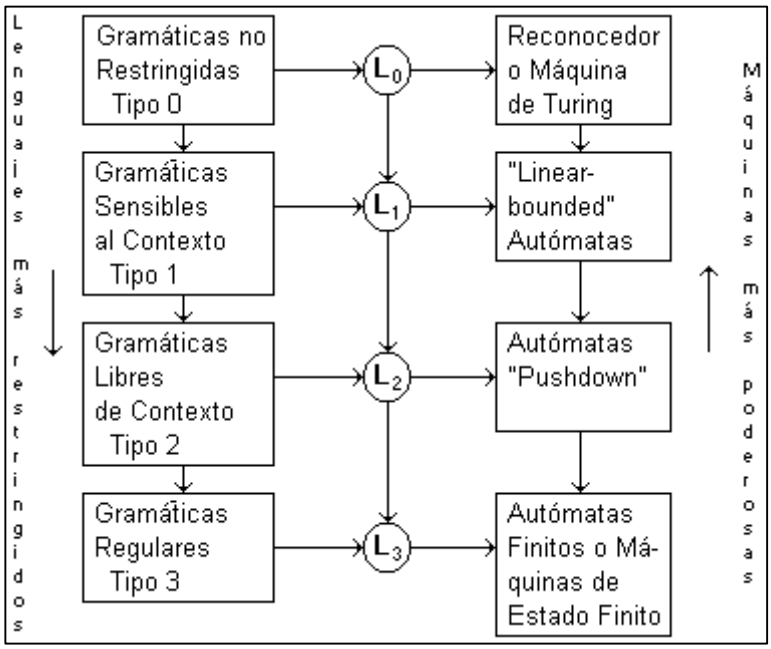
FACULTAD REGIONAL ROSARIO

AUTÓMATAS PUSH-DOWN Y MÁQUINAS DE TURING

GUÍA TEÓRICO-PRÁCTICA
PARA ALUMNOS DE LA CÁTEDRA
SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES
DE LA CARRERA DE INGENIERÍA EN SISTEMAS DE INFORMACIÓN

ING. EDUARDO AMAR

Jerarquía de Máquinas Abstractas y Lenguajes



Comenzaremos por las gramáticas regulares o de tipo 3, que son las más simples, y descubriremos problemas o lenguajes que exceden la capacidad de las máquinas de ese nivel; así nos moveremos hacia niveles con lenguajes y máquinas más poderosos. En cada nivel vamos a encontrar el mismo tipo de inconvenientes hasta llegar al nivel de las gramáticas de tipo 0 y las máquinas de Turing.

1.1.1. Autómatas Push-Down

La clase de lenguajes generados por *Gramáticas Libres de Contexto (GLC)* es más amplia que la clase de lenguajes definidos por *Expresiones Regulares*. Esto significa que todos los lenguajes regulares pueden ser generados por GLC, como así también algunos lenguajes no regulares (por ejemplo, $\{a^n b^n\}$ y Palíndromos).

Luego de introducir los lenguajes regulares definidos por expresiones regulares encontramos una clase de máquinas abstractas, los Autómatas Finitos (AF) con la siguiente propiedad dual:

- **para cada lenguaje regular hay al menos una máquina que funciona exitosamente solo sobre las cadenas de entrada de ese lenguaje y**
- **para cada máquina en la clase, el conjunto de palabras que acepta es un lenguaje regular.**

Esta correspondencia fue crucial para nuestro profundo entendimiento de esta colección de lenguajes.

El Pumping Lemma, complementos, intersección, decibilidad, fueron todos aprendidos desde el punto de vista de la máquina, no desde la expresión regular. Ahora estamos considerando una clase diferente de lenguajes pero queremos contestar algunas preguntas; por lo tanto nos gustaría nuevamente encontrar una formulación para una máquina.

Estamos buscando un modelo matemático de alguna clase de máquinas que corresponda análogamente a las GLC, esto es, debe haber al menos una máquina que acepta cada GLC y el lenguaje aceptado por cada máquina es libre de contexto.

Queremos reconocedores de GLC o aceptores de GLC así como los AF son reconocedores y aceptores de lenguajes regulares.

Esperamos que un análisis de la máquina nos ayudará a comprender los lenguajes en un sentido más profundo, tal como un análisis de AF lleva a teoremas sobre lenguajes regulares.

