está vacía. Es el símbolo inicial de la pila.

Los autómatas de pila terminan exitosamente el proceso cuando se cumplen estas tres condiciones:

- Lee toda la cadena.
- Llega a un estado final.
- La pila queda vacía.

Importante: hay un aspecto importante a considerar del simulador JFLAP (versión 7.1). En algunos casos, un algoritmo debe detenerse cuando termina de leer la cadena de texto sin llegar a un estado final determinado. Por ello, el simulador ofrece la opción de detenerse de ambas formas.

Figura 6. Opciones del autómata de pila.



En ambos casos, la pila debe quedar vacía y la cadena debe llegar al final.

1.2. Entrenamiento para expresiones gramaticales libres de contexto:

A continuación, observemos un ejercicio que lleva a la práctica los conceptos sobre los autómatas de pila, las transiciones y la séptupla que los caracteriza:

1. *A partir de la expresión libre de contexto: 0^n1^n, resolver los siguientes puntos:*

   b. Construir un autómata de pila que pueda reconocer la expresión gramatical.
   c. Construir la séptupla (siete parámetros) del autómata.
   d. Construir la tabla de transiciones.
   e. Comprobar mínimo 5 cadenas que cumplan la gramática y sean aceptadas por el autómata; y 5 cadenas que no cumplan la gramática y sean rechazadas por el autómata.

Solución:

A pesar de lo simple de la expresión 0^n1^n, no puede ser reconocida por un autómata finito, pues, aunque podría leer la primera parte (0^n), luego no sabría cuántas veces debe leer 1^n para aceptarla. Por eso, se requiere el autómata de pila.

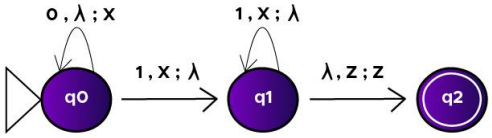
Las siguientes son algunas cadenas que usaremos para probar el autómata de pila:

Debe aceptar	**Debe rechazar**
• 01	• 0
• 0011	• 1
• 000111	• 001
• 00001111	• 0111
• 0000011111	• 1100

Ahora, desarrollemos los puntos solicitados:

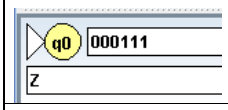
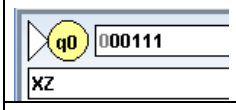
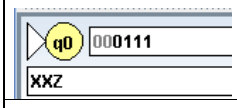
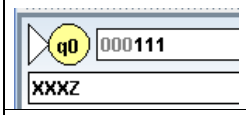
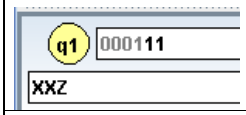
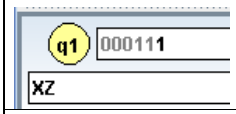
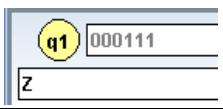
a. Veamos en la siguiente figura el autómata de pila que resulta de la expresión dada:

Figura 7. El autómata correspondiente a la expresión regular 0^n1^n



En las siguientes tablas encontrarás la explicación de la representación del autómata, el paso a paso de cómo llevar la expresión al simulador, los resultados que arrojará y algunas acciones que te serán de ayuda para interpretar su gramática.

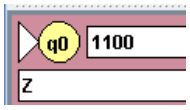
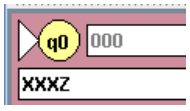
Tabla 8. Paso a paso de reconocimiento con la cadena “000111”.

	Inicialmente, se encuentra en q0, además, la pila está vacía, como indica la “Z”.
	Aquí, el simulador lee el primer cero y agrega un “X” a la pila. El estado sigue siendo q0.
	Lee el segundo cero y agrega otra “X” a la pila. Continúa en el estado q0.
	Lee el tercer cero y agrega la tercera “X” a la pila. Nota que, por cada cero que ha leído, ha agregado una “X” a la pila. Aún se encuentra en q0.
	Debido a que recién lee el primer uno (1) en la cadena, quita una “X” del tope de la pila. Ahora, el estado cambia a q1.
	Lee el segundo uno de la cadena de texto y quita otra “X” de la pila; continúa en q1.
	Lee el tercer uno de la cadena de texto y quita la tercera “X” de la pila. Observa que la pila vuelve a su estado “Z”. Aún está en el estado q1.

