Universidad Nacional de Ingeniería

CC321 Teoría de autómatas

PDA: Aceptación por estado final y pila vacía

Profesor:
Victor Melchor Espinoza
Departamento de Ciencias de la Computación

Grupo:
Ronaldo Lopez
Davis Alderete
Cristhian Sánchez Sauñe

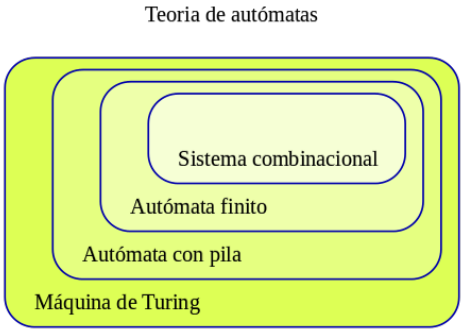
Índice

1. Marco teórico

1.1. Definición de un autómata de pila (PDA)

Antes de entrar en la definición formal de un PDA, es necesario repasar los siguientes conceptos:

- Un autómata de pila es una manera de implementar una gramática libre de contexto de la misma manera que se diseña un autómata finito determinista (DFA) para una gramática regular. Un DFA puede recordar una cantidad finita de información, pero **un PDA puede recordar una cantidad infinita de información**.
- Un autómata de pila es simplemente un autómata finito no-determinista aumentado con una "memoria de pila externa". La adición de pila se utiliza para proporcionar una capacidad de administración de memoria tipo LIFO (como se explicará posteriormente). Los autómatas de pila pueden almacenar una cantidad ilimitada de información en la pila. Un PDA puede apilar un elemento en la parte superior de la pila y des-apilar un elemento desde la parte superior de la pila.
- Un PDA es más potente que un atómata finito. Cualquier lenguaje que pueda ser aceptado por un autómata finito también puede ser aceptado por un PDA. Un PDA también acepta lenguajes que ni siquiera puede ser aceptada por un autómata finito.
- Un autómata es un modelo matemático de un sistema que recibe una cadena constituida por símbolos de un alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce. El lenguaje que reconoce un autómata de pila pertenece al grupo de los lenguajes libres de contexto en la clasificación de la Jerarquía de Chomsky.



1.2. Definición formal

Formalmente, un autómata de pila puede ser descrito como una séptupla $M = (S, \Sigma, \Gamma, \Delta, s_o, Z, F)$ donde:

- S es un conjunto finito de estados
- Σ y Γ son los alfabetos de entrada y pila, respectivamente
- Δ: es una regla de transición, $\Delta : S \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \to (S \times \Gamma^*)$
- $s_0 \in Q$ es el estado inicial o de partida
- $Z \in \Gamma$ es el símbolo inicial de la pila (en algunos textos, Z también es denotado por γ_0)
- $F \subseteq S$ es el conjunto de estados finales o de aceptación

La interpretación de $\Delta(q, a, b) = \{(q_1, \gamma_1), (q_2, \gamma_2), ..., (q_n, \gamma_n)\}$ con $q, q_i \in S$, $a \in (\Sigma \cup \{\varepsilon\})$, $b \in \Gamma$ y $\gamma_i \in \Gamma^*$ es la siguiente:

Cuando el estado del autómata es q, el símbolo que la cabeza lectora está inspeccionando en ese momento es a, y en la cima de la pila nos encontramos el símbolo b, se realizan las siguientes acciones:

- Si $a \in \Sigma$, es decir no es la cadena vacía, la cabeza lectora avanza una posición para inspeccionar el siguiente símbolo.
- Se elimina el símbolo b de la pila del autómata.
- Se selecciona un par (q_i, γ_i) de entre los existentes en la definición de $\Delta(q, a, b)$, la función de transición del autómata.
- Se apila la cadena $\gamma_i = c_1 c_2 \dots c_k$, con $c_i \in \Gamma$ en la pila del autómata, quedando el símbolo c_k en la cima de la pila.
- Se cambia al estado q_i.

1.3. Funcionamiento

Un autómata de pila cuenta con una cinta de entrada y un mecanismo de control que puede encontrarse en uno de entre un número finito de estados. Uno de estos estados se designa como estado inicial, y además algunos estados se llaman de aceptación o finales.

A diferencia de los autómatas finitos, los autómatas de pila cuentan con una memoria auxiliar llamada pila. Los símbolos (llamados símbolos de pila) pueden ser insertados o extraídos de la pila, de acuerdo con el manejo last-in-first-out (LIFO).

Las transiciones entre los estados que ejecutan los autómatas de pila dependen de los símbolos de entrada y de los símbolos de la pila. El autómata acepta una cadena x si la secuencia de transiciones, comenzando en el estado inicial y elemento pila inicial, conduce a un estado final, después de leer toda la cadena x.

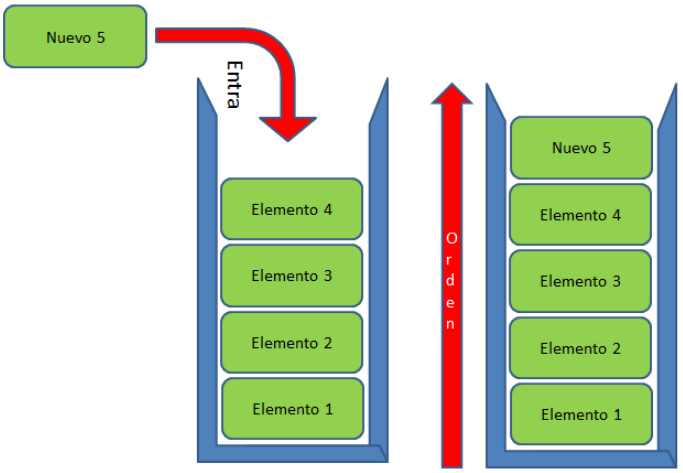
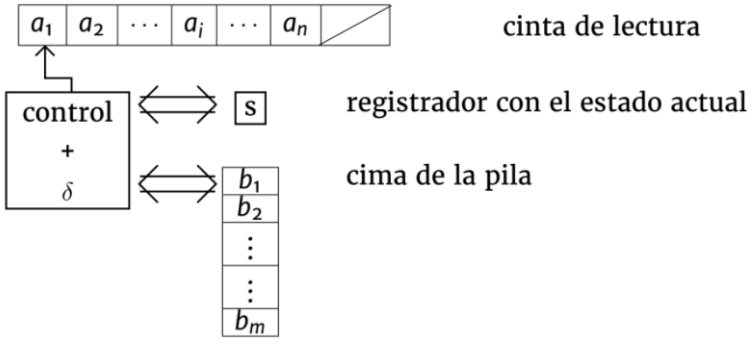


Figura 1: Estructura LIFO de una pila

1.4. Representación

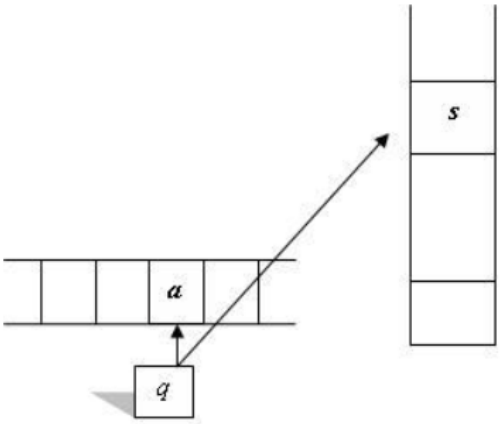
Una máquina de este tipo se representa de la siguiente forma:



Al igual que un autómata finito un autómata de pila cuenta con un flujo de entrada y un flujo de control que puede encontrarse en uno de entre un número finito de estados. Uno de estos estados se designa como el inicial y por lo menos un estado es de aceptación.