En este capítulo desarrollamos una nueva clase de máquinas. En el próximo capítulo probamos que estas nuevas máquinas se corresponden con GLC en el sentido que deseamos.

Para construir estas nuevas máquinas, comenzamos con nuestros viejos AF e incluimos algunos nuevos descubrimientos que los ampliarán y los harán más potentes.

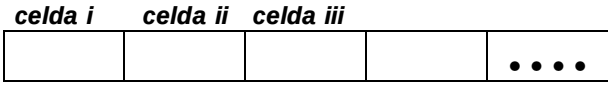
Lo que haremos primero es desarrollar una representación diferente para los AF, una que será fácil de aumentar con nuestros agregados.

No hemos dado hasta ahora un nombre a la parte del AF en la que reside la cadena de entrada mientras se está ejecutando. Llamaremos a la misma la CINTA de Entrada. La CINTA de entrada debe ser suficientemente larga para aceptar cualquier posible entrada, y ya que cualquier palabra en a* es una posible entrada, la CINTA debe ser infinitamente larga (tal CINTA es muy cara).

La CINTA tiene una primera ubicación para la primera letra de la entrada, luego una segunda ubicación, y así sucesivamente. Por lo tanto decimos que la CINTA es infinita en una única dirección solamente. Algunas personas usan el término *"medio infinita"* para esta condición.

Dibujamos la CINTA como se muestra aquí:

Las ubicaciones en las cuales ponemos las letras de entrada se llaman celdas. Nombramos las celdas con números romanos en minúscula.



Abajo mostramos un ejemplo de una CINTA de entrada ya cargada con la cadena de entrada aaba. El carácter "D" se usa para indicar un blanco en una celda de la CINTA.

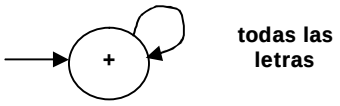
La gran mayoría (todas menos 4) de las celdas en la CINTA de entrada están vacías, o sea que están cargadas con blancos, ΔΔΔΔ.....

A medida que procesamos esta CINTA en la máquina leemos una letra por vez y la eliminamos al ser usada. Cuando alcanzamos la primera celda en blanco nos detenemos. Siempre presumimos que una vez que el primer blanco es encontrado el resto de la CINTA también tiene blancos. Leemos de izquierda a derecha y nunca volvemos atrás a una celda que ya fue leída.

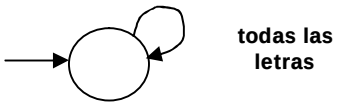
Como parte de nuestra nueva representación para AF, introducimos los siguientes símbolos:

Las flechas (bordes dirigidos) que ingresan o salen de los estados se pueden dibujar en cualquier ángulo.

1. **El estado de START es como el estado – , que se encuentra conectado a otros estados en un Grafo de Transiciones (GT) por una arista Λ. Comenzamos el proceso allí, pero no leemos ninguna letra de entrada. Avanzamos inmediatamente al próximo estado. Un estado de START no tiene flechas ingresando a él.**
2. **Un estado ACCEPT o de *aceptación* es una abreviación para un *estado de detención*, una vez que se ingresa en él, no se puede salir, tal como:**



3. Un estado REJECT es un *estado de detención* que *no es un final aceptable*.

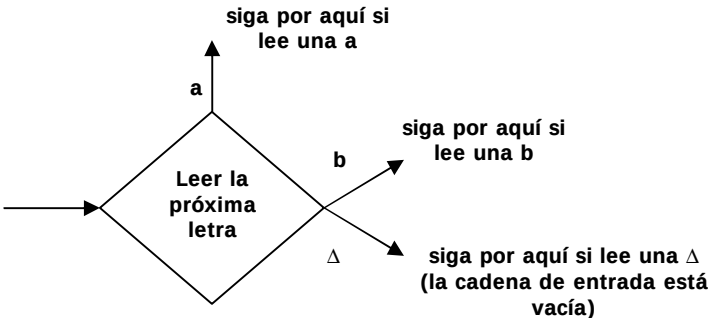


Ya que hemos usado el adjetivo *"final"* para aplicarlo solo a estados de aceptación en AF, llamamos a los nuevos estados ACCEPT y REJECT, *"halt states"* o *"estados de detención"*.

