
TEMA 5:

Autómatas con Pila

Autómatas y Lenguajes Formales.
Grado en Informática (2º curso).

Dpto. de Ingeniería de la Información y las Comunicaciones.



UNIVERSIDAD
DE MURCIA

Índice general

1.	Modelo de autómeta con pila	3
1.1.	Funcionamiento de un AP a grandes rasgos	3
1.2.	Definición formal de un AP	6
1.3.	Lenguaje aceptado por un AP	8
2.	Diseño con autómatas con pila	12
2.1.	Autómatas con pila deterministas	12
2.2.	Lenguajes libres del contexto no deterministas	16
3.	Autómatas con pila y gramáticas libres del contexto	18
4.	Limitaciones de los autómatas con pila y las GLC	20
5.	Clasificación de gramáticas, lenguajes y máquinas	21
6.	Preguntas de evaluación	23
6.1.	Problemas resueltos	23
6.2.	Problemas propuestos	27

1. Modelo de autómatas con pila

Así como las cadenas descritas por una expresión regular (cadenas de un lenguaje regular) pueden ser validadas mediante un autómata finito, las cadenas generadas por una gramática libre de contexto pueden ser aceptadas por un *autómata con pila*. Es decir, los lenguajes que pueden ser aceptados por autómatas con pila coinciden con los **lenguajes libres del contexto**, que son los generados por gramáticas libres del contexto. Un **autómata con pila**, abreviadamente AP, es en esencia un autómata finito aumentado con una *memoria tipo pila*, con lo cual, es un tipo de máquina teórica más potente que un AF. Cada estado en un AP representa una situación particular que se desea recordar o distinguir durante el procesamiento, como en un AF. Entre lo que se recuerda mediante los distintos estados y lo que queda guardado en la pila, se dispone de una información más completa que permite que los AP puedan validar cadenas que no son de formato regular. Por ejemplo, las cadenas del tipo $a^n b^n$ no forman un lenguaje regular y sin embargo pueden ser reconocidas por un AP.

En teoría, los autómatas con pila son máquinas sin salida, salvo devolver *true* (aceptar) o devolver *false* (rechazar). En la práctica, producen salidas o realizan acciones varias cuando se implementan para resolver problemas de procesamiento de cadenas cuya sintaxis puede describirse con una gramática libre del contexto. Los autómatas con pila, junto con las gramáticas libres del contexto, se usan fundamentalmente como **modelo para el diseño de algoritmos de análisis sintáctico de programas**. Estos algoritmos se estudiarán en la asignatura de Compiladores.

1.1. Funcionamiento de un AP a grandes rasgos

Una **configuración inicial** en un AP es una configuración de inicio de ejecución con una cadena de entrada w y matemáticamente se expresa mediante una tripleta (q_0, w, Z) , que indica que q_0 es estado inicial del autómata, w es la cadena de entrada que se quiere procesar y Z es el **símbolo inicial de la pila**. Este símbolo, que por defecto lo denotamos con Z , es lo que inicialmente hay guardado en la pila; por tanto, no se parte de una pila vacía. La pila tiene su propio **alfabeto de pila**, que denotamos con la letra griega Σ , formado por todos los símbolos que se pueden insertar en la pila, incluyendo el símbolo inicial Z . Algunos de estos símbolos pueden coincidir con los símbolos de **alfabeto de entrada** V , que es el que contiene los símbolos con los que se forman las cadenas válidas para el AP.

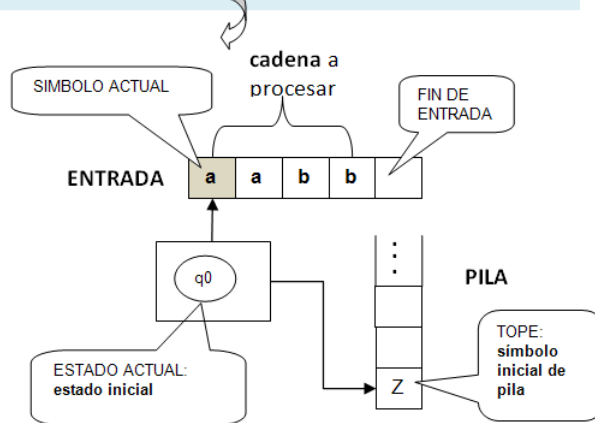
Ejemplo 1 En la siguiente figura mostramos la configuración inicial de cierto AP que procesa cadenas de a 's/ b 's ($V = \{a, b\}$) y está programado para aceptar como válidas aquellas cadenas de la forma $a^n b^n$. De momento no indicamos cuál es el "programa" del AP, es decir, la función de transición que permite realizar esa comprobación; eso lo vemos más adelante.

También se muestra una configuración intermedia que se alcanza tras varios pasos de ejecución, en los que el autómata ha insertado todas las a 's leídas en la pila. Se considera que el símbolo ' a ' es un símbolo del alfabeto de entrada V y también un símbolo del alfabeto de pila Σ .

Configuración inicial de un AP con cadena de entrada **aabb**.

Representación matemática: $(q_0, aabb, Z)$

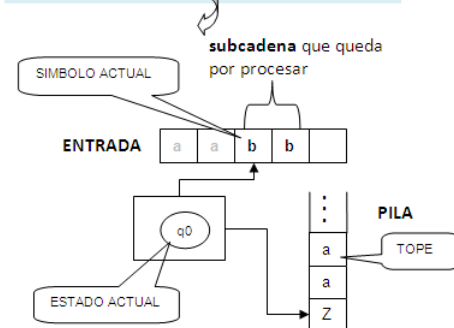
Representación gráfica:



Configuración intermedia tras leer las 'a's e insertarlas en la pila, dejando Z en el fondo.

Representación matemática: (q_0, bb, aaZ)

Representación gráfica:



En cada instante de ejecución el AP se encuentra en cierto estado (**estado actual**), ha leído cierto símbolo de la cadena (**símbolo actual**) y ha extraído (*pop*) el símbolo **tope** de la pila. Teniendo en cuenta esos tres parámetros se aplica la **regla de transición** correspondiente, que permite avanzar en la entrada, cambiar de estado e insertar (*push*) cierta cadena de símbolos en la pila. Por tanto, **los cambios de estado no sólo dependen del estado actual y del símbolo leído, sino también del símbolo tope de la pila**. El AP lleva a cabo un **bucle de procesamiento** donde va aplicando una transición en cada paso, leyendo la cadena de entrada símbolo a símbolo sin retroceder y guardando/extrayendo símbolos de la pila. La **ejecución de un AP finaliza** cuando ya no se pueden aplicar más transiciones, como en un AF.

Una **configuración** de un AP en cierto instante de ejecución viene dada, en notación matemática, por la tripleta (q, x, γ) donde:

- q es el estado actual del autómata.
- x es la porción de la entrada que le queda por procesar. Puede ser $x = \lambda$, en cuyo caso se ha terminado de leer la cadena, o bien, $x = ay$, donde a es el símbolo actual y $y \in V^*$ es el resto de la cadena de entrada.
- γ es una cadena que describe el contenido de la pila. Puede ser $\gamma = \lambda$, lo que indica que la pila está vacía, o bien, $\gamma = A_1 A_2 \dots A_m$, donde cada A_i es un símbolo del alfabeto de la pila (puede ser al mismo tiempo un símbolo del alfabeto de entrada). La pila tendría en este caso m símbolos y el símbolo más a la izquierda A_1 es el tope de la pila.

Condición de aceptación de una cadena en un AP

Hay varios tipos de autómatas con pila. Uno de ellos es el **tipo PV**, que se caracteriza porque no tiene estados finales y **las cadenas son aceptadas cuando se dan estas dos condiciones**:

1. Se ha leído toda la cadena de entrada (se detecta “fin de entrada”).
2. La pila está vacía.

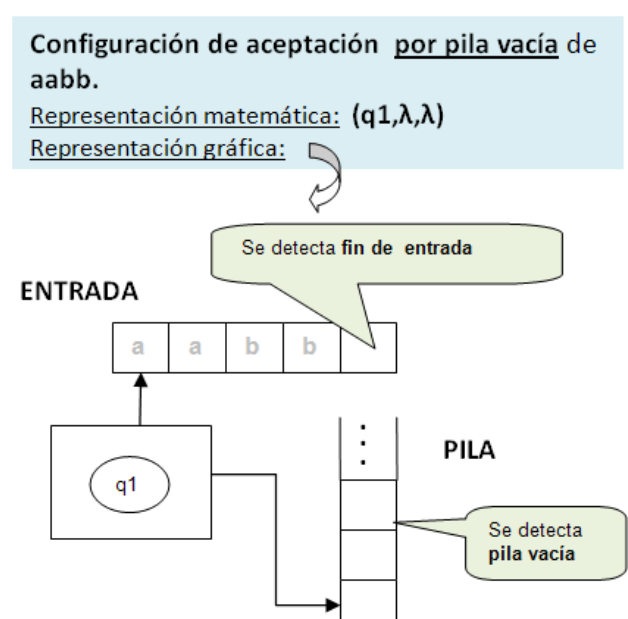
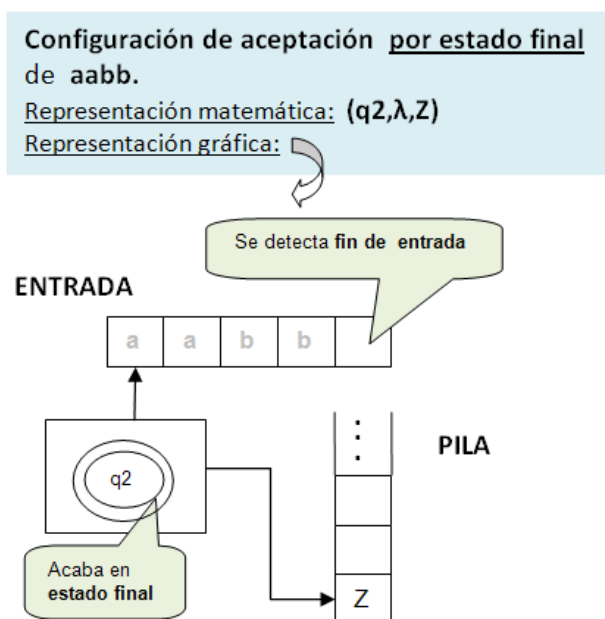
En un AP tipo PV, a una configuración de la forma (q, λ, λ) la llamaremos **configuración de aceptación**: q es un estado cualquiera del AP, λ en la segunda posición de la tripleta indica que no quedan símbolos de entrada por leer y λ en la tercera posición indica que la pila está vacía.

Otro tipo de autómatas con pila es el **tipo EF**, que tiene estados finales y **las cadenas son aceptadas cuando se dan estas dos condiciones**:

1. Se ha leído toda la cadena de entrada.
2. El autómata acaba en estado final.

La **configuración de aceptación** en un AP tipo EF es de la forma (q_F, λ, γ) : el autómata para en estado final q_F , λ en la segunda posición indica que no quedan símbolos de entrada por leer y γ en la tercera posición indica el contenido de la pila, que puede ser una cadena cualquiera de símbolos de la pila, incluso vacía. Eso quiere decir que el contenido de la pila al acabar la ejecución en este tipo de autómatas es irrelevante.

Ejemplo 2 Consideremos ahora que tenemos dos autómatas que aceptan cadenas del tipo $a^n b^n$, muy similares en funcionamiento. En ambos casos se supone que el autómata introduce todas las a 's en la pila y después extrae una a de la pila por cada b que se lee de la cadena de entrada. En el primer autómata, la cadena de entrada será válida si alcanza un estado final (acepta por estado final) y en el segundo la cadena será válida si se consigue vaciar la pila (acepta por pila vacía). En los siguientes diagramas aparecen las configuraciones de aceptación para la cadena $aabb$ en sendos autómatas.