	El último paso verifica que efectivamente la pila esté vacía. Para pasar de q1 a q2, que es el estado final, debe leer λ en la cadena de texto "Z" en la pila y dejar la misma "Z". Aunque no lee una parte de la cadena ni realiza cambios en la pila, este paso es importante para verificar que efectivamente esté vacía. El simulador colorea la ventana de un verde más intenso para indicar que la cadena fue aceptada.

A continuación, observa en la siguiente tabla las cadenas que son rechazadas por el simulador y qué sucede con la pila:

Tabla 9. Cadenas rechazadas por el simulador

El simulador JFLAP indica el rechazo coloreando la ventana de un color rojizo:	
	Esta cadena la rechaza desde el primer paso, pues para pasar de q0 a q1, leyendo un uno, debe desmontar una "X" de la pila, pero la pila está vacía.
	Esta cadena la rechaza en el cuarto paso. Después de leer tres ceros, ha agregado tres "X" a la pila. Como no hay unos (1) para leer en la cadena, no puede quitar las "X" de la pila. La rechaza porque la pila no quedó vacía.
	Si hay más ceros que unos, la pila no queda vacía. Agregó tres "X" cuando leyó los tres ceros, pero solamente desmontó dos "X" y se terminó la cadena de texto.
	Si hay menos ceros que unos, la cadena no queda vacía. A pesar de quedar vacía la pila, la cadena no lo está. Para comprobar que no hay más caracteres qué leer en la cadena de texto, el autómata debe leer "λ", que indica vacío o ausencia de caracteres.

b. La siguiente es la séptupla de este autómata:

Q = {q0, q1, q2}
Σ = {0, 1}

S = q0
F = {q2}
Δ = {(q0, 0, λ, X, q0), (q0, 1, X, λ, q1), (q1, 1, X, λ, q1), (q1, λ, Z, Z, q2)}
T = {X}
Z_0 = Z

c. Las transiciones se pueden expresar en una tabla, así:

Tabla 10. Transición de la expresión 0^n1^n

Transiciones	q0	q1	q2
q0	0, λ, X	1, X, λ	-
q1	-	1, X, λ	λ, Z, Z
q2	-	-	-

d. El cuarto punto del ejercicio nos pide comprobar la gramática de las cadenas, y escribirlas para que sean aceptadas o rechazadas. Por ello, después de tener el autómata de pila, debemos verificar que se acepten las cadenas que cumplen con la gramática.

Recuerda que, para este ejemplo, emplearemos el simulador JFLAP, esta vez utilizando la opción "*Push Down*":

Figura 8. Ruta del simulador múltiple *Push Down.*

Observa en la siguiente tabla los resultados:

Tabla 11. Reconocimiento de cadenas.

Cadenas que debe aceptar	Las cadenas que debe rechazar

Input	Result
01	Accept
0011	Accept
000111	Accept
00001111	Accept
0000011111	Accept

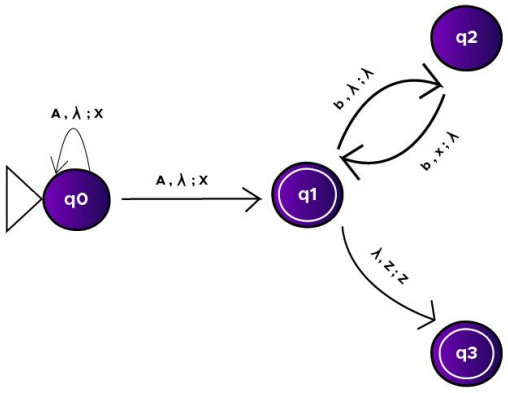
Input	Result
0	Reject
1	Reject
001	Reject
0111	Reject
1100	Reject

Exploremos unos cuantos ejercicios más para una mejor comprensión de los temas que hemos explorado hasta ahora. Presta atención a su desarrollo y a los comentarios que allí se incluyen.