Observación

En un momento dado, la unidad de control del autómata escanea un símbolo a sobre la cinta de entrada y el símbolo s en la cima de la pila.



$$\Delta(q, a, s) = (q^{'}, \gamma)$$

1.5. Transición

$$(\text{p, a, A}), (\text{q, } B_1B_2...B_k) \in \Delta$$

Significa que la máquina estando en el estado p, leyendo el símbolo de entrada a en la cinta y con A en la cima de la pila, puede des-apilar A y apilar los símbolos $B_2...B_k$ (B_1 en la cima de pila y B_k al fondo), mover el cabezal lector una celda a la derecha de a y entrar en el estado q.

La forma habitual de describir una transición es:

$$\Delta(\text{p, a, A}) = \{(q_1, \gamma_1), (q_2, \gamma_2), ..., (q_m, \gamma_m)\}$$

Donde $q_i \in$ Q ($1 \leq$ i $\leq$ m), $\gamma_i \in \Gamma^*$. La transición indica que estando en el estado p, A en la cima de la pila y leyendo el símbolo a en la cinta, puede entrar en cualquier estado q_i, des-apilar A y apilar en su lugar la secuencia γ_i y avanzar una celda a la derecha el cabezal de la lectora.

1.6. Descripción Instantánea

Se llama descripción instantánea o configuración del autómata, a una terna (q, u, a) $\in$ Q $\times\Sigma^* \times \Gamma^*$ en el que q es el estado en el que se encuentra el autómata, u es parte de la cadena de entrada que falta leer y a es el contenido de la pila (el primer símbolo es el cima de la pila).

Una configuración ofrece información completa del estado global de la máquina en un instante durante un determinado cómputo.
Sea M es una autómata de pila y $u \in \Sigma^*$. Se llama configuración inicial a la entrada (q_0, u, Z_0), donde q_0 es el estado inicial y Z_0 es el símbolo inicial de la pila.

1.7. Pasos Computables

- $\vdash$ representa un movimiento
- $\vdash^*$ describe una secuencia de movimientos

Si C_1 y C_2 son dos configuraciones, se dice que se puede llegar de C_1 a C_2 mediante una sucesión de **pasos computables**, y se escribe $C_1 \vdash^* C_2$ si y solo si existe una sucesión de configuraciones $T_1, ..., T_n$ tal que $C_1 = T_1 \vdash T_2 \vdash ... \vdash T_n = C_2$.

Por ejemplo,

$(p, b, T) \vdash (q, w, \alpha)$

En el ejemplo anterior, mientras se realiza una transición del estado p a q, se consume el símbolo de entrada b y la parte superior de la pila T se representa mediante una nueva cadena α.

1.8. Ejemplos

Ejemplo 1: Sea el siguiente lenguaje libre de contexto $L = \{a^k b^k | k \geq 0\}$, formado por las cadenas $L = \{\epsilon, ab, aabb, aaabbb, aaaabbbb, \ldots\}$.

Dicho lenguaje puede ser reconocido por el siguiente autómata de pila:

$M = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \{A, \underline{A}\}, \Delta, q_0, \{q_0, q_3\})$, donde las transiciones son:

$\Delta(q_0, a, \epsilon) = \{(q_1, \underline{A})\}$
$\Delta(q_1, a, \epsilon) = \{(q_1, \underline{A})\}$
$\Delta(q_1, b, \underline{A}) = \{(q_2, \epsilon)\}$
$\Delta(q_1, b, \underline{A}) = \{(q_3, \epsilon)\}$
$\Delta(q_2, b, \underline{A}) = \{(q_2, \epsilon)\}$
$\Delta(q_2, b, \underline{A}) = \{(q_3, \epsilon)\}$
$\Delta(q, \theta, \rho) = \emptyset$ para cualquier (q, θ, ρ)

El significado de las transiciones puede ser explicado analizando la primera transición:

$\Delta(q_0, a, \epsilon) = \{(q_1, \underline{A})\}$

donde q_0 es el estado actual, a es el símbolo en la entrada y ϵ se extrae de la cima de la pila. Entonces, el estado del autómata cambia a q_1 y el símbolo $\underline{A}$ se coloca en la cima de la pila.

La idea del funcionamiento del autómata es que al ir leyendo los diferentes símbolos a, estos pasan a la pila en forma de símbolos A. Al aparecer el primer símbolo b en la entrada, se comienza el proceso de des-apilado, de forma que coincida el número de símbolos b leídos con el número de símbolos A que aparecen en la pila.

Ejemplo 2: Diseñe un PDA para aceptar un lenguaje $L = \{a^n b^{2n} / n >= 1\}$.

Solución: En este lenguaje, n es el número de a's que deben ir seguidos de $2n$ veces b. Por lo tanto, aplicaremos una lógica muy simple, y es que si leemos una sola a, vamos a apilar dos a en la pila. Tan pronto como leamos b, entonces para cada b sólo una a debe ser des-apilada.

La descripción instantánea se puede construir de la siguiente manera:

$$\Delta(q_0, a, Z) = (q_0, aaZ)$$
$$\Delta(q_0, a, a) = (q_0, aaa)$$

Ahora, cuando leamos b, cambiaremos el estado de q_0 a q_1 y comenzaremos a des-apilar la a correspondiente. por lo tanto

$$\Delta(q_0, b, a) = (q_1, \epsilon)$$

Por lo tanto, este proceso de des-apilar se repetirá a menos que se lean todos los símbolos. Tenga en cuenta podemos des-apilar solo en el estado q_1.

$$\Delta(q_1, b, a) = (q_1, \epsilon)$$

Después de leer todas las b, todas las a's correspondientes deben ser des-apiladas. Por lo tanto, cuando leemos ϵ como símbolo de entrada, entonces no debería haber nada en la pila. Por lo tanto, el movimiento será:

$$\Delta(q_1, \epsilon, Z) = (q_2, \epsilon)$$

donde

$$PDA = \{(q_0, q_1, q_2\}, \{a, b\}, \{a, Z\}, \Delta, q_0, Z, \{q_2\})$$

Podemos resumir la descripción instantánea como:

$\Delta(q_0, a, Z) = (q_0, aaZ)$
$\Delta(q_0, a.a) = (q_0, aaa)$
$\Delta(q_0, b, a) = (q_1, \epsilon)$
$\Delta(q_1, b, a) = (q_1, \epsilon)$

$\Delta(q_1, \epsilon, Z) = (q_2, \epsilon)$

Ahora podemos simular este PDA para una entrada **aaabbbbbb**.

$\Delta(q_0, aaabbbbbb, Z)$
$\vdash \Delta(q_0, aaabbbbbb, Z)$
$\vdash \Delta(q_0, aabbbbbb, aaZ)$
$\vdash \Delta(q_0, abbbbbb, aaaaZ)$
$\vdash \Delta(q_0, bbbbbb, aaaaaaZ)$
$\vdash \Delta(q_1, bbbbb, aaaaaZ)$
$\vdash \Delta(q_1, bbbb, aaaaZ)$
$\vdash \Delta(q_1, bbb, aaaZ)$
$\vdash \Delta(q_1, bb, aaZ)$
$\vdash \Delta(q_1, b, aZ)$
$\vdash \Delta(q_1, \epsilon, Z)$
$\vdash \Delta(q_2, \epsilon)$

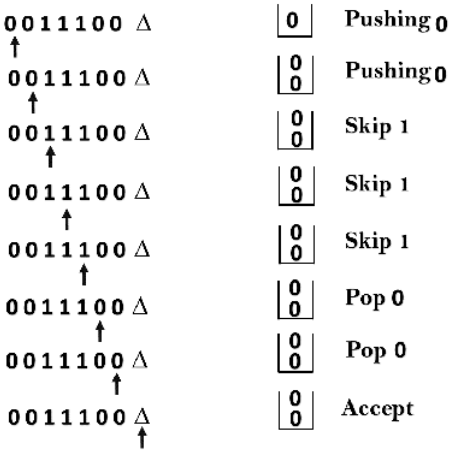
La entrada es aceptada.

Ejemplo 3: Diseñe un PDA para aceptar un lenguaje $L = \{0^n 1^m 0^n / m, n >= 1\}$.