1.2. Definición formal de un AP

Definición 1 Un AUTÓMATA CON PILA (*pushdown automaton, PDA*) es un modelo de máquina teórica que se define matemáticamente mediante una 7-upla $M = (Q, V, \Sigma, \delta, q_0, Z, F)$ donde M es el **nombre** del AP y los **componentes** son:

- Q es un conjunto finito de estados
- V es el alfabeto de entrada
- Σ es el alfabeto de pila
- q_0 es el estado inicial
- $Z \in \Sigma$ es el símbolo inicial de pila
- $F \subseteq Q$ es el conjunto de estados finales (puede ser vacío)
- $\delta : Q \times (V \cup \{\lambda\}) \times \Sigma \longrightarrow \mathcal{P}(Q \times \Sigma^*)$ es la función de transición

Un autómata con pila es, por defecto, **no determinista**. Esto significa que a partir de una configuración es posible que se puedan alcanzar **múltiples estados** o **múltiples configuraciones** en el instante siguiente, en un único paso de cálculo, lo cual supone que puede haber diferentes formas de procesar una cadena, o lo que es lo mismo, diferentes cálculos.

Tipos de reglas de transición y su representación gráfica

A cada caso de la función de transición δ de un AP lo llamamos **regla de transición** y, a partir del dominio y codominio de la función, podemos distinguir los siguientes tipos de reglas.

1. $\delta(q, a, A) = \{(q', \alpha)\}$ regla con lectura de símbolo, con o sin inserción en pila y una opción para cambio de estado. Por haber sólo una opción, puede simplificarse como: $\delta(q, a, A) = (q', \alpha)$.

SEMÁNTICA: esta regla puede aplicarse en una configuración tipo $(q, ay, A\beta)$, que indica que q es el estado actual, a es el símbolo actual, y es el resto de la cadena de entrada ($y \in V^*$), A es el tope de pila y β el contenido de la pila debajo del tope ($\beta \in \Sigma^*$).

Al **aplicar la regla de transición** $\delta(q, a, A) = (q', \alpha)$ se efectúa el siguiente **paso de cálculo**:

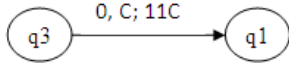
$$(q, ay, A\beta) \Rightarrow (q', y, \alpha\beta)$$

donde se cambia de estado q a q' , se avanza una posición en la entrada (lee/salta el símbolo $a \in V$), se extrae el tope $A \in \Sigma$ y se inserta la cadena $\alpha \in \Sigma^*$ en la pila. Si $\alpha = A_1A_2 \dots A_k$, donde cada $A_i \in \Sigma$, entonces se hace $push(A_k), \dots, push(A_2), push(A_1)$, quedando A_1 como tope. Si $\alpha = \lambda$ entonces no se insertan símbolos.

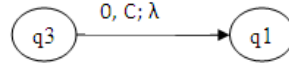
Ej.- Suponiendo que $V = \{0, 1\}$, $\Sigma = \{Z, C, 1\}$, tenemos que $\delta(q_3, 0, C) = (q_1, 11C)$ es una regla de este tipo. También lo es la regla de transición $\delta(q_3, 0, C) = (q_1, \lambda)$, que

indica que se extrae C y no se insertan símbolos en la pila. Estas reglas se representan gráficamente como:

Regla $\delta(q_3, 0, C) = (q_1, 11C)$



Regla $\delta(q_3, 0, C) = (q_1, \lambda)$



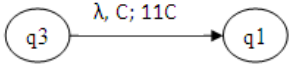
Si la configuración actual es $(q_3, 0101, C1C)$ entonces: 1) aplicando la regla $\delta(q_3, 0, C) = (q_1, 11C)$ se alcanza la configuración $(q_1, 101, 11C1C)$ y 2) aplicando la regla $\delta(q_3, 0, C) = (q_1, \lambda)$ se alcanza $(q_1, 101, 1C)$.

2. $\delta(q, \lambda, A) = (q', \alpha)$ sin lectura de símbolo (λ -transición), con o sin inserción en pila y una opción para cambio de estado.

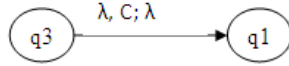
SEMÁNTICA: puede aplicarse en una configuración del tipo $(q, x, A\beta)$, con $x \in V^*$ y al aplicar la regla de transición se efectúa el paso de cálculo: $(q, x, A\alpha) \Rightarrow (q', x, \alpha\beta)$, donde se cambia de estado q a q' , no se avanza en la entrada, se extrae el tope A y se inserta la cadena α en la pila. Si $\alpha = \lambda$ entonces no se insertan símbolos.

Ej.- $\delta(q_3, \lambda, C) = (q_1, 11C)$ y $\delta(q_3, \lambda, C) = (q_1, \lambda)$ son reglas de transición de este tipo, que se representan gráficamente como:

Regla $\delta(q_3, \lambda, C) = (q_1, 11C)$



Regla $\delta(q_3, \lambda, C) = (q_1, \lambda)$



Si la configuración actual es $(q_3, 0101, C1C)$ entonces: 1) aplicando la regla $\delta(q_3, \lambda, C) = (q_1, 11C)$ se alcanza la configuración $(q_1, 0101, 11C1C)$ y 2) aplicando la regla $\delta(q_3, \lambda, C) = (q_1, \lambda)$ se alcanza $(q_1, 0101, 1C)$.

No determinismo: con una regla de transición del tipo $\delta(q, \lambda, A) = (q', \alpha)$, se puede producir una situación de no determinismo en una configuración como $(q, x, A\beta)$, si x comienza por cierto símbolo b ($x = by, y \in V^*$) y también hay definida una regla del tipo $\delta(q, b, A) = (q'', \alpha'')$, porque a partir de la misma configuración se pueden alcanzar configuraciones diferentes en un paso de cálculo:

$$(q, by, A\beta) \begin{cases} \Rightarrow (q', by, \alpha\beta) \text{ [cambia a } q', \text{ extrae } A \text{ y apila } \alpha, \text{ por } \delta(q, \lambda, A) = (q', \alpha)] \\ \Rightarrow (q'', y, \alpha''\beta) \text{ [como antes pero leyendo } b \text{ (avanza), por } \delta(q, b, A) = (q'', \alpha'')] \end{cases}$$

Ej.- Supongamos que tenemos las reglas $\delta(q_3, \lambda, C) = (q_1, 11C)$ y $\delta(q_3, 0, C) = (q_4, CC)$. Entonces tenemos dos posibles configuraciones siguientes a la configuración $(q_3, 0101, C1C)$:

$$(q_3, 0101, C1C) \begin{cases} \Rightarrow (q_1, 0101, 11C1C) \text{ [por } \delta(q_3, \lambda, C) = (q_1, 11C)] \\ \Rightarrow (q_4, 101, CC1C) \text{ [por } \delta(q_3, 0, C) = (q_4, CC)] \end{cases}$$

3. $\delta(q, a, A) = \{(q_{i1}, \alpha_1), (q_{i2}, \alpha_2), \dots, (q_{ik}, \alpha_k)\}$ regla con lectura de símbolo, con o sin inserción en pila y $k > 1$ opciones para cambio de estado.

SEMÁNTICA: puede aplicarse en una configuración del tipo $(q, ay, A\beta)$, con $y \in V^*$, y para aplicar la regla hay que elegir una de las k opciones para el cambio de estado. Suponiendo

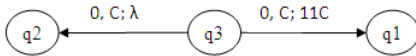
que se elige la opción $(q_{i_2}, \alpha_2) \in \delta(q, a, A)$, la regla se aplica como en el caso 1. Por tanto, con esa opción se efectúa el paso de cálculo: $(q, ay, A\beta) \Rightarrow (q_{i_2}, y, \alpha_2\beta)$.

No determinismo: con una regla del tipo $\delta(q, a, A) = \{(q_{i_1}, \alpha_1), (q_{i_2}, \alpha_2), \dots, (q_{i_k}, \alpha_k)\}$ se produce no determinismo en una configuración como $(q, ay, A\beta)$, porque las distintas opciones de la regla de transición conducen a configuraciones diferentes en un paso de cálculo:

$$(q, ay, A\beta) \begin{cases} \Rightarrow (q_{i_1}, y, \alpha_1\beta) & [\text{por opción } (q_{i_1}, \alpha_1) \in \delta(q, a, A)] \\ \Rightarrow (q_{i_2}, y, \alpha_2\beta) & [\text{por opción } (q_{i_2}, \alpha_2) \in \delta(q, a, A)] \\ \dots \\ \Rightarrow (q_{i_k}, y, \alpha_k\beta) & [\text{por opción } (q_{i_k}, \alpha_k) \in \delta(q, a, A)] \end{cases}$$

Ej.- $\delta(q_3, 0, C) = \{(q_1, 11C), (q_2, \lambda)\}$ es una regla de transición con lectura de símbolo y

múltiples opciones, que se representa como:



Si la **configuración actual** es $(q_3, 0101, C1C)$ entonces se pueden alcanzar dos configuraciones distintas:

$$(q_3, 0101, C1C) \begin{cases} \Rightarrow (q_2, 101, 1C) & [\text{por opción } (q_2, \lambda) \in \delta(q_3, 0, C)] \\ \Rightarrow (q_1, 101, 11C1C) & [\text{por opción } (q_1, 11C) \in \delta(q_3, 0, C)] \end{cases}$$

4. $\delta(q, \lambda, A) = \{(q_{i_1}, \alpha_1), (q_{i_2}, \alpha_2), \dots, (q_{i_k}, \alpha_k)\}$ λ -transición con o sin inserción en pila y $k > 1$ opciones para cambio de estado.

SEMÁNTICA: puede aplicarse en una configuración tipo $(q, x, A\alpha)$, y, como en el caso anterior, hay que elegir una de las k opciones para el cambio de estado. Suponiendo que se elige la opción $(q_{i_2}, \alpha_2) \in \delta(q, \lambda, A)$, la regla se aplica como en el caso 2. Por tanto, con esa opción se efectúa el paso de cálculo: $(q, x, A\alpha) \Rightarrow (q_{i_2}, x, \alpha_2\beta)$.

Con una regla de transición de este tipo también se produce **no determinismo** en una configuración como $(q, x, A\beta)$, de forma análoga a lo que ocurre con reglas del tipo 2.

Nota: al aplicar cualquier regla siempre se extrae el tope de la pila y en unos casos se insertan símbolos y en otros no. Puede darse la situación de que se puedan seguir aplicando reglas aunque se hayan leído todos los símbolos de entrada, debido a las λ -transiciones.

1.3. Lenguaje aceptado por un AP

Anteriormente hemos dicho que consideramos dos tipos de autómatas con pila (hay más):

1. AP tipo PV: no tienen estados finales ($F = \emptyset$) y **aceptan cadenas por pila vacía** (se detecta el fin de entrada y la pila vacía). Las configuraciones de aceptación son del tipo (q, λ, λ) , donde q es cualquier estado del AP.
2. AP tipo EF: tienen estados finales y **aceptan cadenas por estado final** (se detecta fin de entrada y para en estado final). Las configuraciones de aceptación son del tipo (q_F, λ, γ) , donde q_F es cualquier estado del AP y γ es el contenido de la pila.

Informalmente, el **lenguaje aceptado por un AP** es el conjunto de las cadenas que el AP puede aceptar. Formalmente, definimos el lenguaje aceptado por un AP dependiendo del tipo de AP como sigue a continuación.