2. Considera la siguiente expresión libre de contexto $a^n b^{2n}$ *para* $n > 0$.

a. A partir de la expresión gramatical dada, construyamos el autómata de pila que la pueda reconocer, así:

Figura 9. Autómata de la expresión $a^n b^{2n}$



Analicemos a continuación el autómata que construimos:

- Si n > 0, debe leer al menos una “a” para pasar de q0 a q1.
- Por cada “a” que lea, agrega una “X” a la pila.
- De q1 a q2 lee una “b”, pero no hace cambios en la pila.
- Al regresar de q2 a q1, lee otra “b” y desmonta una “X” de la pila.”. Este paso obliga a desmontar una “X” por cada dos “b” que lea. Es decir, la pila quedará vacía cuando la cantidad de “b” leída sea el doble que la cantidad de “a”.

- La transición de q1 a q3, "λ, Z, Z", asegura que la cadena de texto haya finalizado y que la pila quede vacía.

b. Ahora, veamos la séptupla del autómata:

Q = {q0, q1, q2, q3}
Σ = {a, b}
S = q0
F = {q3}
Δ = {(q0, a, X, λ, q0), (q0, a, X, λ, q1), (q1, b, X, λ, q2), (q2, b, λ, λ, q1)}, {q1, λ, Z, Z, q3}
T = {X}
Z_0 = Z

c. Luego, elaboramos la tabla de transiciones:

Tabla 12. Transición de la expresión a^nb^{2n}

Transiciones	q0	q1	q2	q3
q0	a, X, λ,	a, X, λ	-	-
q1	-	-	b, X, λ	λ, Z, Z
q2	-	b, λ, λ	-	-
q3	-	-	-	-

d. De nuevo, como en el ejercicio anterior, comprobaremos al menos 5 cadenas que sean aceptadas por el autómata y 5 que sean rechazadas.

Tabla 13. Simulación múltiple de la cadena a^nb^{2n}

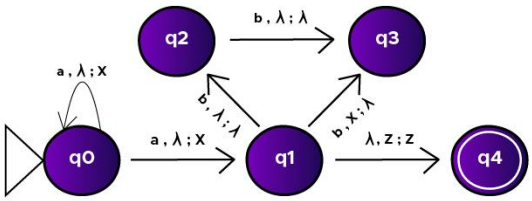
Input	Result
abb	Accept
aabbbb	Accept
aaabbbbbb	Accept
aaaabbbbbbbb	Accept
aaaaabbbbbbbbbb	Accept

bba	Reject
bbaaaa	Reject
abbabb	Reject
abab	Reject
aaaaa	Reject

1. *A partir de la expresión libre de contexto* a^nb^{3n}*, resolver los siguientes puntos:*

a. Primero, debemos construir un autómata de pila que pueda reconocer la expresión gramatical:

Figura 10. Autómata de la expresión a^nb^{3n}



El autómata lee la letra “a” n veces y por cada “a” monta una “X” en la pila. El recorrido por q1-q2-q3-q1, lee tres veces la letra “b” pero solamente quita una “X” en cada ciclo. Cuando la pila esté vacía, habrá leído el triple de “b” que de “a”.

b. Luego, analizamos la séptupla del autómata:

Q = {q0, q1, q2, q3, q4}
Σ = {a, b}
S = q0
F = {q4}
Δ = para autómatas con muchas transiciones es preferible recurrir a la tabla.
T = {X}
Z_0 = Z

c. A continuación, construimos la tabla de transiciones.

Tabla 14. Transiciones a^nb^{3n}

Transiciones	**q0**	**q1**	**q2**	**q3**	**q4**
q0	a, λ, X	a, λ, X	-	-	-
q1	-	-	b, λ, λ	-	λ, Z, Z
q2	-	-	-	b, λ, λ	-
q3	-	b, X, λ	-	-	-
q4	-	-	-	-	-

d. Finalmente, comprobamos mínimo 5 cadenas que sean aceptadas por el autómata y 5 que sean rechazadas por la cadena a^nb^{3n}.

Figura 11. Pantallazo del simulador con la cadena a^nb^{3n}

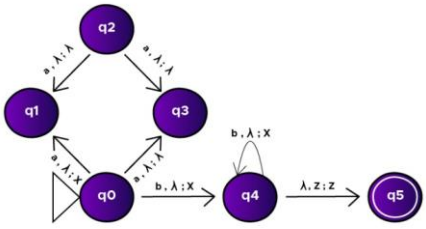
Input	Result
abbb	Accept
aabbbbbb	Accept
aaabbbbbbbbb	Accept
aaaabbbbbbbbbbb	Accept
aaaaabbbbbbbbbbbbbb	Accept
bbba	Reject
aabb	Reject
abbbb	Reject
bbbbbbaa	Reject
abab	Reject

2. *A partir de la expresión libre de contexto: **$a^{4n}b^{n}$**, resolver el autómata correspondiente y comprobar expresiones que cumplan y que no cumplan.*

Para este ejercicio podemos plantear dos soluciones, presta atención a cada una de ellas y luego define cuál puede ser la mejor.