Solución: En este PDA, n número de 0's son seguidos por cualquier número de 1 seguido n número de 0's. Por lo tanto, la lógica para el diseño de dicha PDA será la siguiente:

Apilamos todos los 0 en la pila al encontrar los primeros 0. Entonces si leemos 1, no hacemos nada. A continuación, leemos 0, y en cada lectura de 0, des-apilamos un 0 de la pila.

Por ejemplo:



Este escenario se puede escribir en forma de una descripción instantánea:

$\Delta(q_0, 0, Z) = (q_0, 0Z)$
$\Delta(q_0, 0, 0) = (q_0, 00)$
$\Delta(q_0, 1, 0) = (q_1, 0)$
$\Delta(q_1, 1, 0) = (q_1, 0)$
$\Delta(q_1, 0, 0) = (q_1, \epsilon)$
$\Delta(q_1, \epsilon, Z) = (q_2, Z)$

Ahora simularemos este PDA para una cadena de entrada **0011100**.

$\Delta(q_0, 0011100, Z)$
$\vdash \Delta(q_0, 011100, 0Z)$
$\vdash \Delta(q_0, 11100, 00Z)$
$\vdash \Delta(q_0, 1100, 00Z)$
$\vdash \Delta(q_1, 100, 00Z)$
$\vdash \Delta(q_1, 00, 00Z)$
$\vdash \Delta(q_1, 0, 0Z)$
$\vdash \Delta(q_1, \epsilon, Z)$
$\vdash \Delta(q_2, \epsilon)$

La entrada es aceptada.

2. Criterios de aceptación

Existen 2 formas de aceptar un lenguaje mediante un autómata de pila

2.1. Estado final

Reconoce el lenguaje ingresado en el autómata de pila si se logra llegar al estado final seleccionado sin la necesidad que la pila este vacía.

Ejemplo:

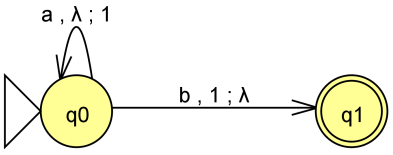
Diseñar un autómata de pila que permita reconocer el siguiente lenguaje

$$L = \{a^n b | n \geq 1\}$$

y además que reconozca la cadena **aaab**.

Solución:

El diagrama de transición del autómata de pila respectivo será el siguiente



Estados	cadena de entrada	pila
Estado q0	aaab	Z
Estado q0	aaab	1Z
Estado q0	aaab	11Z
Estado q0	aaab	111Z
Estado q1	aaab	11Z

2.2. Pila vacía

Reconoce el lenguaje ingresado en el autómata de pila si se logra eliminar todos los elementos de la pila sin la necesidad de que éste llegue al estado final. En este tipo de criterio, no será necesario declarar un estado final. Ejemplo:

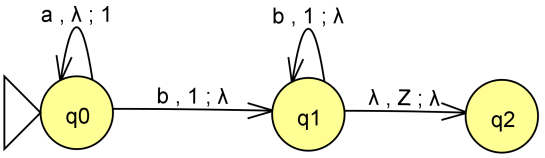
Diseñar un autómata de pila que permita reconocer el siguiente lenguaje

$$L = \{a^n b^n | n \geq 1\}$$

y además que reconozca la cadena **aaabbbb**.

Solución:

El diagrama de transición del autómata de pila respectivo será el siguiente:



Estados	cadena de entrada	pila
Estado q0	aaabbb	Z
Estado q0	aaabbb	1Z
Estado q0	aaabbb	11Z
Estado q0	aaabbb	111Z
Estado q1	aaabbb	11Z
Estado q1	aaabbb	1Z
Estado q1	aaabbb	Z
Estado q2	aaabbb	vacío

2.3. Equivalencia de aceptación por estado final y pila vacía

Sea L el lenguaje de aceptación por estado final, y sea N el lenguaje de aceptación por pila vacía.

- Si $L = N(P1)$ para algún PDA $P1$, entonces hay un PDA $P2$ tal que $L = L(P2)$. Eso significa que el lenguaje aceptado por PDA de pila vacía también será aceptado por el PDA de estado final.

- Si hay un lenguaje $L = L(P1)$ para algún PDA $P1$, entonces hay un PDA $P2$ tal que $L = N(P2)$. Eso significa que el lenguaje aceptado por el PDA de estado final también es aceptado por el PDA de pila vacía.

3. Autómata de pila no determinista (NPDA)

Son muy similares a los autómatas finitos no deterministas. Las gramáticas libres de contexto que aceptan PDA's deterministas también acepta PDA no deterministas. Del mismo modo, hay algunas gramáticas libres de contexto que pueden ser aceptadas sólo por NPDA y no por DPDA. Por lo tanto, NPDA es más poderoso que el DPDA.

3.1. Definición formal

Un autómata finito de pila no determinista (APND) consta de los mismos parámetros de un APD.

$$\mathrm{M} = (\mathrm{Q},\ \Sigma, \Gamma, \Delta,\ \mathrm{s},\ \mathrm{F},\ \mathrm{z})$$

Donde:

Q: es conjunto finito de estados
Σ: es un alfabeto de entrada
Γ: es un alfabeto llamado alfabeto de pila
Δ: es una regla de transición, Δ: Q $\times(\Sigma \cup \{\varepsilon\})$ $\times\Gamma \to P(Q \times \Gamma^*)$. Donde $P(Q \times \Gamma^*)$ es un conjunto de subconjuntos finitos de $Q \times \Gamma^*$
s $\in$ Q es el estado inicial o de partida
F $\subseteq$ Q es el conjunto de estados finales o de aceptación
z $\in \Gamma$ es el símbolo inicial o de partida

Es decir, el comportamiento del autómata depende en cada transición de:

- El estado actual.
- Posiblemente del siguiente símbolo de la entrada.
- El símbolo en la cima de la pila.

Y se modifica el autómata en el sentido de que:

- Se cambia (posiblemente) el estado.
- Se selecciona (posiblemente) el siguiente símbolo de la entrada.

- Se modifica (posiblemente) el contenido de cima de la pila.

Ejemplo: Sea el autómata M = $(\{q_0, q_1, q_2\}, \{a, b\}, \{Z, A, \epsilon\}, \Delta, q_0, q_2, Z)$ donde:

$\Delta(q_0, a, Z) = \{(q_0, AZ)\}$
$\Delta(q_0, \epsilon, Z) = \{(q_2, Z)\}$
$\Delta(q_0, a, A) = \{(q_0, AA)\}$
$\Delta(q_0, b, A) = \{(q_1, \epsilon)\}$
$\Delta(q_1, b, a) = \{(q_1, \epsilon)\}$
$\Delta(q_1, \epsilon, Z) = \{(q_2, Z)\}$

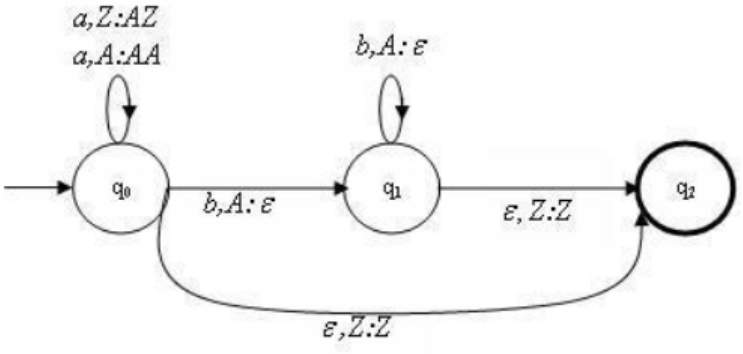


Figura 2: Diagrama de estado correspondiente

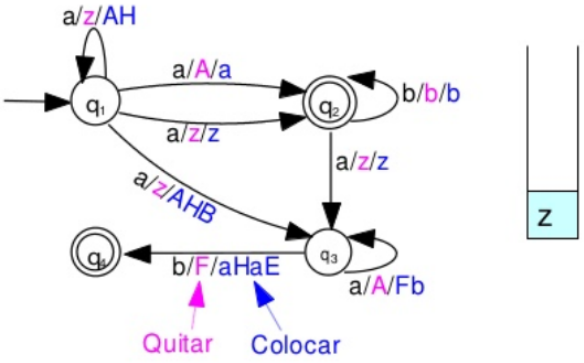
3.2. Lenguaje aceptado por un Autómata no Determinista