Definición 2 Dado un autómata con pila $M = (Q, V, \Sigma, \delta, q_0, Z, \emptyset)$ tipo PV, el **lenguaje aceptado por pila vacía** por el autómata M es:

$$L_{PV}(M) = \{w \in V^* \mid (q_0, w, Z) \Rightarrow^* (q, \lambda, \lambda), \text{ donde } q \in Q\}$$

La condición $(q_0, w, Z) \Rightarrow^* (q, \lambda, \lambda)$ quiere decir que **existe un cálculo** que parte de la configuración inicial con w y acaba en configuración de aceptación, es decir, existe una secuencia de pasos de cálculos del tipo $C_0 \Rightarrow C_1 \Rightarrow \dots \Rightarrow C_n$, donde $C_0 = (q_0, w, Z)$ (configuración inicial) y $C_n = (q, \lambda, \lambda)$ (configuración de aceptación) y en cada paso se aplica una regla de transición. La secuencia de pasos de cálculo se corresponde con la **traza de ejecución** del autómata M con la cadena w .

Definición 3 Dado un autómata con pila $M = (Q, V, \Sigma, \delta, q_0, Z, F)$ tipo EF, el **lenguaje aceptado por estado final** por el autómata M es:

$$L_{EF}(M) = \{w \in V^* \mid (q_0, w, Z) \Rightarrow^* (q_F, \lambda, \gamma), \text{ donde } q_F \in F, \gamma \in \Sigma^*\}$$

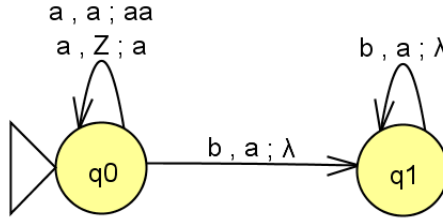
Nota: la **cadena vacía es aceptada** si a partir de la configuración inicial (q_0, λ, Z) se llega a una configuración de aceptación. Si el autómata acepta por estado final entonces es posible aceptar λ sin aplicar ninguna regla (cálculo en 0 pasos); esto pasa cuando q_0 es final, porque siempre se tiene que $(q_0, \lambda, Z) \Rightarrow^* (q_0, \lambda, Z)$ y si q_0 es final entonces la configuración (q_0, λ, Z) es de aceptación. Si el AP acepta por pila vacía entonces tendrá que aplicar al menos una regla para llegar a una configuración de aceptación.

Ejemplo 3 Sea el autómata con pila M_1 sin estados finales y función de transición dada por:

- 1 : $\delta(q_0, a, Z) = (q_0, a)$
- 2 : $\delta(q_0, a, a) = (q_0, aa)$
- 3 : $\delta(q_0, b, a) = (q_1, \lambda)$
- 4 : $\delta(q_1, b, a) = (q_1, \lambda)$

A partir de las transiciones podemos deducir el resto de componentes del autómata: $Q = \{q_0, q_1\}$; $V = \{a, b\}$, porque a y b son los únicos símbolos que se leen según las reglas; $\Sigma = \{a, Z\}$ porque esos son los únicos símbolos que se extraen o insertan en la pila. Por convenio Z es el símbolo inicial.

El **diagrama de transición** de este autómata lo podemos editar en Jflap, seleccionando la opción *Pushdown Automaton* del menú principal y en *Type of PDA input* seleccionamos *Multiple Character Input*. Una vez editado el diagrama, puede ejecutarse el autómata paso a paso con una cadena de entrada, eligiendo en *Input \rightarrow Step by State* el modo *Accepted by Empty Stack* (ver vídeo *tutorial6-AP-jflap*, cuyo enlace se encuentra en la carpeta *JFLAP* de recursos del aula virtual). El diagrama, que se encuentra guardado en un archivo *.jff* en *recursos \rightarrow JFLAP \rightarrow tema5-jflap.zip*, es el siguiente:



Vamos a **deducir el lenguaje** que acepta este autómata, se supone que por pila vacía, ya que se dice que no tiene estados finales. Para ello analizamos el sentido de cada regla de transición y ponemos algunos ejemplos de cálculos.

1. Inicialmente el tope de la pila es Z , por lo que la única regla aplicable desde cierta configuración inicial es la regla 1: $\delta(q_0, a, Z) = (q_0, a)$. De aquí deducimos que las cadenas que acepta deben comenzar por a . Al aplicar la regla se extrae Z y se introduce a en la pila.
2. La regla 2: $\delta(q_0, a, a) = (q_0, aa)$ permite que por cada a que se lea de la cadena de entrada se apile una a . Extraer una a de la pila e insertar dos a 's tiene el efecto de apilar la a leída.
3. La regla 3: $\delta(q_0, b, a) = (q_1, \lambda)$ es aplicable cuando se lee una b de la cadena de entrada y aparece a en el tope de la pila. Al aplicar la regla se extrae esa a y cambia al estado q_1 .
4. La regla 4: $\delta(q_1, b, a) = (q_1, \lambda)$ permite que por cada b que se lea de la cadena de entrada se extraiga una a de la pila. Observamos que a partir de q_1 ya no se pueden leer más a 's, por tanto deducimos ahora que la cadenas aceptadas sólo deben tener b 's tras las a 's.

La pila se vacía en el estado q_1 sólo cuando se han leído el mismo número de a 's que de b 's. Por todo esto deducimos que:

$$L_{PV}(M_1) = \{a^n b^n \mid n > 0\}$$

Ej.- La cadena de entrada $aabb$ es aceptada. La configuración inicial es $(q_0, aabb, Z)$ y la configuración de aceptación es (q_1, λ, λ) , que gráficamente se representa como en la figura derecha mostrada en el ejemplo 2. El cálculo o traza de ejecución con $aabb$ es:

$$(q_0, aabb, Z) \xRightarrow{Tr1} (q_0, abb, a) \xRightarrow{Tr2} (q_0, bb, aa) \xRightarrow{Tr3} (q_1, b, a) \xRightarrow{Tr4} (q_1, \lambda, \lambda)$$

Encima del símbolo de paso de cálculo \Rightarrow se indica la transición aplicada.

Ej.- La cadena abb no es aceptada (tiene más b 's que a 's). La configuración de parada es (q_1, b, λ) , que no es de aceptación porque, aunque se ha vaciado la pila, no se ha leído toda la cadena. El cálculo con abb es:

$$(q_0, abb, Z) \Rightarrow (q_0, bb, a) \Rightarrow (q_1, b, \lambda)$$

Ej.- La cadena aab no es aceptada (tiene más a 's que b 's). La configuración de parada es (q_1, λ, a) , que no es de aceptación porque, aunque se ha leído toda la cadena, no acaba con la pila vacía. El cálculo con aab es:

$$(q_0, aab, Z) \Rightarrow (q_0, ab, a) \Rightarrow (q_0, b, aa) \Rightarrow (q_1, \lambda, a)$$

Ej.- La cadena λ no es aceptada porque a partir de la configuración inicial (q_0, λ, Z) no se puede aplicar ninguna regla y, por tanto, no puede vaciarse la pila.

Teorema 1 Si un lenguaje es aceptado por estado final por un autómata con pila M_f entonces también puede ser aceptado por pila vacía por otro autómata M_v y al contrario.

En este sentido podemos decir que **aceptar por pila vacía es equivalente a aceptar por estado final** y, dependiendo del lenguaje, en ocasiones es más sencillo o conveniente hacerlo de una manera que de otra.

Justificación.- Si tenemos un autómata M_v que acepta cierto lenguaje por pila vacía entonces puede transformarse en otro M_f que acepta el mismo lenguaje por estado final. La idea es dejar en M_f el símbolo inicial Z en el fondo de la pila y se añaden transiciones desde aquellos estados de M_v en los que se vaciaba la pila hasta un nuevo estado final sin transiciones. Ej.- si la pila se vaciaba en M_v en el estado q_3 entonces se añade en M_f la regla $\delta(q_3, \lambda, Z) = (q_F, \lambda)$.

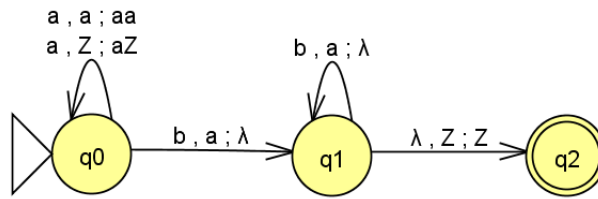
De igual forma, si tenemos un autómata M_f que acepta cierto lenguaje por estado final entonces puede transformarse en otro M_v que acepte el mismo lenguaje por pila vacía. La idea es añadir transiciones en M_v que permitan vaciar la pila cuando se alcanza un estado final en M_f .

Ejemplo 4 Ahora queremos obtener un AP que acepte cadenas del tipo $a^n b^n, n > 0$ por estado final. Podemos partir de M_1 del ejemplo 3 y modificarlo para obtener M_2 que acepte las mismas cadenas por estado final. Indicamos la función de transición y el diagrama correspondiente.

Función de transición

- 1 : $\delta(q_0, a, Z) = (q_0, aZ)$
- 2 : $\delta(q_0, a, a) = (q_0, aa)$
- 3 : $\delta(q_0, b, a) = (q_1, \lambda)$
- 4 : $\delta(q_1, b, a) = (q_1, \lambda)$
- 5 : $\delta(q_1, \lambda, Z) = (q_2, Z)$

Diagrama de transición



Los cambios con respecto a M_1 son los siguientes:

1. En la regla 1: $\delta(q_0, a, Z) = (q_0, aZ)$ se ha dejado el símbolo inicial Z en la pila, a modo de marcador de fin de pila. Se deja porque luego se usa en la transición 5 para detectar cuándo se ha sacado todos los símbolos y sólo queda Z .
2. Se incluye la transición 5 para permitir acabar en estado final y sólo será aplicable en el caso que hayan extraído todas las a 's.

Es fácil darse cuenta de que M_2 acepta una cadena, acabando en q_2 , si y sólo si M_1 la acepta acabando con la pila vacía. Por eso $L_{EF}(M_2) = L_{PV}(M_1) = \{a^n b^n \mid n > 0\}$

Ej.- La cadena de entrada aabb es aceptada y la configuración de aceptación es (q_2, λ, Z) , que gráficamente se representa como en la figura izquierda mostrada en el ejemplo 2. El cálculo con aabb es:

$$(q_0, aabb, Z) \xRightarrow{Tr1} (q_0, abb, aZ) \xRightarrow{Tr2} (q_0, bb, aaZ) \xRightarrow{Tr3} (q_1, b, aZ) \xRightarrow{Tr4} (q_1, \lambda, Z) \xRightarrow{Tr5} (q_2, \lambda, Z)$$

Ej.- La cadena abb no es aceptada. La configuración de parada es (q_1, b, Z) , que no es de aceptación porque ni se ha leído toda la cadena ni ha acabado en estado final. El cálculo con abb es:

$$(q_0, abb, Z) \Rightarrow (q_0, bb, aZ) \Rightarrow (q_1, b, Z)$$

Ej.- La cadena aab no es aceptada. La configuración de parada es (q_1, λ, aZ) , que no es de aceptación porque no acaba en estado final. El cálculo con aab es:

$$(q_0, aab, Z) \Rightarrow (q_0, ab, aZ) \Rightarrow (q_0, b, aaZ) \Rightarrow (q_1, \lambda, aZ)$$

Ej.- La cadena λ no es aceptada porque a partir de la configuración inicial (q_0, λ, Z) no se puede aplicar ninguna regla y, por tanto, no puede alcanzarse el estado final q_2 .

2. Diseño con autómatas con pila

Para resolver un problema de validación o análisis de cadenas, que pueden ser procesadas leyéndolas símbolo a símbolo sin retroceder y usando únicamente una memoria tipo pila, conviene hacer un diseño inicial del algoritmo que resuelve el problema tomando como modelo un autómata con pila, simulándolo. Una vez obtenido el *AP* hay que asegurarse de que el **AP es correcto** y para ello debe cumplir dos condiciones:

1. El AP no es demasiado estricto, en el sentido de que acepta todas las cadenas consideradas válidas.
2. El AP no es demasiado general, en el sentido de que rechaza cualquier cadena que no se considere correcta.