Previamente podríamos pasar a través de un estado final si no hubiéramos terminado de leer los datos de entrada; estos *"estados de detención"* no pueden ser atravesados. Podemos entrar a un estado ACCEPT o REJECT pero no podemos salir de ellos.

Estamos cambiando nuestros diagramas de AF para que cada función que corresponda a un estado se ejecute en una caja separada del diagrama.

La tarea más importante desarrollada por un estado en un AF es leer una letra de entrada y saltar a otros estados dependiendo de que letra ha sido leída. Para hacer este trabajo desde ahora introducimos el estado READ. Estos estados se dibujan como cajas con forma de rombo como se muestra a continuación:

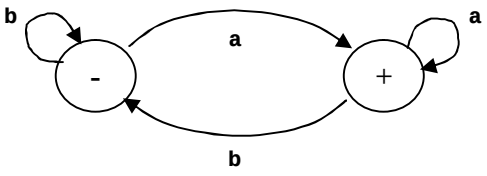


Aquí nuevamente las direcciones de las aristas en el dibujo de arriba muestran solo una de las muchas posibilidades. Cuando el carácter Δ es leído desde la CINTA, significa que no tenemos más letras de entrada. Entonces, hemos terminado el procesamiento de la cadena de entrada.

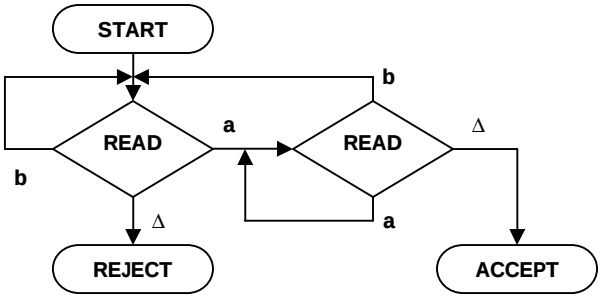
La línea con el símbolo Δ llevará a un estado ACCEPT si el estado en el que nos encontramos es un estado final. En nuestros viejos diagramas para AF nunca explicamos como sabíamos que no teníamos más letras de entrada. En estos nuevos dibujos podemos reconocer este hecho *leyendo un blanco desde la CINTA*.

Estas sugerencias no han alterado el poder de nuestras máquinas. Solo hemos introducido una nueva representación gráfica que no alterará las habilidades de aceptación del lenguaje.

Los AF que solían ser graficados de esta manera:

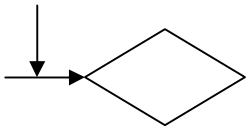


(AF que acepta todas las palabras terminadas en la letra a, se grafican ahora con el nuevo simbolismo, como la máquina que se ve abajo:



Note que la línea desde START no necesita etiqueta ya que START no lee ninguna letra. Todas las otras aristas requieren etiquetas. Hemos dibujado las aristas como segmentos de líneas rectas, no curvas y loops como antes.

También usamos la notación de diagramas de electrónica para tendidos de cables. Por ejemplo:



Nuestra máquina es todavía un AF. Las aristas etiquetadas D no serán confundidas con aristas etiquetadas con el símbolo L. Estas aristas, Δ, van solo de cajas de tipo READ hacia estados de detención.

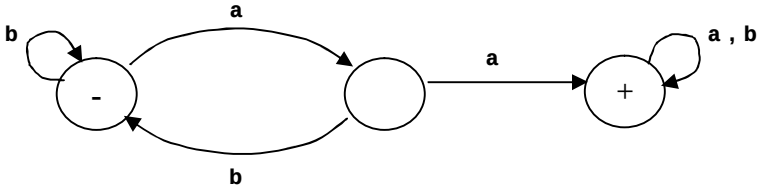
Hemos extraído los signos + y – fuera de los círculos que se usaban para indicar estados. Los *"estados"* ahora son solo cajas de READ y no tienen status de *"finales"* o *"no finales"*.

En el AF anterior, si nos quedamos sin letras de entrada en el estado READ de la izquierda, encontraremos un Δ en la CINTA de entrada y por lo tanto tomamos la arista Δ hasta REJECT. Al leer un Δ en el estado READ que corresponde a un estado final de un AF llegamos a un estado ACCEPT.