Solución 1: por cada cuatro "a" que lee, monta una "X" a la pila. Por cada "b" que lee, quita una "X" de la pila. Una vez lea la primera "b", no puede volver a leer otra "a".

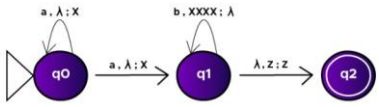
Figura 12. Solución 1 con la cadena $a^{4n}b^{n}$



Input	Result
aaaab	Accept
aaaaaaaabb	Accept
aaaaaaaaaaaabbb	Accept
aabb	Reject
bbaaaaaaaa	Reject
aaaabaaaab	Reject

Solución 2: existe otra solución, la cual, aunque funciona como autómata en la simulación, podría presentar problemas a la hora de escribir un código, pues es diferente tener un arreglo con cuatro posiciones y en cada una de ellas la variable "X". Es decir, no es lo mismo P= ["X", "X", "X", "X"] que una variable P= "XXXX".

Figura 13. Solución 2 con la cadena $a^{4n}b^{n}$



Input	Result
aaaab	Accept
aaaaaaaabb	Accept
aaaaaaaaaaaabbb	Accept
aabb	Reject
bbaaaaaaaa	Reject
aaaabaaaab	Reject

Al comparar ambas soluciones, la primera opción parece ser la mejor, porque el alfabeto de su expresión gramatical es Σ = {a, b} y el alfabeto de la pila T = {X}. Mientras que la *Solución 2* necesita que el alfabeto T = {X, XXXX}.

Ten en cuenta que es preferible que la pila sea un arreglo al cual se le van agregando o quitando elementos y no una palabra a la cual se le van concatenando o eliminando letras.

Continuemos el recorrido por los temas de la unidad descubriendo otro concepto importante: los palíndromos.

2. Palíndromos

Un palíndromo es una palabra o frase que se lee igual de derecha a izquierda y de izquierda a derecha. Veamos algunos ejemplos:

Palabras palíndromas	Frases palíndromas
OJO OSO SERES ARENERA SOMETEMOS SOMOS ANILINA	• A TI NO, BONITA • AJÍ TRAGA LA LAGARTIJA • ANITA LAVA LA TINA • LUZ AZUL • YO HAGO YOGA HOY • AMAN A PANAMÁ • ALLÍ VES SEVILLA

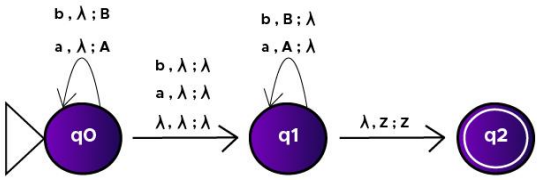
El uso de palíndromos se debe a que los autómatas de pila guardan la cantidad de veces que lee cada letra en la primera mitad de la cadena con el objetivo de reconocer la segunda mitad.

Ejemplos del uso de palíndromos:

1. Dado un alfabeto de lenguaje Σ= {a, b} y un alfabeto de pila T= {A, B}, resolver un autómata de pila que reconozca las cadenas: aba, aabbaa, abababbbababa, aaabbaaa, aaaabbbabbbaaaa.

Solución:

Figura 14. Solución de un palíndromo con Σ={a, b}



aba	Accept
babbab	Accept
abababbbababa	Accept
aaabbaaa	Accept
baaabbbabbbaaab	Accept
aab	Reject
aaabb	Reject
ababab	Reject
aabaab	Reject
bbbbbbba	Reject

Después de leer la primera mitad de la cadena, debe desmontar la pila en el orden inverso que se formó.

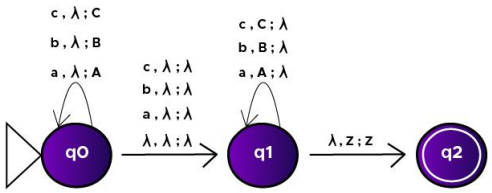
La transición de q0 a q1, marca la mitad del proceso. Si la cantidad de letras del palíndromo es par, en la mitad lee λ, λ, λ. Pero si la cantidad de letras es impar, en la transición de q0 a q1, lee la letra que hay en toda la mitad de la palabra.