Para ello conviene hacer un test de prueba que cubra todos los casos, para comprobar que el lenguaje aceptado por el autómata coincide con el lenguaje dado en la especificación del problema.

2.1. Autómatas con pila deterministas

Para simular el funcionamiento de un AP, con objeto de diseñar un *algoritmo de validación* o *análisis sintáctico* de cadenas de lenguajes libres del contexto, conviene que el autómata con pila sea determinista, por cuestiones de eficiencia. **El algoritmo obtenido a partir de un**

AP determinista tiene un tiempo de ejecución de orden lineal: $O(n)$, donde n es la longitud de la cadena de entrada, que es muy eficiente.

En la sección 1.2 vimos que algunos tipos de reglas de transición pueden dar lugar a una situación de no determinismo, en la que hay que elegir entre varias reglas o entre varias opciones de una regla de transición.

Definición 4 Un AP es **determinista** si no es posible que a partir de una configuración se puedan alcanzar dos o más configuraciones distintas en un paso de cálculo. En otro caso decimos que el AP es **no determinista**.

Para que un AP sea determinista debe cumplir dos condiciones:

1. No tiene reglas de transición de múltiples opciones, tipo $\delta(q, a, A) = \{(q_{i1}, \alpha_1), \dots, (q_{ik}, \alpha_k)\}$, o bien, $\delta(q, \lambda, A) = \{(q_{i1}, \alpha_1), \dots, (q_{ik}, \alpha_k)\}$.
2. No hay definida simultáneamente una transición con un símbolo $a \in V$ y con λ , para un mismo estado q y tope de pila $A \in \Sigma$. Es decir, si $\delta(q, a, A) \neq \emptyset$ entonces debe ser $\delta(q, \lambda, A) = \emptyset$ y al contrario, si $\delta(q, \lambda, A) \neq \emptyset$ entonces debe ser $\delta(q, a, A) = \emptyset$, para cualquier $a \in V$.

De no cumplirse alguna de las condiciones anteriores el AP será no determinista. Ej.- los autómatas vistos hasta ahora en los ejemplos 3 y 4 son deterministas.

Ejemplo 5 Queremos obtener un **AP determinista** que acepte el lenguaje $\{a^n b^n \mid n \geq 0\}$. Es una variación del lenguaje de los ejemplos 3 y 4, donde se requería aceptar cadenas del tipo $a^n b^n$, pero con $n > 0$. Habría que modificar alguno de los AP de esos ejemplos para que acepten también la cadena vacía (caso $n = 0$). Tenemos que pensar si es mejor diseñar un AP que acepte el lenguaje $\{a^n b^n \mid n \geq 0\}$ por estado final o por pila vacía.

Diseñamos un AP M_{v0} tipo PV que acepta el lenguaje requerido por pila vacía y otro AP M_{f0} tipo EF que acepta por estado final, a partir los autómatas de los ejemplos 3 y 4, respectivamente:

M_{v0} con $F = \emptyset$	M_{f0} con $F = \{q_2\}$
0 : $\delta(q_0, \lambda, Z) = (q_1, \lambda)$	0 : $\delta(q_0, \lambda, Z) = (q_2, Z)$
1 : $\delta(q_0, a, Z) = (q_0, a)$	1 : $\delta(q_0, a, Z) = (q_0, aZ)$
2 : $\delta(q_0, a, a) = (q_0, aa)$	2 : $\delta(q_0, a, a) = (q_0, aa)$
3 : $\delta(q_0, b, a) = (q_1, \lambda)$	3 : $\delta(q_0, b, a) = (q_1, \lambda)$
4 : $\delta(q_1, b, a) = (q_1, \lambda)$	4 : $\delta(q_1, b, a) = (q_1, \lambda)$
	5 : $\delta(q_1, \lambda, Z) = (q_2, Z)$

En ambos autómatas se ha añadido la regla de transición 0 para que λ sea aceptada. En M_{v0} se acepta por el cálculo: $(q_0, \lambda, Z) \Rightarrow (q_1, \lambda, \lambda)$, que deja la pila vacía, y en M_{f0} se acepta por el cálculo: $(q_0, \lambda, Z) \Rightarrow (q_2, \lambda, Z)$, que acaba en estado final. Sin embargo, ambos AP son no deterministas, porque no se cumple la condición 2, lo que origina dos configuraciones distintas a partir de cierta configuración inicial donde sean aplicables las reglas 0 y 1 al mismo

tiempo. Por ejemplo, a partir de $(q_0, aabb, Z)$ se pueden alcanzar dos configuraciones distintas en M_{v0} :

$$(q_0, aabb, Z) \begin{cases} \Rightarrow (q_1, aabb, \lambda) & \text{por } \delta(q_0, \lambda, Z) = (q_1, \lambda) \\ \Rightarrow (q_0, abb, a) & \text{por } \delta(q_0, a, Z) = (q_0, a) \end{cases}$$

A partir de $(q_0, aabb, Z)$ también se pueden alcanzar dos configuraciones distintas en M_{f0} :

$$(q_0, aabb, Z) \begin{cases} \Rightarrow (q_2, aabb, Z) & \text{por } \delta(q_0, \lambda, Z) = (q_2, Z) \\ \Rightarrow (q_0, abb, aZ) & \text{por } \delta(q_0, a, Z) = (q_0, aZ) \end{cases}$$

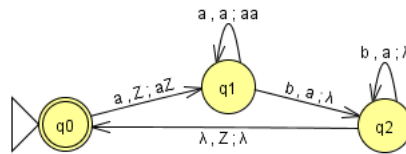
La cadena $aabb$ es aceptada por ambos autómatas porque existe al menos un cálculo que conduce a una configuración de aceptación, aunque haya otros por los que no se acepta. A partir de la configuración inicial $(q_0, aabb, Z)$ no debe aplicarse la regla 0 porque el autómata no para en configuración de aceptación.

Para aceptar λ por pila vacía es necesario incluir una regla como $\delta(q_0, \lambda, Z) = (q_1, \lambda)$ que permita vaciar la pila, con lo cual no se puede evitar que el AP sea no determinista. Sin embargo, para aceptar λ por estado final, podemos hacer que **el estado inicial sea un estado final** y de ese modo no es necesario aplicar ninguna regla de transición, porque la configuración (q_0, λ, Z) es a la vez configuración inicial y de aceptación. Siguiendo esta idea podemos modificar el autómata M_{f0} para que sea $F = \{q_0\}$ en lugar de que el estado final sea q_2 . El AP **determinista** M_{def0} que acepta el lenguaje $\{a^n b^n \mid n \geq 0\}$ por estado final es:

Función de transición

1. $\delta(q_0, a, Z) = (q_1, aZ)$
2. $\delta(q_1, a, a) = (q_1, aa)$
3. $\delta(q_1, b, a) = (q_2, \lambda)$
4. $\delta(q_2, b, a) = (q_2, \lambda)$
5. $\delta(q_2, \lambda, Z) = (q_0, \lambda)$

Diagrama de transición:



- La cadena $\lambda = a^0 b^0$ es aceptada por el cálculo en cero pasos: $(q_0, \lambda, Z) \Rightarrow^* (q_0, \lambda, Z)$.

- Con la transición 1 cambiamos al estado q_1 porque siguiendo en q_0 podrían aceptarse cadenas que no son del lenguaje, como a, aa, aaa , etc.

- Las cadenas del tipo $a^n b^n$, con $n > 0$ se aceptan insertando las a 's y extrayendo luego una a por cada b que se lee. Si la cadena es correcta aparecerá Z en la configuración (q_2, λ, Z) , en la cual se aplica finalmente la transición 5 para pasar a estado final y aceptar. Esto se refleja en el cálculo genérico:

$$(q_0, a^n b^n, Z) \Rightarrow (q_1, a^{n-1} b^n, aZ) \Rightarrow^* (q_1, b^n, a^n Z) \Rightarrow^* (q_2, \lambda, Z) \Rightarrow (q_0, \lambda, \lambda)$$

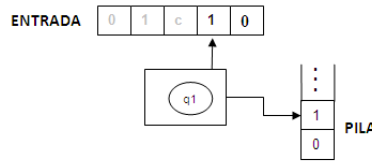
Nota: sería un error poner en última transición $\delta(q_2, \lambda, Z) = (q_0, Z)$ en lugar de $\delta(q_2, \lambda, Z) = (q_0, \lambda)$, porque en ese caso se podría usar la transición 5, después de haber leído una subcadena del tipo $a^n b^n$, para volver al estado inicial y comenzar de nuevo a leer otra subcadena de ese tipo. Por tanto, si se hace ese cambio aceptaría el lenguaje $\{a^n b^n \mid n \geq 0\}^*$. Como ejemplo podemos comprobar que haciendo en cambio indicado en la transición 5 aceptaría la cadena $aaabbbbaabb$, porque:

$$(q_0, aaabbbbaabb) \Rightarrow^* (q_2, aabb, Z) \Rightarrow (q_0, aabb, Z) \Rightarrow^* (q_2, \lambda, Z) \Rightarrow (q_0, \lambda, Z)$$

Ejemplo 6 Vamos a diseñar un AP, llamado M_{wcwr} , que acepte el lenguaje $L_{wcwr} = \{wcw^R \mid w \in \{0,1\}^*\}$ por pila vacía. La **idea para diseñar el autómata** va a ser:

- Al leer el primer símbolo 0/1 se apila, pero se extrae Z de la pila, ya no se necesita para controlar cuándo se han extraído todos los símbolos apilados. Si el primer símbolo que se lee es c se extrae Z pero no se inserta nada. Lo anterior se consigue con las reglas de transición 1-3 indicadas más adelante.
- Mientras no se lea una c en la entrada, vamos introduciendo los símbolos leídos en la pila (transiciones 4 a 7).
- Cuando se lee una c cambiamos de estado, para comenzar a comparar la cadena de la pila con lo que queda de la entrada (tr. 8 y 9; ver ej. de configuración más abajo).
- Por la forma de introducir símbolos en la pila, si antes de la c se lee por ejemplo 001, entonces después de leer la c la cadena de la pila será $100 = (001)^R$. Luego, la cadena de entrada será aceptada si y sólo si la subcadena que queda por leer coincide exactamente con la cadena de la pila.
Para hacer esta última comprobación se extrae un símbolo de la pila por cada símbolo que se lea si ambos coinciden (reglas de transición 10 y 11).
- Si la cadena de entrada es correcta entonces el autómata conseguirá vaciar la pila, en otro caso no se vacía la pila o no se termina de leer la cadena de entrada.