Sea M = (Q, Σ, Γ, Δ, s, F, z) un autómata de pila no determinista. El lenguaje aceptado por M se denota por L(M) y es el conjunto:

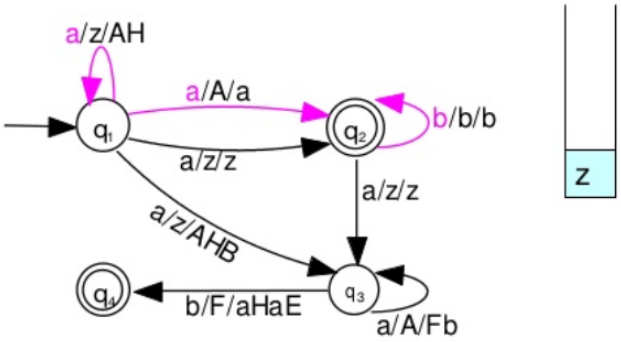
$$L(M) = \{w \in \Sigma^* | (s, w, z) \vdash (p, \varepsilon, u) \; para \; p \in F \; y \; u \in \Gamma^*\}$$

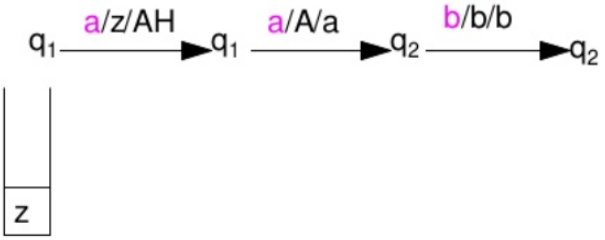
3.3. Aceptación de un lenguaje

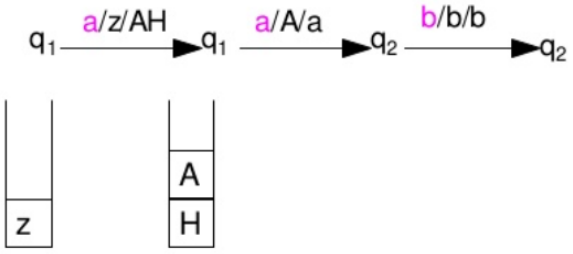
Ejemplo:

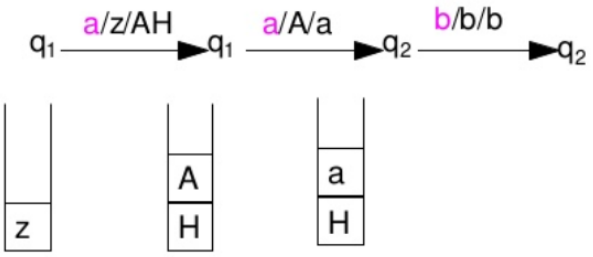


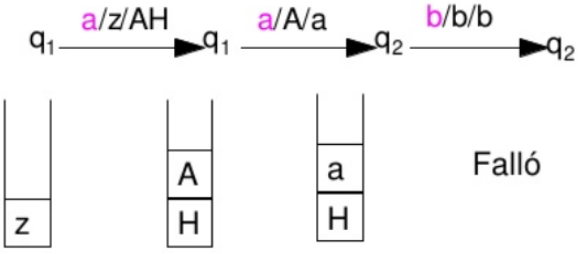
- Diagrama 1, ¿Aceptará **aab**?



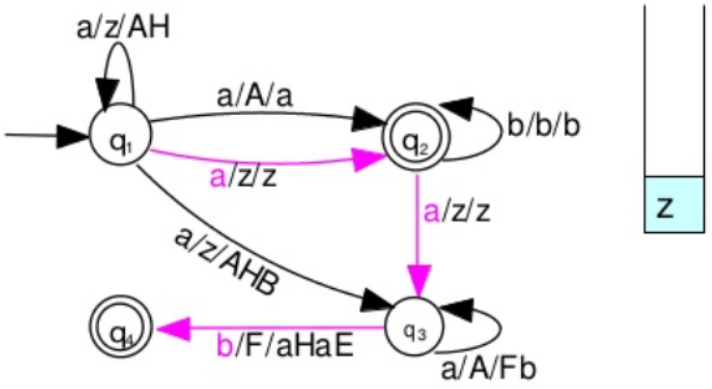


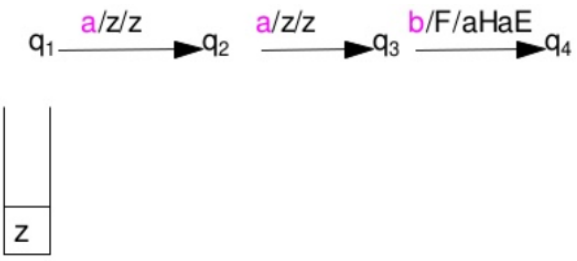


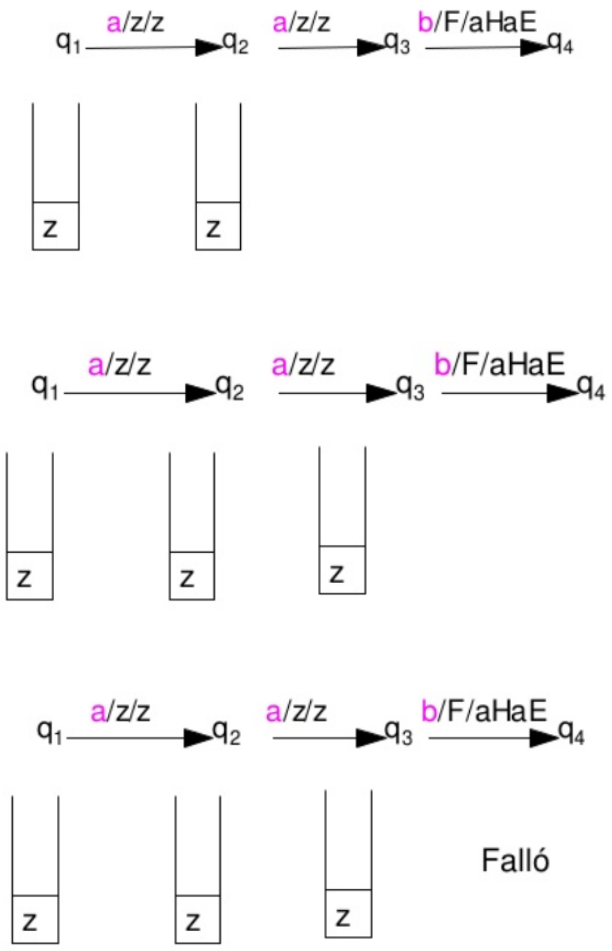




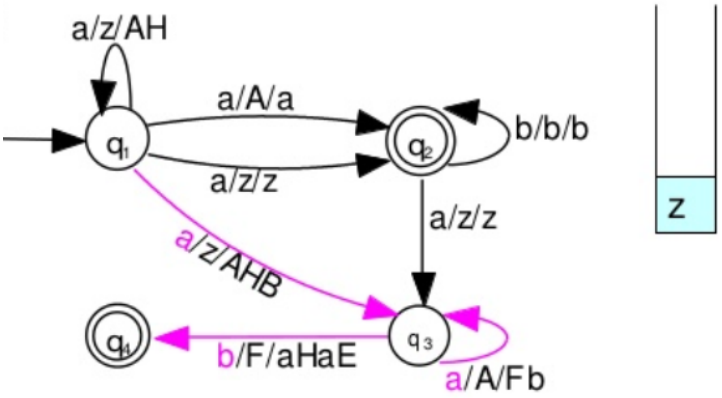
- Diagrama 2, ¿Aceptará **aab**?