Observa que, al hacer la simulación (*Figura 14*), las palabras que no son palíndromos fueron rechazadas.

2. Dado un alfabeto de lenguaje Σ= {a, b, c} y un alfabeto de pila T= {A, B, C}, resolver un autómata de pila que reconozca las cadenas: acbca, aabccbaa, abcabcabccbacbacba, aaabcccbaaa, aaccbbbcbbbccaa.

Solución:

Figura 15. Solución de un palíndromo con Σ={a, b, c}



acbca,	Reject
aabccbaa	Accept
abcabcabccbacbacba	Accept
aaabcccbaaa	Accept
aaccbbbcbbbccaa	Accept
aabcc	Reject
cccaab	Reject
abcabcabc	Reject
aabaccab	Reject
bbbbbbbaaaacc	Reject

El autómata para leer palíndromos siempre será el mismo, solamente cambia la cantidad de elementos de los dos alfabetos. No obstante, debe haber un elemento del alfabeto de la pila por cada elemento del alfabeto del lenguaje.

En la programación de *software*, por mencionar un caso, los signos de agrupación deben comportarse como palíndromos, tanto para programar como para hacer operaciones matemáticas. Así, en lugar de letras, se puede tener un alfabeto como:

Σ= (), o Σ= [], () o Σ= {}, [], ()

El palíndromo obliga a cerrar todos los paréntesis, corchetes o llaves que se abren, en el orden correcto.

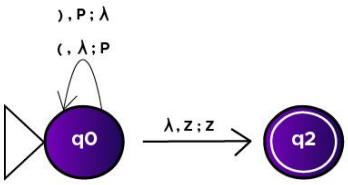
Ten en cuenta que puedes encontrarte con palíndromos compuestos, es decir, un palíndromo seguido de otro, por ejemplo: (())(). Sin embargo, el autómata debe reconocerlos.

3. Se tiene el alfabeto de lenguaje Σ= (,) y el alfabeto de pila T= P. Resolver un autómata de pila que reconozcan cadenas compuestas por uno o varios palíndromos consecutivos formados con paréntesis.

Solución:

Recuerda que todos los paréntesis que se abren, deben cerrarse. Observa cómo se cumple esto en la siguiente figura:

Figura 16. Solución de palíndromos consecutivos con paréntesis



Input	Result
()	Accept
(())	Accept
(())()	Accept
()()()()	Accept
()(())((()))	Accept
(	Reject
(()	Reject
())	Reject
(((	Reject
()(())((())	Reject

Las dos primeras cadenas son palíndromos simples. Las tres siguientes son cadenas compuestas de varios palíndromos consecutivos. El cierre correcto de los distintos niveles de paréntesis se garantiza cada vez que la pila queda vacía.

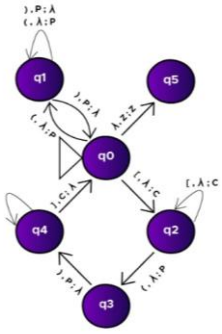
4. Resolver un autómata de pila que reconozca cadenas compuestas por uno o varios palíndromos consecutivos formados con paréntesis y corchetes.

Para este ejemplo, debes tener en cuenta que:

- Varios niveles de paréntesis pueden estar dentro de corchetes, pero los corchetes no pueden estar dentro de los paréntesis.
- Un palíndromo puede tener varios niveles de paréntesis sin usar corchetes.
- Un palíndromo puede tener varios niveles de corchetes, pero debe tener al menos un juego de paréntesis en el interior. Es decir, no acepta palíndromos que solamente tengan corchetes.

Observa la representación y la solución de este tipo de palíndromos:

Figura 17. Solución de palíndromos consecutivos con paréntesis y corchetes



Input	Result
((()))	Accept
[()]	Accept
()(())((()))	Accept
[[()]][((()))][[[((((()))))]]]	Accept
[()][()][()]	Accept
((())	Reject
[[()]	Reject
([])	Reject
((([]))	Reject
[]	Reject

Puedes apreciar que las cadenas [([])] y ([]) se rechazan a pesar de ser palíndromos, esto sucede porque contienen corchetes dentro de paréntesis.