Ej.- si la entrada es 01c10 entonces la configuración del AP tras haber leído la c sería:



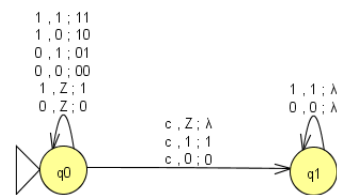
El autómata que necesitamos según la idea del diseño es:

$$M_{wcwr} = (\{q_0, q_1\}, \{0, 1, c\}, \{0, 1, Z\}, \delta, q_0, Z, \emptyset)$$

Función de transición

- | | |
|---|--|
| 1. $\delta(q_0, 0, Z) = (q_0, 0)$ | 7. $\delta(q_0, 1, 1) = (q_0, 11)$ |
| 2. $\delta(q_0, 1, Z) = (q_0, 1)$ | 8. $\delta(q_0, c, 0) = (q_1, 0)$ |
| 3. $\delta(q_0, c, Z) = (q_1, \lambda)$ | 9. $\delta(q_0, c, 1) = (q_1, 1)$ |
| 4. $\delta(q_0, 0, 0) = (q_0, 00)$ | 10. $\delta(q_1, 0, 0) = (q_1, \lambda)$ |
| 5. $\delta(q_0, 1, 0) = (q_0, 10)$ | 11. $\delta(q_1, 1, 1) = (q_1, \lambda)$ |
| 6. $\delta(q_0, 0, 1) = (q_0, 01)$ | |

Diagrama de transición



Test de prueba:

1. Cadenas válidas (pertenecen a L_{wcwr}). Probamos con la cadena de menor longitud, que es c , y con otra que tenga 0s/1s, por ejemplo 01c10. Estas cadenas son aceptadas por M_{wcwr}

como muestran los siguientes cálculos:

$$\begin{array}{c}
(q_0, c, Z) \xRightarrow{Tr3} (q_1, \lambda, \lambda) \text{ [acepta } c] \\
(q_0, 01c10, Z) \xRightarrow{Tr1} (q_0, 1c10, 0) \xRightarrow{Tr5} (q_0, c10, 10) \xRightarrow{Tr9} (q_1, 10, 10) \xRightarrow{Tr11} (q_1, 0, 0) \\
\xRightarrow{Tr10} (q_1, \lambda, \lambda) \text{ [acepta } 01c10]
\end{array}$$

2. Cadenas incorrectas (no pertenecen a L_{wcvr}). La cadena vacía no es aceptada porque no hay transición del tipo $\delta(q_0, \lambda, Z)$, por lo que no se puede vaciar la pila. Probamos casos de cadenas no válidas que no son aceptadas por distintos motivos. Por ejemplo, la cadena 01c01 no es aceptada porque el AP para sin vaciar la pila y sin terminar de leer la entrada:

$$(q_0, 01c01, Z) \Rightarrow (q_0, 1c10, 0) \Rightarrow (q_0, c01, 10) \Rightarrow (q_1, 01, 10)$$

Otro caso es la cadena 10c011, que no es aceptada porque el AP para vaciando la pila pero sin terminar de leer la entrada:

$$(q_0, 10c011, Z) \Rightarrow (q_0, 0c011, 1) \Rightarrow (q_0, c011, 01) \Rightarrow (q_1, 011, 01) \Rightarrow (q_1, 11, 1) \Rightarrow (q_1, 1, \lambda)$$

Puede probarse que con la cadena 01c1 termina de leer pero no vacía la pila.

Una vez probados casos concretos, mostramos un **cálculo genérico** para una cadena del tipo wcw^R arbitraria, que refleja los cambios de estado y cambios en la pila. Observamos que la pila siempre se vacía en el estado q_1 :

$$(q_0, wcw^R, Z) \Rightarrow^* (q_0, cw^R, w^R) \Rightarrow (q_1, w^R, w^R) \Rightarrow^* (q_1, \lambda, \lambda)$$

Por el razonamiento dado sobre el diseño y funcionamiento del AP podemos decir que M_{wcvr} es **correcto** para el lenguaje L_{wcvr} , es decir, se cumple que $L_{PV}(M_{wcvr}) = L_{wcvr}$.

2.2. Lenguajes libres del contexto no deterministas

Sería deseable que para cualquier autómata con pila pudiéramos encontrar un autómata con pila determinista equivalente, ya que esto haría que la simulación de un AP en un programa fuera muy eficiente. Sin embargo, **no todos los lenguajes libres del contexto pueden ser aceptados por autómatas con pila deterministas**. En este sentido, a diferencia de lo que ocurría con los AF, **los AP deterministas y no deterministas no son equivalentes**.

Definición 5 Decimos que un lenguaje es **libre del contexto determinista** si existe un AP determinista que lo acepta (da igual si es por pila vacía o por estado final). Si no existe un AP determinista pero sí un AP no determinista que lo acepta entonces el lenguaje es **libre del contexto no determinista**.

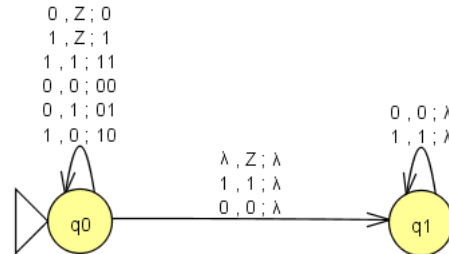
Ejemplo 7 Un ejemplo de lenguaje libre del contexto no determinista es el lenguaje de palíndromos binarios de longitud par: $L_{wvr} = \{ww^R \mid w \in \{0,1\}^*\}$. Este lenguaje es similar al del ejemplo 6, pero en este caso no tenemos un símbolo que marque el centro de la cadena y, por

tanto, a priori no podemos saber cuándo se acaba de leer la primera mitad de la cadena de entrada. Es necesario incluir transiciones con dos opciones para la situación en que coincida el símbolo de la entrada con el tope de pila. La función de transición de un AP tipo PV que acepta el lenguaje es la siguiente:

Función de transición:

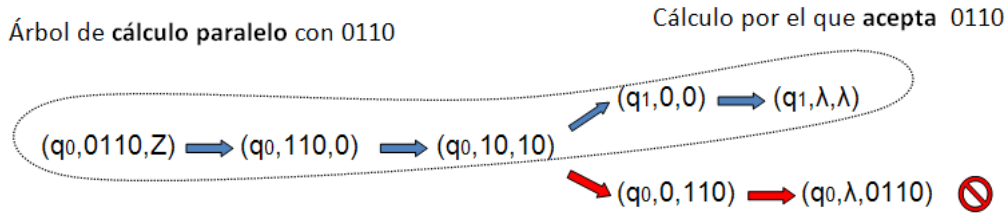
1. $\delta(q_0, \lambda, Z) = (q_1, \lambda)$
2. $\delta(q_0, 0, Z) = (q_0, 0)$
3. $\delta(q_0, 1, Z) = (q_0, 1)$
4. $\delta(q_0, 0, 0) = \{(q_0, 00), (q_1, \lambda)\}$
5. $\delta(q_0, 1, 0) = (q_0, 10)$
6. $\delta(q_0, 0, 1) = (q_0, 01)$
7. $\delta(q_0, 1, 1) = \{(q_0, 11), (q_1, \lambda)\}$
8. $\delta(q_1, 0, 0) = (q_1, \lambda)$
9. $\delta(q_1, 1, 1) = (q_1, \lambda)$

Diagrama de transición:



Independientemente del no determinismo que se produce por la transición 1 junto con la 2 ó 3, que digamos es evitable porque la 1 sólo se incluye para aceptar λ , no hay manera de evitar el no determinismo por las transiciones 4 ó 7, que son necesarias para dar la posibilidad de pasar a extraer símbolos de la pila tras la primera mitad de la cadena. Se puede obtener otro autómata diferente para aceptar el lenguaje L_{wtr} pero seguirá siendo no determinista. Por eso este lenguaje es **libre del contexto no determinista**.

En la siguiente figura tenemos un **árbol de cálculo paralelo** con la cadena de entrada 0110, donde se muestran los dos posibles cálculos, uno en cada rama del árbol, que se generan como consecuencia de la aplicación de las dos opciones de la regla de transición no determinista $\delta(q_0, 1, 1) = \{(q_0, 11), (q_1, \lambda)\}$. La cadena 0110 es aceptada porque por uno de los cálculos se llega a la configuración de aceptación.



Para **simular el funcionamiento de un AP no determinista** mediante un algoritmo es necesario explorar las distintas ramas del árbol de cálculo paralelo para una cadena de entrada, lo cual supone tener que desarrollar un **algoritmo con backtracking**. Este algoritmo recorre el árbol en profundidad, eligiendo una de las transiciones aplicables en cada paso. Si el camino seguido no lleva a aceptar la cadena entonces deshace los cambios hechos en la configuración actual y retrocede hasta la configuración anterior para elegir otra opción. Así hasta llegar a una configuración de aceptación y aceptar la cadena, o rechazar por agotar todas las posibilidades. El número de configuraciones que se generan en esta simulación puede ser en el peor de los casos de orden $O(2^n)$, donde n es la longitud de la cadena de entrada, lo cual supone un **tiempo exponencial en la simulación**. Por eso es mejor un AP determinista, siempre que sea posible.

3. Autómatas con pila y gramáticas libres del contexto

En esta sección vamos a ver que dada una gramática libre del contexto G existe un autómata con pila M que acepta el mismo lenguaje que genera G y al contrario, dado un autómata con pila M puede obtenerse una GLC G que genera el mismo lenguaje que acepta M . Esto se traduce en el siguiente teorema:

Teorema 2 (equivalencia AP-GLC) *Un lenguaje puede ser aceptado por un autómata con pila si y sólo si puede ser generado por una gramática libre del contexto.*

Este teorema implica que los autómatas con pila y las gramáticas libres del contexto son formalismos matemáticos con la misma expresividad, pues ambos tratan con la clase de **lenguajes libres del contexto**, aunque con distinto objetivo: las gramáticas describen la estructura de las cadenas del lenguaje (la sintaxis) y los autómatas con pila describen el procedimiento para comprobar si una cadena es correcta (pertenece al lenguaje).

La demostración de este teorema no la vamos a ver completa, sólo vamos a ver la parte que permite pasar de una GLC a un AP. Hay varios métodos algorítmicos para hacer esto y aquí presentamos uno de ellos, que llamamos *método de construcción de AP no determinista*. Este método consiste básicamente en obtener un AP no determinista, que acepta cadenas **simulando derivaciones más a la izquierda** en la gramática de partida. Hacemos que el AP acepte por pila vacía, aunque se puede modificar el método fácilmente para obtener un AP que acepte por estado final.

Método GLCtoAP no determinista

ENTRADA: una GLC $G = (V_N, V_T, S, P)$.

SALIDA: un AP M tal que $L_{PV}(M) = L(G)$.

1. Se incluyen dos **estados**: $Q := \{q_0, q_1\}$ y no hay estados finales;
2. El **alfabeto de entrada** V coincide con el alfabeto de símbolos terminales de G : $V = V_T$;
3. El **alfabeto de pila** Σ está formado por todos los símbolos de la gramática más el símbolo inicial de pila: $\Sigma := V_N \cup V_T \cup \{Z\}$.
4. La **función de transición** δ se obtiene del siguiente modo:

- $\delta(q_0, \lambda, Z) = (q_1, S)$ [introduce símbolo inicial S de la gramática en pila]
- $\delta(q_1, a, a) = (q_1, \lambda)$, $\forall a \in V_T$ [cuando coinciden terminales extrae de pila]
- Si $A \rightarrow \alpha \in P$ entonces $(q_1, \alpha) \in \delta(q_1, \lambda, A)$ [simula aplicación de regla]

Ejemplo 8 *Sea una gramática no ambigua que genera expresiones aritméticas. El terminal i representa un operando.*

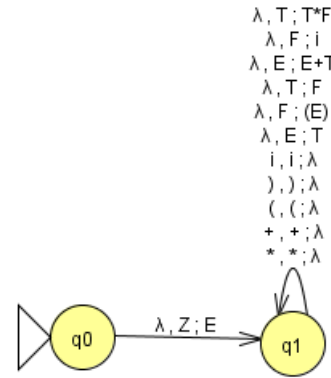
$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

El AP correspondiente a esta gramática se indica más abajo. La transición 0 tiene por objeto insertar el símbolo inicial de la gramática en la pila. Las reglas de transición 1-3 son las que se obtienen a partir de las reglas de producción de la gramática. Sirven para ‘simular’ la aplicación de una regla de producción: sustituye la parte izquierda de una regla de producción en la pila por la parte derecha de la regla. Las transiciones 4-8 se incluyen para avanzar en la entrada cuando coincida el símbolo de la entrada con el tope de la pila. Con cierta cadena de entrada w , el autómata conseguirá vaciar la pila si y sólo si la cadena w es derivable del símbolo inicial. Por eso este autómata acepta el lenguaje generado por la gramática.