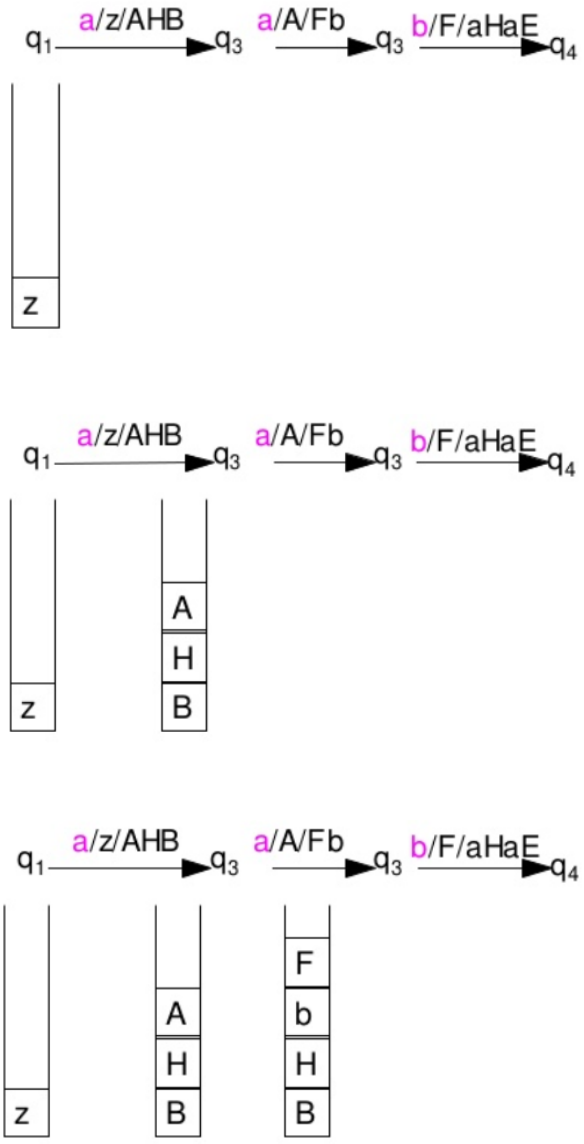


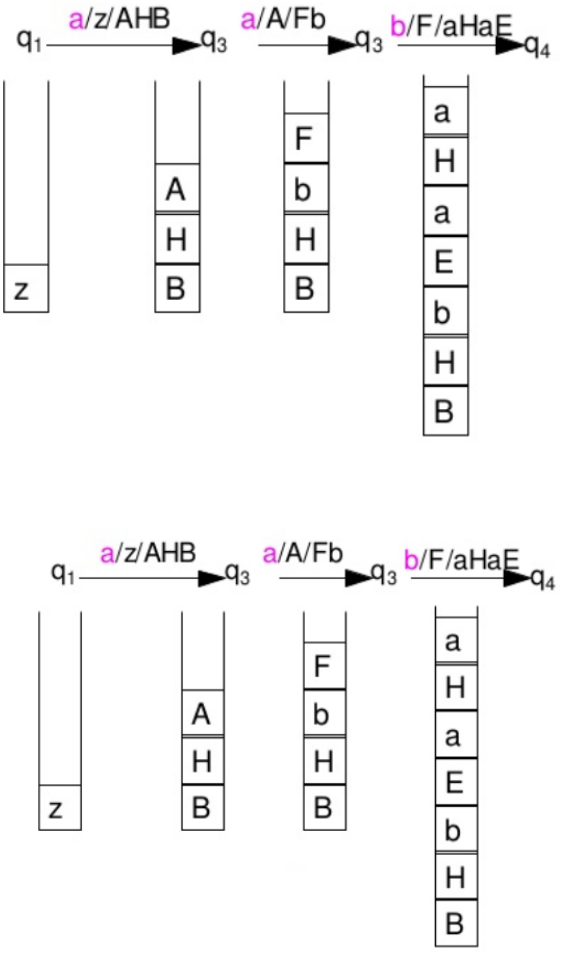




- Diagrama 3, ¿Aceptará **aab**?



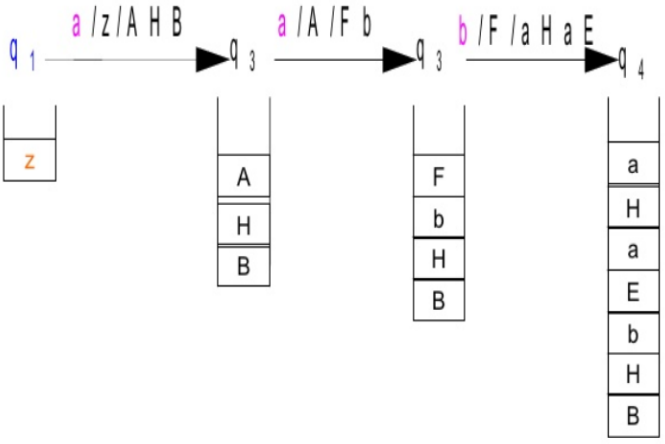

q1
a/z/AHB
q3
a/A/Fb
q3
b/F/aHaE
q4
z
A
H
B
F
b



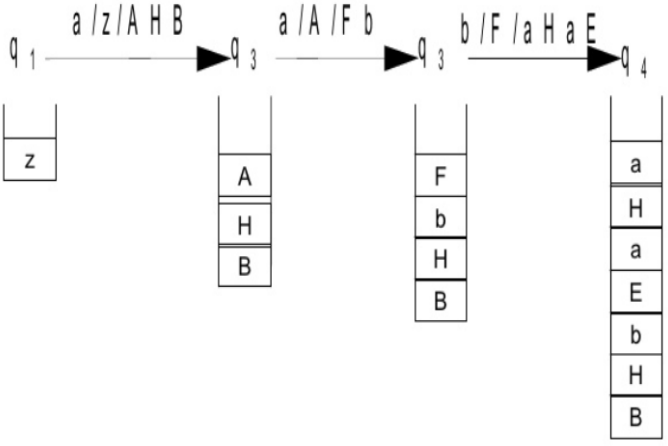
El autómata acepta **aab**.