Llegaste de manera exitosa al final de esta unidad. Sin embargo, antes de avanzar a la siguiente, es importante llevar a la práctica las generalidades de los temas vistos mediante una evidencia de aprendizaje. Lee con atención las indicaciones y consulta la rúbrica para asegurar un buen resultado.

Cierre

El recorrido por la Unidad 1 llega a su fin. Hemos aprendido que los autómatas, su gramática y lenguaje, tienen amplia utilidad en las tareas de programación para el reconocimiento de cadenas de texto. A través de ellos, puedes advertir errores de sintaxis cuando estás escribiendo código, o detectar llaves, corchetes o paréntesis que no se cerraron correctamente. Asimismo, aprendiste cómo se implementa el concepto de memoria en los autómatas de pila.

Nos preparamos para la Unidad 2, en donde descubriremos las *máquinas virtuales,* tales como las máquinas de Mealy y de Moore, usadas ampliamente en procesos de automatización industrial y en robótica; o la máquina de Turing, que podría considerarse la base misma de la programación computacional.

¡Seguimos adelante!

Referencias de imágenes

- Ridel, R. (2015). Máquina de Turing mecánica [Captura de imagen]. YouTube. https://youtu.be/vo8izCKHiF0?t=35
- Sutulweb. (2022, 5 de julio). Máquina de Turing eléctrica. https://sutilweb.com/2022/07/05/maquina-de-turing

Ver: Anexo_Biblioteca_Imágenes_Autómatas_1

.

Bibliografía

- Acevedo, A. (2006). Teoría de la computación [Tesis de grado, Instituto Tecnológico de Celaya]. Repositorio Academia. https://www.academia.edu/8275514/Teoria_Computacion
- Amaya, C. (2015). Autómatas y Lenguajes Formales. Academia.edu. https://www.academia.edu/23726621/AUTOMATAS_Y_LENGUAJES_FORMALES
- Birchenall, L. B., & Müller, O. (2014). La teoría lingüística de Noam Chomsky: del inicio a la actualidad. Lenguaje, 42(2), 417-442.
- Chacón, J. L. (2005). Lenguajes y autómatas finitos [Archivo PDF]. Universidad de Los Andes.
- Cueva, J. M., Izquierdo, R., Luengo, M. C., Ortín, F., & Labra, J. E. (2003). Lenguajes, Gramáticas y Autómatas en Procesadores de Lenguaje. Editorial SERVITEC. http://di002.edv.uniovi.es/~cueva/publicaciones/libros/36_LGA.pdf
- Hamada, M. (2013). Turing machine and automata simulators. Science Direct, 1466-1474.
- Hopcroft, J., & Ullman, J. (1969). Their Formal Languages Relation to Automata. Addison – Wesley.
- Huertas, M. (2011). Teoría de conjuntos básica. UOC. https://openlibra.com/es/book/teoria-de-conjuntos-basica
- Hurtado, J. A., Kantor, R., Luna, C., Sierra, L., & Zanarini, D. (2014). Temas de Teoría de la Computación. LATIn.
- Ivorra C. (2011). Lógica y teoría de conjuntos [Autoedición].
- Ivorra, C. (2010). Teoría Descriptiva de Conjuntos I [Autoedición]. https://www.uv.es/~ivorra/Libros/TD.pdf
- Jurado Málaga, E. (2008). Teoría de autómatas y lenguajes formales. Universidad de Extremadura.

- Jurado, E. (2008). Teoría de autómatas y lenguajes formales [Tesis doctoral, Universidad de Extremadura]. https://dehesa.unex.es/bitstream/10662/2367/1/978-84-691-6345-0.pdf
- Navarro, G. (2017). Hacia Teoría de la Computación: Lenguajes Formales, Computabilidad y Complejidad. Universidad de Chile.
- Ortega, J. L. (2004). Breves notas sobre autómatas y lenguajes. UNAM.
- Puig, S. (2012). Alan Turing lo imaginó. Historia y vida, (531), 74-77. https://www.pressreader.com/spain/historia-y-vida/20120604/282041914189522

IU Digital de Antioquia

INSTITUCIÓN UNIVERSITARIA DIGITAL DE ANTIOQUIA

www.**iudigital**.edu.co