Función de transición:

0. $\delta(q_0, \lambda, Z) = (q_1, E)$
1. $\delta(q_1, \lambda, E) = \{(q_1, E + T), (q_1, T)\}$
2. $\delta(q_1, \lambda, T) = \{(q_1, T * F), (q_1, F)\}$
3. $\delta(q_1, \lambda, F) = \{(q_1, (E)), (q_1, i)\}$
4. $\delta(q_1, (, () = (q_1, \lambda)$
5. $\delta(q_1,),) = (q_1, \lambda)$
6. $\delta(q_1, i, i) = (q_1, \lambda)$
7. $\delta(q_1, +, +) = (q_1, \lambda)$
8. $\delta(q_1, *, *) = (q_1, \lambda)$

Diagrama de transición:



Consideremos la sentencia $i + i$. La **derivación más a la izquierda** para esta sentencia es:

$$E \xRightarrow{E \rightarrow E+T} [E] + T \xRightarrow{E \rightarrow T} [T] + T \xRightarrow{T \rightarrow F} [F] + T \xRightarrow{F \rightarrow i} i + [T] \xRightarrow{T \rightarrow F} i + [F] \xRightarrow{F \rightarrow i} i + i$$

Si tenemos en cuenta la secuencia de reglas que se aplican en cada paso podemos encontrar un **cálculo por el que se acepta la cadena** $i + i$, ya que los cálculos del autómata simulan la derivación más a la izquierda de la cadena. Encima del símbolo \Rightarrow de paso de cálculo indicamos la transición (Tr) que se aplica. Algunos pasos de cálculo se corresponden con la aplicación de una regla de la gramática. Ej.- con Tr1,2; $E \rightarrow T$ indicamos que se aplica la opción 2 de la transición 1, que se obtiene a partir de la regla de producción $E \rightarrow T$.

$$\begin{aligned} (q_0, i + i, Z) &\xRightarrow{\text{Tr0}} (q_1, i + i, E) \xRightarrow{\text{Tr1,1; } E \rightarrow E+T} (q_1, i + i, E + T) \xRightarrow{\text{Tr1,2; } E \rightarrow T} (q_1, i + i, T + T) \xRightarrow{\text{Tr2,2; } T \rightarrow F} \\ &(q_1, i + i, F + T) \xRightarrow{\text{Tr3,2; } F \rightarrow i} (q_1, i + i, i + T) \xRightarrow{\text{Tr6}} (q_1, i + i, +T) \xRightarrow{\text{Tr7}} (q_1, i, T) \xRightarrow{\text{Tr2,2; } T \rightarrow F} (q_1, i, F) \\ &\xRightarrow{\text{Tr3,2; } F \rightarrow i} (q_1, i, i) \xRightarrow{\text{Tr6}} (q_1, \lambda, \lambda) \quad \underline{\text{acepta}} \end{aligned}$$

Como vemos, el AP resulta **no determinista** porque hay varias opciones para aplicar una transición, igual que hay varias opciones para aplicar una regla gramatical. Por ejemplo, en la configuración $(q_1, i + i, E)$ hay que aplicar la transición número 1: $\delta(q_1, \lambda, E) = \{(q_1, E + T), (q_1, T)\}$. Pero esta transición tiene dos opciones y en el cálculo anterior se ha escogido la

opción 1, que extrae E e inserta $E + T$. Si se hubiera aplicado la otra opción de la transición entonces el cálculo no llegaría a acabar en configuración de aceptación.

El **gran inconveniente del método GLCtoAP** no determinista es que al implementarlo habría que desarrollar un algoritmo de análisis sintáctico con backtracking, o por fuerza bruta (*brute force parser*). En cada paso se elige una de las transiciones posibles y si se llega a una configuración que no conduce a aceptar la cadena entonces hay que retroceder deshaciendo la aplicación de la regla de transición y volver a aplicar otra transición. Así hasta llegar a aceptar la cadena o darla por incorrecta si se exploran todos los posibles cálculos y ninguno acaba en configuración de aceptación. Esto supone un tiempo exponencial en el proceso de análisis sintáctico.

En la práctica, **los métodos de análisis sintáctico implementados en los compiladores usan autómatas con pila deterministas**, contruidos a partir de gramáticas para lenguajes de programación (como las llamadas *LL* y *LR*) que son más restringidas que las gramáticas libres del contexto generales. Estos métodos eficientes de análisis sintáctico aplicados a lenguajes de programación se verán en la asignatura de Compiladores.

4. Limitaciones de los autómatas con pila y las GLC

Existen lenguajes que no son libre del contexto. Eso quiere decir que no tienen gramáticas libres del contexto que los generen, ni autómatas con pila capaces de aceptarlos, debido a las **limitaciones de un AP** a la hora de procesar cadenas:

- En un autómata con pila *la memoria de la pila es de acceso limitado*: sólo tiene acceso directo al tope de la pila, no a los símbolos que hay debajo. Para acceder a estos símbolos hay que extraer primero los que están por encima, y al hacerlo se ‘pierden’. No se puede recorrer la pila con un apuntador de forma arbitraria.
- Por otra parte *el recorrido por la cadena de entrada es limitado*: los símbolos se leen de izquierda a derecha y los símbolos leídos no se pueden volver a leer. No se puede recorrer la entrada con un apuntador de forma arbitraria.

Si para validar las cadenas de un lenguaje necesitamos hacer un recorrido arbitrario por la pila o la entrada entonces diremos que el lenguaje no es libre del contexto.

Ejemplo 9 $L_{abc} = \{a^n b^n c^n \mid n \geq 0\}$ no es libre del contexto. Para aceptarlo con un AP tendríamos que introducir las a 's de la cadena de entrada en la pila para compararlas luego con las b 's. Pero compararlas con las b 's significa extraer las a 's de la pila, con lo cual después ya no habría forma de comparar con las c 's. A no ser que se retroceda en la cinta de entrada y se metan las b 's en la pila para compararlas con las c 's. Esto no puede hacerse con un autómata con pila. Luego L_{abc} no es un lenguaje libre del contexto.

Ejemplo 10 $L_{ww} = \{ww \mid w \in \{0, 1\}^*\}$ no es libre del contexto. Se parece al lenguaje $L_{ww^R} = \{ww^R \mid w \in \{0, 1\}^*\}$ que sí es libre del contexto, pero hay una diferencia fundamental a la hora

de reconocer las cadenas. En L_{ww^R} , cuando se lee la primera mitad w de una cadena se puede ir introduciendo en la pila, quedando en la pila w^R . Entonces el contenido de la pila se usa para comparar con la segunda mitad de la cadena en la entrada y si esta es w^R entonces se acepta. Para hacer lo mismo con el lenguaje L_{ww} tendríamos que leer hasta la primera mitad w de la cadena de entrada y luego retroceder para introducir los símbolos de w desde la derecha a la izquierda, de tal manera que la cadena de pila sea w , que luego debe compararse con la segunda mitad de la cadena en la entrada. Pero no podemos retroceder en la entrada con un AP. Otra forma sería leer la primera mitad w e introducirla en la pila como w^R y luego recorrer la cadena de entrada hasta el final y retroceder para comparar con el contenido de la pila, símbolo a símbolo de derecha a izquierda. Otra vez el mismo problema, no se puede retroceder en la entrada. Luego L_{ww} no puede ser aceptado con un autómata con pila y por tanto no es un lenguaje libre del contexto.

5. Clasificación de gramáticas, lenguajes y máquinas

En este curso hemos estudiado los lenguajes regulares y los lenguajes libres del contexto. Sabemos también que hay lenguajes formales que no son libres del contexto, como $\{a^n b^n c^n \mid n \geq 0\}$. Otros tipos de lenguajes son los lenguajes sensibles al contexto y los lenguajes con estructura de frase. Noam Chomsky estableció una **clasificación de los lenguajes formales en función de los tipos de gramáticas capaces de describirlos**. De esta forma tenemos:

- **Lenguajes regulares (tipo 3).** Un lenguaje es regular si puede ser generado por una gramática regular. Equivalentemente, un lenguaje es regular si puede ser aceptado por un autómata finito o puede ser descrito por una expresión regular. Los lenguajes regulares se agrupan en una clase o conjunto llamado \mathcal{L}_{reg} , también conocido como \mathcal{L}_3 .
- **Lenguajes libres del contexto (tipo 2).** Un lenguaje es libre del contexto si puede ser generado por una gramática libre del contexto. Esto es equivalente a decir que puede ser aceptado por un autómata con pila. Los lenguajes libres del contexto forman el conjunto \mathcal{L}_{LC} o \mathcal{L}_2 .
- **Lenguajes sensibles al contexto (tipo 1).** Un lenguaje es sensible al contexto si puede ser generado por una *gramática sensible al contexto*. Las reglas de producción de una gramática sensible al contexto son de la forma:

$$\boxed{\alpha A \beta \rightarrow \alpha \gamma \beta}$$

donde A es una variable, α, β, γ son cadenas de símbolos de la gramática (variables y/o terminales). La cadena γ no puede ser vacía, salvo en el caso $S \rightarrow \lambda$, siempre y cuando S no aparezca en la parte derecha de ninguna regla de producción. El sentido de estas reglas de producción es el de especificar que una variable A puede ser reemplazada por γ en una derivación sólo cuando A aparezca entre α y β . Las cadenas α y β forman el “contexto” de la variable A y este contexto se mantiene al aplicar la regla, de ahí el nombre “sensibles al contexto”. Con las gramáticas sensibles al contexto se pueden generar algunos lenguajes

que no son libres del contexto, como $L_{abc} = \{a^n b^n c^n \mid n \geq 0\}$.

Los lenguajes sensibles al contexto forman el conjunto \mathcal{L}_{sc} o \mathcal{L}_1 .

- **Lenguajes recursivamente enumerables (tipo 0).** Son los lenguajes más generales de todos porque son aquellos que pueden ser generados por las gramáticas más generales de todas, las *gramáticas sin restricciones*. En estas gramáticas las reglas de producción son de la forma $\alpha \rightarrow \beta$, donde en α aparece al menos una variable y β es cualquier cadena de símbolos de la gramática.

Los lenguajes recursivamente enumerables forman la clase \mathcal{L}_{re} o \mathcal{L}_0 y entre estos lenguajes están los más complejos desde el punto de vista computacional.

Teorema 3 (Jerarquía de Chomsky de lenguajes) *El conjunto de los lenguajes regulares está incluido propiamente en el conjunto de los lenguajes libres de contexto, que a su vez está incluido propiamente en el conjunto de lenguajes sensibles al contexto, que finalmente está incluido propiamente en el conjunto de lenguajes recursivamente enumerables. Esto es, se cumple la siguiente relación de inclusión propia entre conjuntos de lenguajes:*

$$\mathcal{L}_3 \subsetneq \mathcal{L}_2 \subsetneq \mathcal{L}_1 \subsetneq \mathcal{L}_0$$

Nota: “incluido propiamente” significa que está incluido pero no es igual y se denota con el símbolo \subsetneq . Aunque no vamos a ver la demostración de este teorema, ya hemos mostrado algunos resultados intermedios. Por ejemplo, en el tema anterior se vio que todo lenguaje regular es también libre del contexto, pero no al contrario: esto es lo que indica la inclusión propia $\mathcal{L}_3 \subsetneq \mathcal{L}_2$.

A partir del teorema de la jerarquía de Chomsky sabemos que si un lenguaje lo hemos clasificado como de un tipo, porque hemos encontrado una gramática de ese tipo que lo genera, automáticamente podemos afirmar que también es un lenguaje que pertenece a una clase más amplia en la jerarquía. Ahora bien, si sabemos que un lenguaje pertenece a una clase no podemos excluir la posibilidad de que también sea un lenguaje de una clase más restringida. Por ejemplo, si tenemos una gramática G que es libre del contexto no podemos afirmar de forma general que $L(G)$ no es regular. Esto es así porque puede existir una gramática G' que sea equivalente a G y regular, con lo cual $L(G) = L(G')$ también sería un lenguaje regular.