3.4. Descripción Instantánea

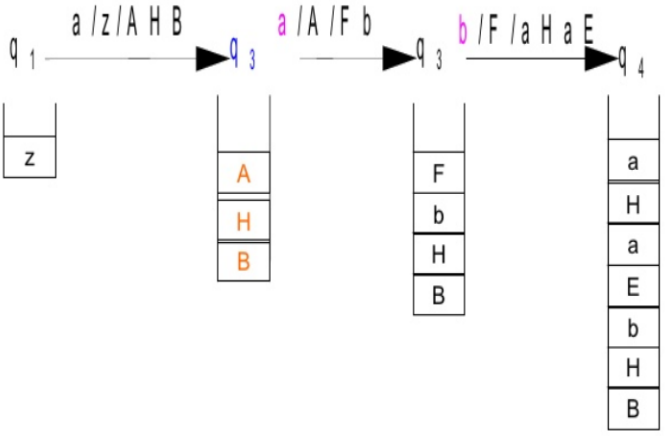
$$(q_1, aab, z)$$



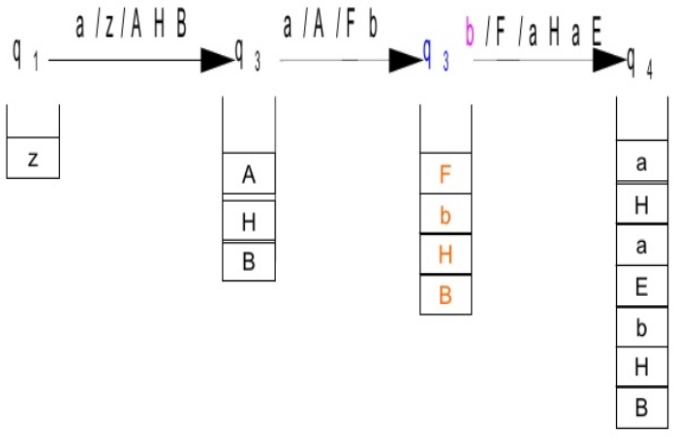
$(q_1, aab, z) \vdash (..)$



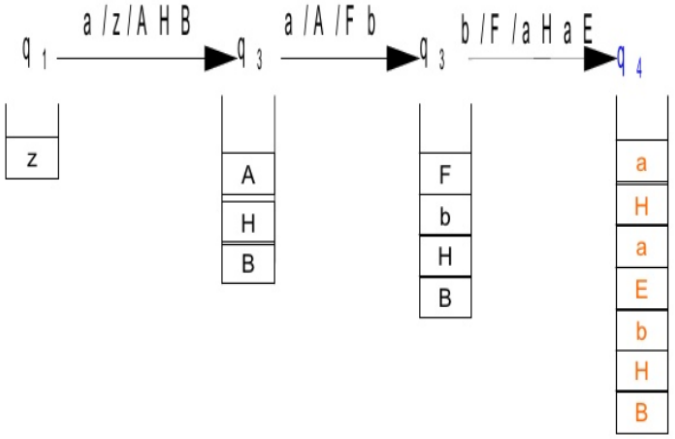
$(q_1, aab, z) \vdash (q_3, ab, AHB)$



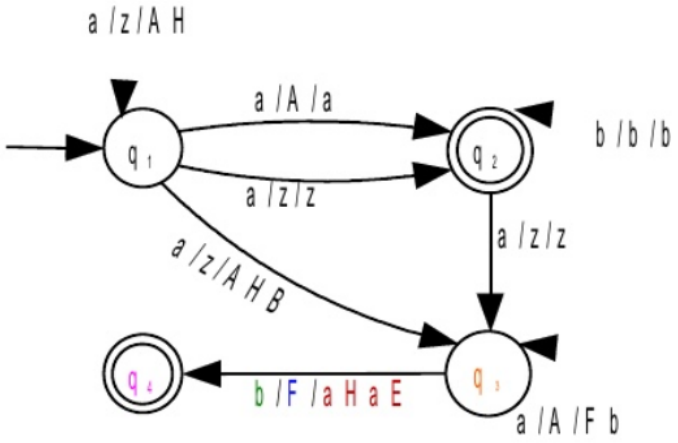
$$(q_1, aab, z) \vdash (q_3, ab, AHB) \vdash (q_3, b, FbHB)$$



$$(q_1, aab, z) \vdash (q_3, ab, AHB) \vdash (q_3, b, FbHB) \vdash (q_4, \epsilon, aHaEbHB)$$



Transiciones restantes:



$$\Delta(q_3, b, F) = \{(q_4, aHaE)\}$$

$\Delta(q_1, a, z) = \{(q_1, AH), (q_2, z), (q_3, AHB)\}$
$\Delta(q_1, a, A) = \{(q_2, a)\}$
$\Delta(q_2, b, b) = \{(q_2, b)\}$
$\Delta(q_2, a, z) = \{(q_3, z)\}$
$\Delta(q_3, a, A) = \{(q_3, Fb)\}$
$\Delta(q_3, b, F) = \{(q_4, aHaE)\}$

4. Conversión de una gramática de libre contexto a un autómata de pila

Una gramática de libre contexto no puede ser representado directamente por un autómata finito, ya que no posee las propiedades suficientes para su conversión, es por eso que la mejor manera de representar dicha gramática es mediante un autómata de pila.

Ejemplo:

Sea $\Sigma_T = \{a, b\}$ y sea P dado por:

$$S ::= aS|aB$$
$$B ::= bC$$
$$C ::= aC|a$$

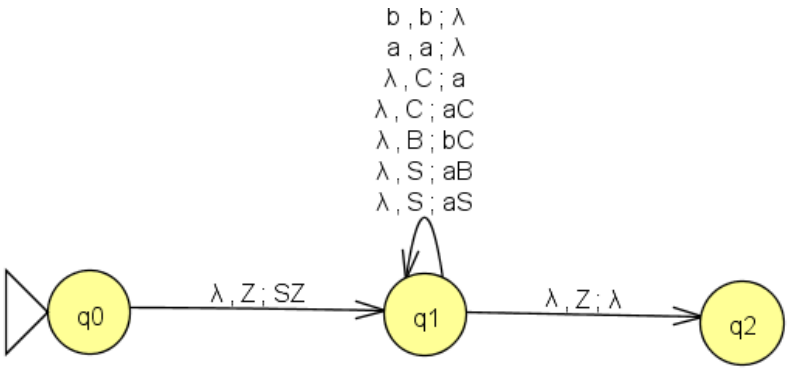
Convertir la gramática de libre contexto **P** a un autómata de pila y que reconozca la cadena **aaabaaa**.

Solución:

$$L(G) = \{a^n b a^m / n, m \geq 1\}$$

Pasos

- La transición inicial de q_0 a q_1 servirá para ingresar a la pila el elemento no terminal inicial después del elemento inicial de la pila Z. Para nuestro ejemplo el elemento inicial es S.
- Una vez que esté en el estado q_1, el autómata de pila realizará auto transiciones en el estado q_1 hasta que los todos los elementos de la cadena hallan sido leídos.
- Las reglas de nuestra gramática realizará las auto transiciones de la siguiente manera.
- No habrá ningún elemento en la entrada, el elemento que se va a quitar a la pila es el elemento no terminal que esta a la izquierda de la regla y el elemento que se añadirá a la pila serán los elementos terminales y no terminales que estén a la derecha de la regla. Repetir este paso para todas las reglas.
- Una vez realizado las auto transiciones de las reglas, también debemos añadir las auto transiciones de los elementos terminales, y lo harán de la siguiente manera.
- El elemento de entrada sera el elemento terminal. El elemento que se quitará de la pila también será el mismo elemento terminal y no se agregará elementos a la pila. Repetir este paso para todos los elementos terminales.
- Si la cadena cumple con la gramática, entonces en la pila solo nos quedará el elemento inicial de la pila. Para quitarlo de la pila usamos una transición de q_1 a q_2 para quitar el elemento Z. Una vez la pila esté vacía, la cadena quedará aceptada por el criterio de pila vacía.



Estados	cadena de entrada	pila
Estado q0	aaabaaa	Z
Estado q1	aaabaaa	SZ
Estado q1	aaabaaa	aSZ
Estado q1	aaabaaa	SZ
Estado q1	aaabaaa	aSZ
Estado q1	aaabaaa	SZ
Estado q1	aaabaaa	aBZ
Estado q1	aaabaaa	BZ
Estado q1	aaabaaa	bCZ
Estado q1	aaabaaa	CZ
Estado q1	aaabaaa	aCZ
Estado q1	aaabaaa	CZ
Estado q1	aaabaaa	aCZ
Estado q1	aaabaaa	CZ
Estado q1	aaabaaa	aZ
Estado q1	aaabaaa	Z
Estado q2	aaabaaa	vacío

5. Aplicaciones

En esta parte nos concentraremos en la aplicaciones que nos brinda este tipo especial de autómata.

5.1. Implementación Python

Se ha desarrollado un autómata de pila determinista, al cual se le proporcionará todos los datos necesarios (estado inicial, estado de aceptación, etc) para poder construirlo. Se evaluarán los ejemplos 2 y 3 presentados en la sección 1.

A continuación se muestra la función encargada de crear el árbol de búsqueda, que será usada para verificar posteriormente si una determinada cadena pertenece a un determinado lenguaje.

```
# Generamos de forma recursiva todo el árbol posible
# y finalizamos con éxito
def generate(state, input, stack, config):
        global productions
        global found

        total = 0

        # comprobamos el éxito de otro nodo de árbol
        if found:
                return 0

        # comprobamos si nuestro nodo puede terminar con éxito
        if is_found(state, input, stack):
                found = 1 # marcamos que la palabra es aceptada
                          # para que otros nodos del árbol sepan
                          # y terminen


                # añadimos satisfactoriamente la configuración
                accepted_config.extend(config)

                return 1

        # verificamos si hay más movimientos
        # (o debemos terminar)
        moves = get_moves(state, input, stack, config)
```

```
if len(moves) == 0:
        return 0


# para cada movimiento creamos un árbol
for i in moves:
        total = total + generate(i[0], i[1], i[2],
                                 config + [(i[0], i[1],
                                 i[2])])

return total
```

5.2. JFLAP

Es una extensa herramienta visual e interactiva que hace posible crear y simular autómatas con el objetivo de experimentar con ellos, con sus lenguajes y gramáticas. Requiere la instalación previa de java para poder ejecutarla correctamente.