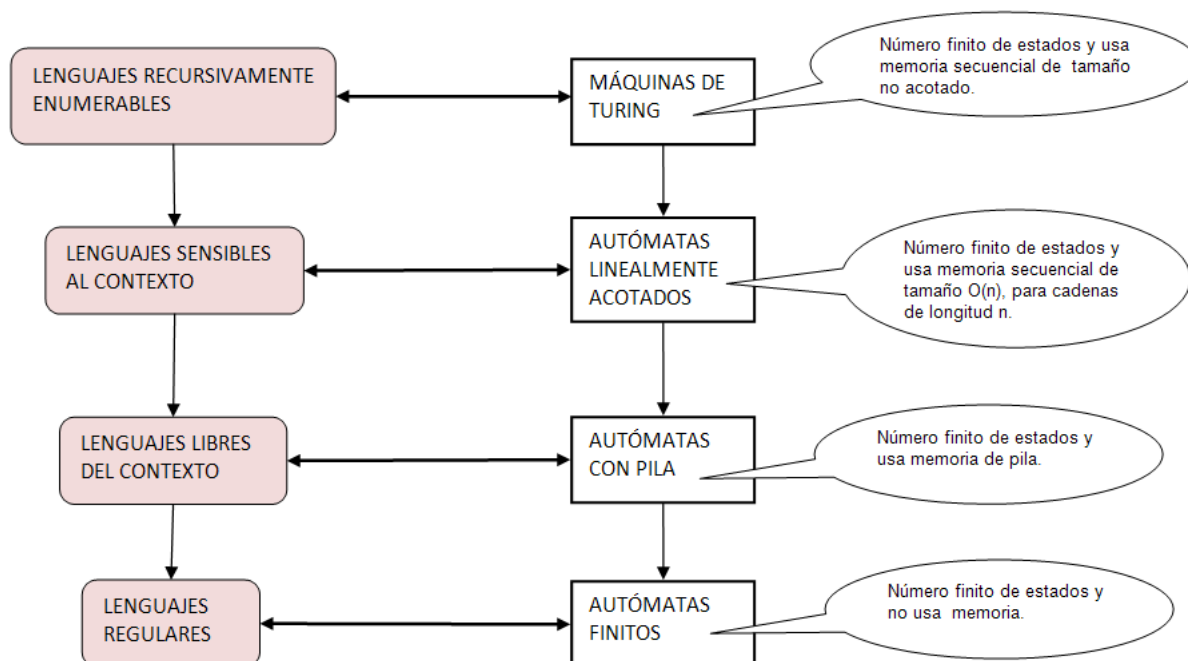
En la práctica, conviene encontrar una gramática del tipo más restringido posible para generar cierto lenguaje. **Cuanto más restringida sea la gramática más fácil será encontrar un algoritmo eficiente para resolver el problema de la pertenencia al lenguaje generado (validación)** y otras tareas de procesamiento de cadenas relacionadas con el problema de la validación. A continuación indicamos las cotas superiores de complejidad de algoritmos de validación para cada tipo de lenguaje, en función de la longitud n de la cadena de entrada. Indican la complejidad en el peor de los casos, que serían aquellos casos de lenguajes de mayor dificultad computacional en cada clase y el caso más desfavorable de cadena a validar.

- **Lenguajes regulares:** $O(n)$. Se consigue mediante simulación del autómata finito determinista que acepta el lenguaje dado.
- **Lenguajes libres del contexto:** $O(n^3)$. En lugar de simular un AP con backtracking en $O(2^n)$, se aplica un algoritmo llamado *CYK*, que se obtiene a partir de una GLC, previa transformación a lo que se denomina *forma normal de Chomsky*.

- **Lenguajes sensibles al contexto:** tiempo de ejecución exponencial.
- **Lenguajes recursivamente enumerables:** tiempo de ejecución infinito, el problema de la validación no puede resolverse de forma algorítmica, es indecidible.

Gramáticas y autómatas tienen una relación muy estrecha. Paralelamente a la jerarquía de lenguajes impuesta por los distintos tipos de gramáticas, existe otra equivalente de máquinas teóricas. La relación la podemos observar en la figura de abajo. Una flecha horizontal indica el tipo de máquina que es capaz de reconocer cada tipo lenguaje. Una flecha hacia abajo indica que cada clase de lenguajes contiene a los lenguajes por debajo en la jerarquía y cada máquina de un tipo puede simular a una máquina de tipo inferior, más simple.

Esquema de relación entre lenguajes y máquinas



6. Preguntas de evaluación

6.1. Problemas resueltos

1. Obtén un autómata con pila determinista M que acepte el lenguaje $I = \{w \in \{a, b\}^* \mid a's(w) = b's(w)\}$.

Solución: necesitamos dos estados. El estado q_0 representa la situación en que se ha leído una subcadena de igual número de a 's y b 's. El estado q_1 representa la situación en que se está comparando el número de a 's y b 's, usando la pila como 'contador' de símbolos y Z en el tope indica que el contador de a 's y b 's 'está a cero' (el número de a 's y b 's hasta el momento es igual). Se hace $F = \{q_0\}$ y así se permite aceptar λ sin que se produzca no determinismo en la configuración inicial. Por tanto, el autómata con pila aceptará las

cadenas por estado final.

La función de transición de M , con las reglas de transición comentadas, es:

1. $\delta(q_0, a, Z) = (q_1, aZ)$, 2. $\delta(q_0, b, Z) = (q_1, bZ)$
// comienza la 'cuenta' de a's o b's, apilando el símbolo correspondiente
3. $\delta(q_1, a, b) = (q_1, \lambda)$
// una a de entrada por una b de pila, disminuye la cuenta de b 's en la pila
4. $\delta(q_1, b, a) = (q_1, \lambda)$
// una b de entrada por una a de pila, disminuye la cuenta de a 's en la pila
5. $\delta(q_1, a, a) = (q_1, aa)$, 6. $\delta(q_1, b, b) = (q_1, bb)$
// apila para incrementar la cuenta de a 's o b 's leídas
7. $\delta(q_1, \lambda, Z) = (q_0, Z)$ // como al inicio, empieza un nuevo recuento de a 's y b 's

La configuración de aceptación para cualquier cadena de igual número de a 's y b 's es (q_0, λ, Z) : se ha leído toda la cadena y el autómata ha acabado en estado final. Si el número de a 's y b 's es distinto no se alcanza la configuración de aceptación. Por tanto, el AP acepta el lenguaje I por estado final, esto es, $L_{\text{EF}}(M) = I$.

2. Obtén un AP determinista que acepte el siguiente lenguaje y explica el funcionamiento a modo de justificación de la corrección del autómata. Indica cómo son, en general, las configuraciones de aceptación en este AP.

$$L_{a < b} = \{a^n b^m \mid 0 < n < m\}$$

Solución: describimos la **función de transición** del autómata, con estado final q_2 .

- | | |
|---|---|
| 1. $\delta(q_0, a, Z) = (q_0, aZ)$ | 4. $\delta(q_1, b, a) = (q_1, \lambda)$ |
| 2. $\delta(q_0, a, a) = (q_0, aa)$ | 5. $\delta(q_1, b, Z) = (q_2, b)$ |
| 3. $\delta(q_0, b, a) = (q_1, \lambda)$ | 6. $\delta(q_2, b, b) = (q_2, bb)$ |

Explicamos el funcionamiento de este autómata M , que acepta las cadenas del lenguaje **por estado final**:

- Con las transiciones 1 y 2 se introducen las a 's en la pila y debe haber al menos una a para que se produzca cambio al estado q_1 (porque $n > 0$).
- Por la transición 3, cuando se lee la primera b se extrae una a y cambia al estado q_1 . En el estado q_1 se extrae una a por cada b que se lee de la cadena de entrada (regla de tran. 4).
- Como tiene que haber más b 's que a 's en la cadena ($n < m$), cuando se saquen todas las a 's y quede al menos una b en la cadena de entrada será aplicable la transición 5, con la que se cambia a estado final q_2 . A partir de este estado sólo se pueden leer b 's de la cadena.
- La transición 6 se incluye para terminar de leer la cadena, como saltando las b 's que quedan.
- Por cada b en exceso en la cadena de entrada, las regla 5 y 6 insertan una b en la pila. De esta forma, cuando el AP acabe aceptando la cadena, quedarán b^{m-n} b 's en la pila.

Un cálculo genérico que refleja los cambios de estado para una cadena arbitraria $a^n b^m$ con $0 < n < m$ es:

$$(q_0, a^n b^m, Z) \Rightarrow^* (q_0, b^m, a^n Z) \Rightarrow^* (q_1, b^{m-n}, Z) \Rightarrow^* (q_2, \lambda, b^{m-n})$$

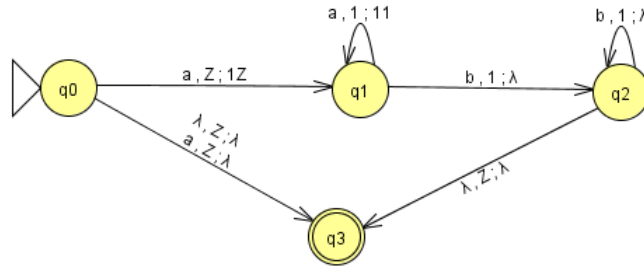
Como vemos las configuraciones de aceptación son siempre de la forma (q_2, λ, b^{m-n}) . Si la cadena de entrada no tiene más b 's que a 's entonces no se alcanzará el estado final habiendo leído toda la cadena, lo mismo que si hay b 's y a 's mezcladas. Luego el autómata es correcto para este lenguaje, se cumple: $L_{\text{EF}}(M) = L_{a < b}$.

3. Consideremos el AP con $F = \{q_3\}$ y función de transición δ dada por:

- | | |
|--|---|
| 1. $\delta(q_0, a, Z) = \{(q_1, 1Z), (q_3, \lambda)\}$ | 4. $\delta(q_1, b, 1) = \{(q_2, \lambda)\}$ |
| 2. $\delta(q_0, \lambda, Z) = \{(q_3, \lambda)\}$ | 5. $\delta(q_2, b, 1) = \{(q_2, \lambda)\}$ |
| 3. $\delta(q_1, a, 1) = \{(q_1, 11)\}$ | 6. $\delta(q_2, \lambda, Z) = \{(q_3, \lambda)\}$ |

a) Indica cuál es el alfabeto de entrada, el alfabeto de la pila y dibuja el diagrama de transición.

Solución: por las transiciones vemos que el alfabeto de entrada es $V = \{a, b\}$ y el de la pila es $\Sigma = \{Z, 1\}$, porque son los únicos símbolos que se extraen o insertan en la pila. El diagrama de transición es el siguiente:



b) Indica si el autómata es determinista o no y por qué.

Solución: el AP es **no determinista** porque tiene una regla de transición con múltiples opciones (la 1) y, además, las transiciones 1 y 2 también producen no determinismo, pues ambas son aplicables en una configuración como (q_0, ab, Z) , con lo que pueden obtenerse dos configuraciones distintas en un paso de cálculo.

c) Mostrar un cálculo por el que se acepta λ , a y $aabb$, se supone que por estado final, ya que este autómata tiene un estado final.

Solución: La **cadena vacía** λ es aceptada por el cálculo:

$$(q_0, \lambda, Z) \Rightarrow (q_3, \lambda, \lambda)$$

La cadena **a** es aceptada por el cálculo:

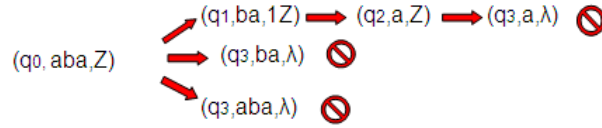
$$(q_0, a, Z) \Rightarrow (q_3, \lambda, \lambda)$$

La cadena **aabb** es aceptada por el cálculo:

$$(q_0, aabb, Z) \Rightarrow (q_1, abb, 1Z) \Rightarrow (q_1, bb, 11Z) \Rightarrow (q_2, b, 1Z) \Rightarrow (q_2, \lambda, Z) \Rightarrow (q_3, \lambda, \lambda)$$

d) Justifica si la cadena aba es aceptada o no por este autómata.

Solución: Como el AP es no determinista tenemos que analizar todos los posibles cálculos para la cadena aba y para ello mostramos un árbol de cálculo paralelo donde cada rama es un posible cálculo del AP con aba .



Como vemos, ninguno de los dos cálculos posibles acaba en configuración de aceptación, porque no se termina de leer la cadena. Luego *aba* es rechazada.

e) Describe el lenguaje $L_{\text{EF}}(M)$ y razona la respuesta.

Solución: se trata de deducir el lenguaje que acepta este autómata por estado final. Para ello analizamos los posibles cálculos que acaban en configuración de aceptación:

1. Sabemos que acepta λ y a . Además, esas son las únicas cadenas que se aceptan aplicando la regla 2: $\delta(q_0, \lambda, Z) = \{(q_3, \lambda)\}$, para aceptar λ , y la regla de transición 1, opción 2: $(q_3, \lambda) \in \delta(q_0, a, Z)$, para aceptar a .
2. Para aplicar la opción 1 de la regla 1 la cadena debe comenzar por a . Al aplicarla pasa al estado q_1 e inserta un 1 en la pila. En q_1 puede seguir leyendo a 's y por cada a inserta un 1. Para pasar a q_2 se tiene que leer al menos una b y una vez en 2 se puede leer una b por cada 1 de la pila. Para que sea aplicable la regla 6, que permite llegar al estado final, es necesario que se hayan extraído todos los 1's insertados, pero para ello es necesario que el número de a 's en la cadena sea igual al número de b 's.

Por tanto, la única configuración de aceptación posible para este autómata es: (q_3, λ, λ) , que se puede alcanzar solamente con los siguientes cálculos:

$$\begin{aligned} (q_0, \lambda, Z) &\Rightarrow (q_3, \lambda, \lambda) \text{ y acepta } \lambda \\ (q_0, a, Z) &\Rightarrow (q_3, \lambda, \lambda) \text{ y acepta } a \\ (q_0, a^n b^n, Z) &\Rightarrow^* (q_2, \lambda, Z) \Rightarrow (q_3, \lambda, \lambda) \text{ y acepta } a^n b^n, n > 0 \end{aligned}$$

Por lo dicho anteriormente se puede deducir que:

$$L_{\text{EF}}(M) = \{a^n b^n \mid n \geq 0\} \cup \{a\} = \{w \in \{a, b\}^* \mid (w = a) \vee (w = a^n b^n), n \geq 0\}$$

4. Obtén un AP que acepte el lenguaje $L_{\text{wwr}} = \{ww^R \mid w \in \{0, 1\}^*\}$ y cumpla $|\delta(q, \sigma, Z)| \leq 1$, es decir, que no haya reglas de transición de múltiples opciones. Indica si este autómata es no determinista y por qué.

Solución: en los apuntes se dice que este lenguaje es libre del contexto no determinista y se mostró un AP no determinista con reglas de múltiples opciones que lo aceptaba. Si queremos ahora que no haya reglas de múltiples opciones entonces debemos introducir λ -transiciones, que van a producir no determinismo. Este lenguaje no puede ser aceptado por un autómata determinista. La función de transición de un AP que acepta el lenguaje por pila vacía es:

$$\begin{array}{ll} 1. \delta(q_0, 0, Z) = (q_0, 0) & 7. \delta(q_0, \lambda, 0) = (q_1, 0) \\ 2. \delta(q_0, 1, Z) = (q_0, 1) & 8. \delta(q_0, \lambda, 1) = (q_1, 1) \\ 3. \delta(q_0, 0, 0) = (q_0, 00) & 9. \delta(q_0, \lambda, Z) = (q_1, \lambda) \\ 4. \delta(q_0, 1, 1) = (q_0, 11) & 10. \delta(q_1, 0, 0) = (q_1, \lambda) \\ 5. \delta(q_0, 0, 1) = (q_0, 01) & 11. \delta(q_1, 1, 1) = (q_1, \lambda) \\ 6. \delta(q_0, 1, 0) = (q_0, 10) & \end{array}$$

Ahora el autómata es no determinista porque las transiciones 3 y 7 son ambas aplicables en una configuración del tipo $(q_0, 0y, 0\beta)$. De forma similar, la 7 produce conflicto con la 6, la 8 con la 4 y la 5 y la 9 con la 1 y la 2. En definitiva, no se cumple la segunda condición que se exige a un autómata con pila para que sea determinista, porque tenemos definidas simultáneamente transiciones con λ y con un símbolo de entrada para la misma combinación de estado y tope de pila.

5. La siguiente gramática: $R \rightarrow RR \mid R^* \mid R' \mid R \mid (R) \mid 0 \mid 1$, genera el lenguaje de las expresiones regulares restringidas a símbolos binarios, sin considerar que λ es una ER. Se pone el símbolo \mid de operador de alternancia o unión de ER entre comillas para indicar que es un símbolo terminal y evitar así confusión con el símbolo separador de reglas de producción. Obtén un autómata con pila por el método *GLCtoAP* no determinista y un cálculo que demuestre que la cadena 0^*01 es una expresión regular sintácticamente correcta según la gramática dada.

Solución: el autómata con pila tendrá dos estados $Q = \{q_0, q_1\}$ y no se usan estados finales, aceptará por pila vacía. El vocabulario de entrada es $V = V_T = \{0, 1, (,), *, |\}$ y el de pila Σ contiene todos los símbolos de la gramática más el símbolo inicial Z . La función de transición es:

$$\begin{aligned}\delta(q_0, \lambda, Z) &= (q_1, R) \\ \delta(q_1, \lambda, R) &= \{(q_1, R|R), (q_1, RR), (q_1, R^*), (q_1, (R)), (q_1, 0), (q_1, 1)\} \\ \delta(q_1, 0, 0) &= (q_1, \lambda) \\ \delta(q_1, 1, 1) &= (q_1, \lambda) \\ \delta(q_1, *, *) &= (q_1, \lambda) \\ \delta(q_1, |, |) &= (q_1, \lambda) \\ \delta(q_1, (, () &= (q_1, \lambda) \\ \delta(q_1,),)) &= (q_1, \lambda)\end{aligned}$$

Dado que el autómata es no determinista, para obtener un cálculo que acepte 0^*01 sin equivocarnos, podemos fijarnos en la derivación más a la izquierda de esa cadena para ver qué reglas se aplican, ya que el autómata funciona simulando la derivación más la izquierda, por lo que aplicará las opciones de la regla segunda en el mismo orden en que se aplican las reglas de la gramática correspondientes.

$$R \xRightarrow{R \rightarrow RR} \boxed{R} R \xRightarrow{R \rightarrow R^*} \boxed{R}^* R \xRightarrow{R \rightarrow 0} 0^* R \xRightarrow{R \rightarrow RR} 0^* \boxed{R} R \xRightarrow{R \rightarrow 0} 0^* 0 R \xRightarrow{R \rightarrow 1} 0^* 0 1$$

El cálculo que acepta 0^*01 es:

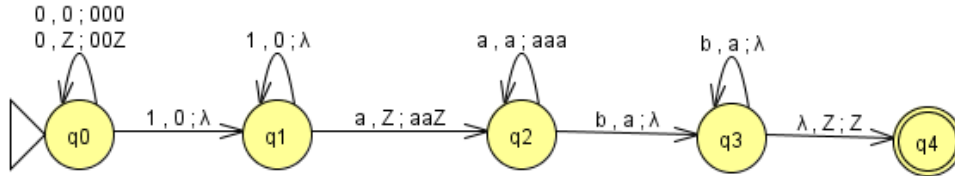
$$\begin{aligned}(q_0, 0^*01, Z) &\Rightarrow (q_1, 0^*01, R) \Rightarrow (q_1, 0^*01, RR) \Rightarrow (q_1, 0^*01, R^*R) \Rightarrow (q_1, 0^*01, 0^*R) \Rightarrow \\ &(q_1, 0^*01, 0^*R) \Rightarrow (q_1, 01, R) \Rightarrow (q_1, 01, RR) \Rightarrow (q_1, 01, 0R) \Rightarrow (q_1, 1, R) \Rightarrow (q_1, 1, 1) \Rightarrow (q_1, \lambda, \lambda)\end{aligned}$$

6.2. Problemas propuestos

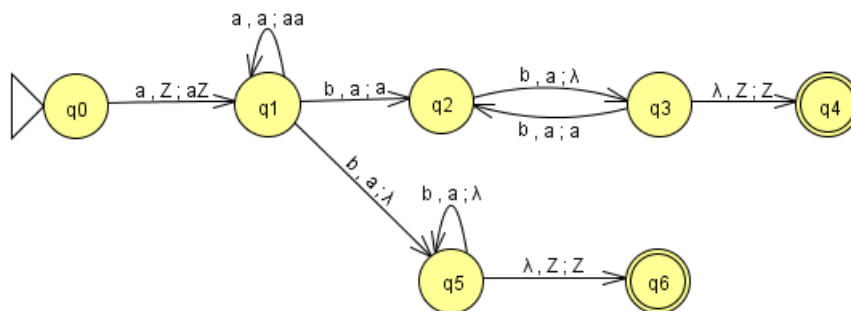
1. Obtén un AP determinista que acepte el lenguaje $L = \{a^n b^n \mid n \geq 0\} \cup \{a\}$.
2. Dado el lenguaje $L = \{0^k 1^m 0^n \mid n = k + (m/2), m, k > 0, m \text{ es par}\}$,

- a) Diseña un AP determinista que acepte L (por estado final y no más de 5 estados contando el final) y mostrar un cálculo para 01100. Explica el funcionamiento del AP obtenido.
- b) Modifica convenientemente las transiciones del AP anterior para obtener otro equivalente determinista que acepte por pila vacía y con un estado menos.

3. Consideramos un AP con símbolo inicial de pila Z y con diagrama de transición:



- a) Describe los componentes del AP en notación matemática.
 - b) Deduce el lenguaje $L_{\text{EF}}(M)$, con un argumento claro y preciso, y descríbelo por comprensión.
 - c) Obtén un AP determinista equivalente que tenga sólo dos estados y acepte por pila vacía.
4. Dado el lenguaje $L = \{xwx \mid x \in \{0,1\}, w \in \{a,b,c\}^*\}$
- a) Obtén un AP determinista que acepte L .
 - b) Demuestra que L es un lenguaje regular.
 - c) Obtén una GLC que genere L .
5. La siguiente gramática: $S \rightarrow SS \mid [S] \mid []$, genera el lenguaje de los corchetes balanceados.
- a) Obtén un autómata con pila por el método *GLCtoAP* no determinista y un cálculo que demuestre que la cadena $[[[]]$ es válida.
 - b) Obtén un AP determinista que acepte el lenguaje de los corchetes balanceados.
6. Describe formalmente el lenguaje aceptado por el autómata con pila siguiente y razona la respuesta. ¿Es determinista? ¿Por qué?



7. (!) Obtén un autómata con pila (puede ser no determinista) que acepte el lenguaje $L = \{a^n b^m \mid 1 \leq m \leq n \leq 3m\}$.