5.2.1. Pasos de instalación

- Debemos ir a la pagina web http://www.jflap.org/
- En la parte izquierda seleccionamos la opción **Get JFLAP**. Una vez, dentro hacemos click en **fill out this form**.
- Omitimos todo el formulario, ingresamos la palabra que aparece en la parte inferior y seleccionamos **Submit**.

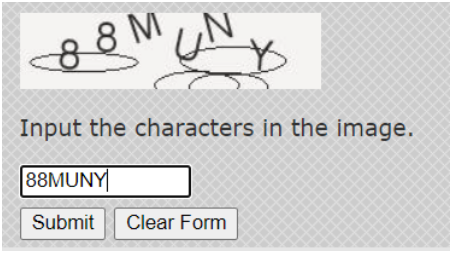


Figura 3: Completar el captcha

- Nos llevará a otra página y buscaremos la opción **JFLAP.jar**. Descargaremos la aplicación y ya podremos acceder a la simulación.

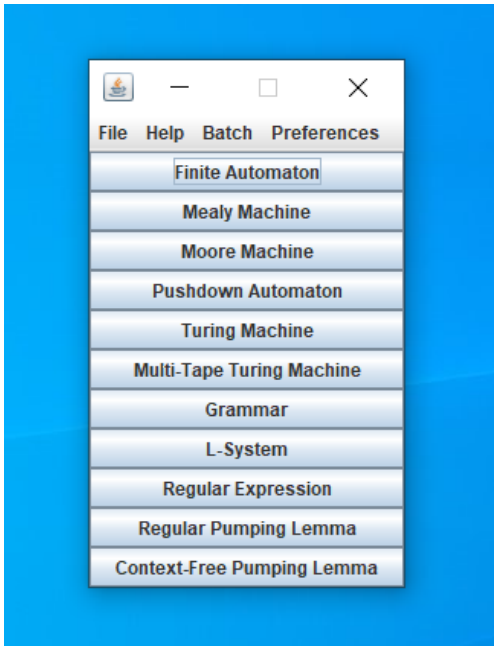
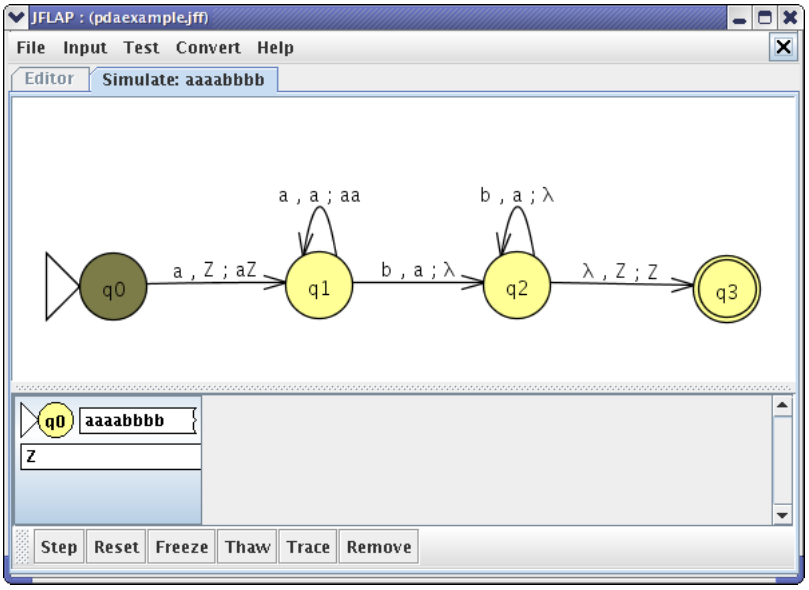
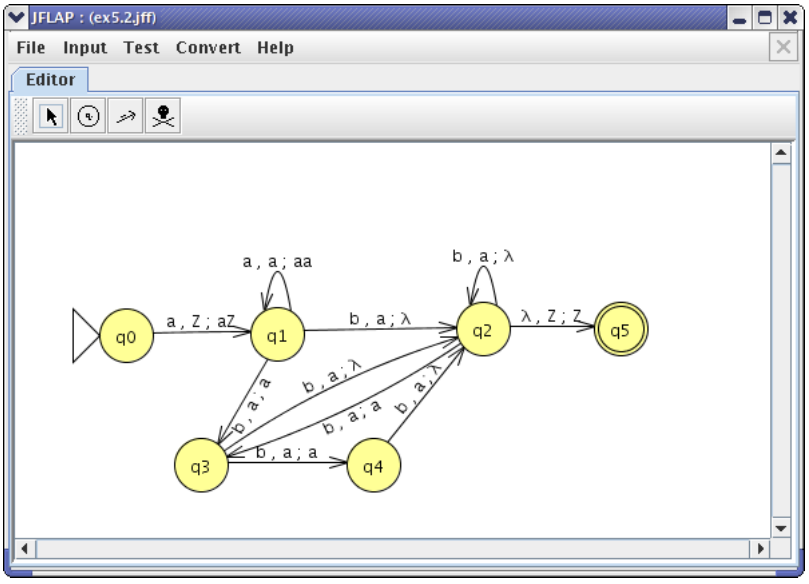


Figura 4: Inicio de la aplicación

5.2.2. Muestras

A continuación se muestran algunos ejemplos de autómatas de pila, asi como su respecto diagrama de estados.





5.3. Procesamiento del Lenguaje Natural (NLP)

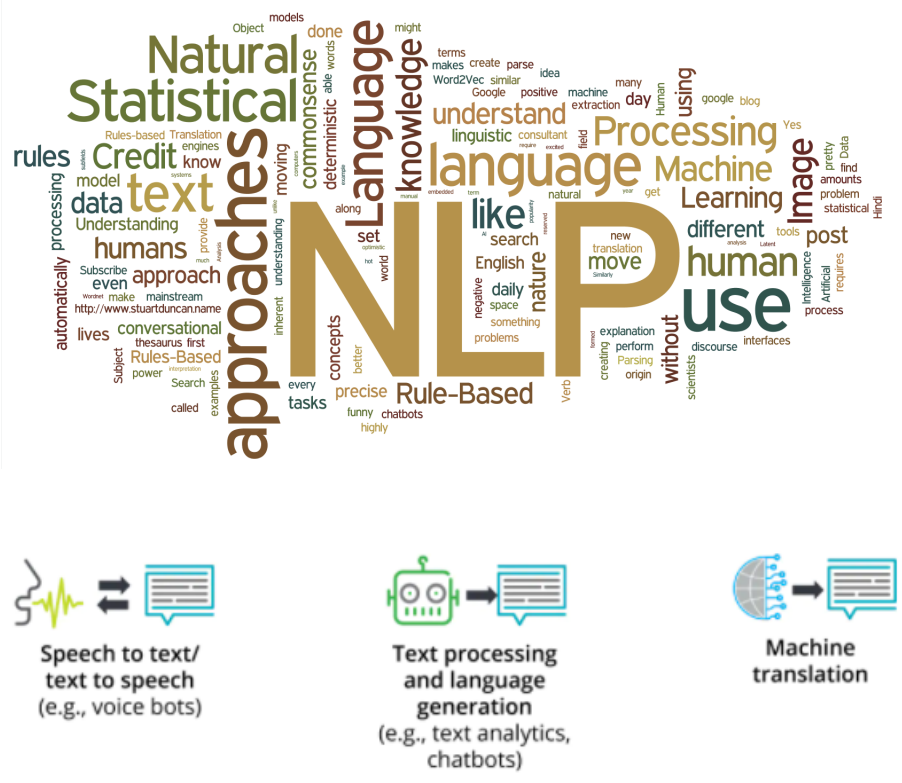


Figura 5: Aplicaciones actuales de NLP

¿Cuál es más confiable y eficiente en NLP, una máquina de estado finito o un autómata de pila?

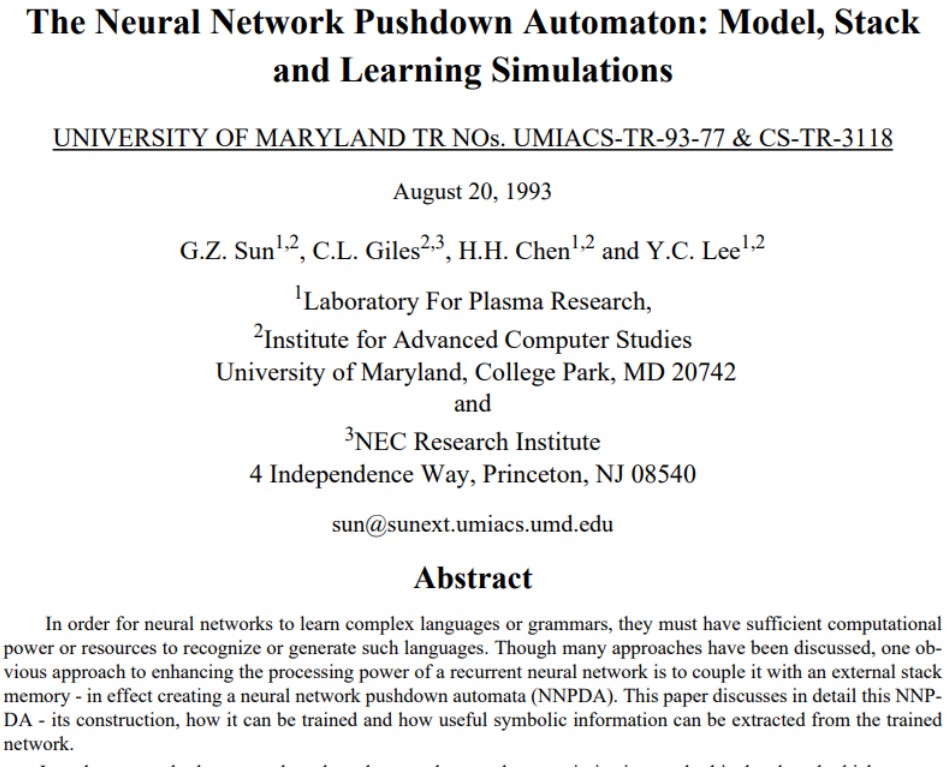
En primer lugar, ni el autómata de estado finito (FSA) ni el autómata de pila (PDA) son técnicas suficientes para modelar el lenguaje. Los FSA pueden manejar idiomas regulares. Sin embargo, ni siquiera pueden responder a la pregunta de si una palabra es un palíndromo. Los PDA son un poco más poderosos y pueden responder a tales preguntas. Las máquinas Turing proporcionan un cálculo universal y son útiles para escribir programas de complejidad arbitraria.

Los idiomas naturales no son idiomas normales. Por lo tanto, no pueden ser manejados por las FSA. Algunas gramáticas libres de contexto, son manejadas por un PDA, sin embargo, el lenguaje humano natural no está libre de contexto. Como ejemplo, las tres instrucciones siguientes: *"Jill condujo a la tienda de comida para conocer a su amiga Sally antes de recoger a sus hijos. Sally compró tres cajas de cereales. Luego condujo a la escuela"*. Si bien esta es una gramática pobre, es 'natural' en que son expresiones que la gente hace y generalmente son analizables por otras personas. En la tercera oración se refiere claramente a Jill, ya que ella es la que tiene hijos. Sin embargo, es ambiguo y tenemos que inferir esa asociación.

La cantidad de ambigüedad de contexto en el lenguaje humano natural hace imposible analizarlo de forma determinada. En su lugar, nos dirigimos hacia los campos de las estadísticas y la teoría de decisiones para hacer nuestras inferencias sobre el modelo de máxima probabilidad para la comunicación.

La vanguardia esta en utilizar el aprendizaje profundo. Muchas mejoras significativas se muestran en NLP mediante técnicas de aprendizaje profundo, en comparación con sus contrapartes clásicas basadas en reglas.. Esto ha ocurrido debido a la enorme cantidad de potencia de procesamiento que está disponible a bajo costo.

Sin embargo, ello no ha sido impedimento para que algunos investigadores creen híbridos. Un ejemplo de ello se muestra a continuación.



The Neural Network Pushdown Automaton: Model, Stack and Learning Simulations

UNIVERSITY OF MARYLAND TR NOs. UMIACS-TR-93-77 & CS-TR-3118

August 20, 1993

G.Z. Sun[1,2], C.L. Giles[2,3], H.H. Chen[1,2] and Y.C. Lee[1,2]

[1]Laboratory For Plasma Research,
[2]Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
and
[3]NEC Research Institute
4 Independence Way, Princeton, NJ 08540

sun@sunext.umiacs.umd.edu

Abstract

In order for neural networks to learn complex languages or grammars, they must have sufficient computational power or resources to recognize or generate such languages. Though many approaches have been discussed, one obvious approach to enhancing the processing power of a recurrent neural network is to couple it with an external stack memory - in effect creating a neural network pushdown automata (NNPDA). This paper discusses in detail this NNPDA - its construction, how it can be trained and how useful symbolic information can be extracted from the trained network.

Figura 6: Investigadores usan PDA's para solucionar problemas de memoria en las Redes Neuronales Recurrentes

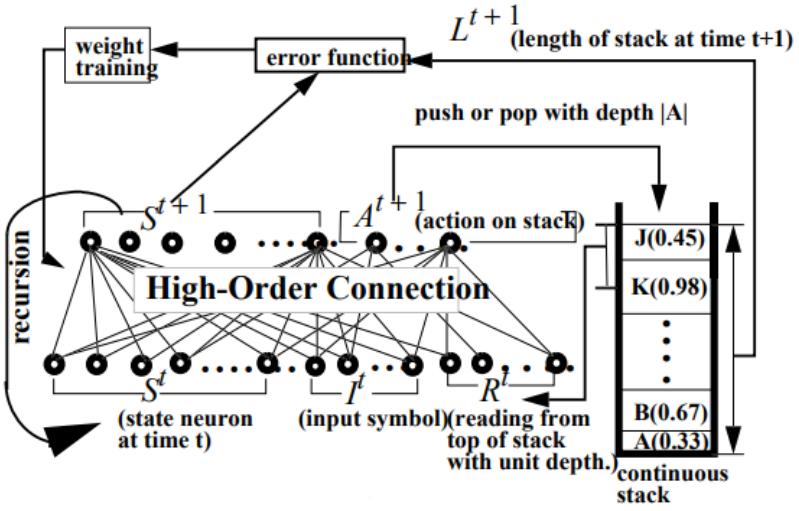


Figura 7: El diagrama esquemático de la NNPDA, donde una red recurrente de orden superior se acopla con una pila continua externa. Las entradas a la red neuronal son los estados internos actuales (S^t), los símbolos de entrada (I^t) y la lectura de la pila (R^t). Las salidas de la red neuronal son el estado interno de la próxima vez (S^{t+1}) y la acción de la pila (A^{t+1}). Esta acción se realizará en la pila externa, que a su vez renovará la siguiente lectura de la pila (R^{t+1}). Los pesos del controlador de red neuronal recurrente se entrenarán minimizando la función de error, que es una función del estado final y la longitud de la pila al final de la cadena de entrada.

6. Referencias Bibliográficas

- H. Rodger, S., 2021. JFLAP. [online] Jflap.org. Available at: http://www.jflap.org [Accessed 9 February 2021].
- Sun, G. Z., Giles, C. L., Chen, H. H., & Lee, Y. C. (2017). The neural network pushdown automaton: Model, stack and learning simulations. arXiv preprint arXiv:1711.05738.
- Hopcroft, J., Ullman, J. and Motwani, R., 2002. Introduction to the theory of automata, languages and computation. Madrid: Addison-Wesley.