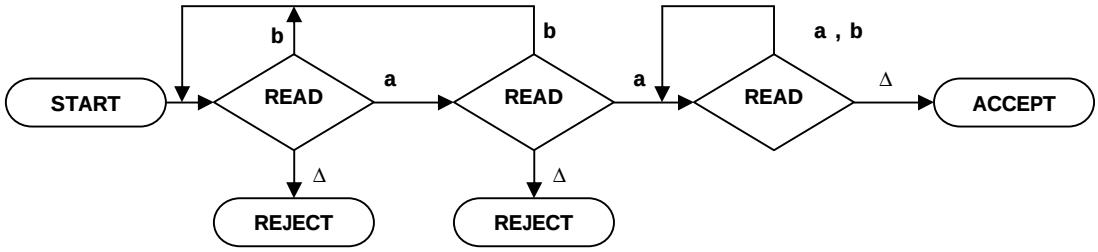
Ejemplo

Reconocimiento de una Expresión Regular

Demos otro ejemplo sobre esta nueva notación gráfica:



Se representa como:



Estos gráficos se parecen más a los *"diagramas de flujo"* que a los viejos gráficos para los AF.

El estudio generalizado de los diagramas de flujo es una estructura matemática que forma parte de la Teoría de la Computación pero está más allá de nuestro interés.

La razón que tenemos para construir nuevos gráficos para los AF (que ya tenían buenos gráficos) es que ahora es más fácil hacer un agregado a nuestra máquina llamado PUSHDOWN STACK o PUSHDOWN STORE o Pila Push-Down.

Un PUSHDOWN STACK es un lugar en el que las letras de entrada (u otra información) puede almacenarse hasta que queramos referirnos a ellas nuevamente.

Esta estructura mantiene las letras que han sido leídas en una larga línea (tantas letras como deseemos).

La operación PUSH agrega una nueva letra a la línea. La nueva letra es colocada en el tope del STACK y todas las demás letras son empujadas abajo o atrás. Antes que la máquina comience a procesar una cadena de entrada el STACK se presume que está vacío lo que significa que cada ubicación de almacenamiento en el mismo contiene un blanco.

Si el STACK es entonces llenado con las letras a, b, c, d en esta secuencia de instrucciones:

```
PUSH a
PUSH b
PUSH c
PUSH d
```

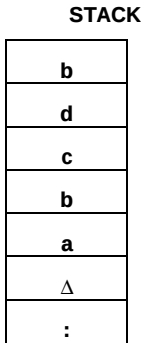
Entonces la letra de arriba en el STACK, es d, la segunda es c, la tercera es b, y la cuarta es a.

Si ahora ejecutamos la instrucción:

```
PUSH b
```

La letra b se agregará al STACK en la parte de arriba. La d será empujada hacia abajo a la posición 2, la c a la 3, la otra b a la 4 y la última a irá a la posición 5.

Una representación gráfica de un STACK con estas letras incluidas se muestra abajo. Debajo del fondo de la a presumimos que el resto del STACK, que parece una CINTA de entrada, tiene infinitas ubicaciones de almacenamiento, llenas solo con blancos.



La instrucción para tomar una letra y sacarla del STACK se llama POP.

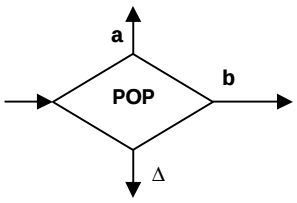
Esto causa que la letra que está más arriba en el STACK sea sacada del mismo. El resto de las letras son movidas una posición hacia arriba cada una. Un STACK PUSH DOWN es llamado un archivo LIFO *(Last In First Out, Ultimo Entrado Primero Salido).*

No es como un área de almacenamiento normal de una computadora, que permite acceso directo (podemos recuperar objetos de cualquier parte, más allá del orden en que fueron almacenados). Un STACK PUSHDOWN permite leer solo la letra almacenada en su parte superior. Si queremos leer la tercera letra del STACK debemos ejecutar POP, POP, POP, pero entonces habremos adicionalmente extraído las dos primeras letras y ya no estarán más en el STACK.

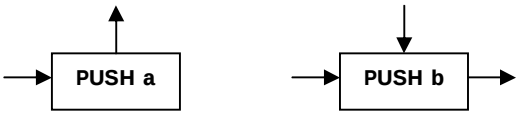
Tampoco tenemos instrucciones simples para determinar la última letra del STACK, o para decir cuantas letras b hay en el mismo. Las únicas operaciones permitidas sobre el STACK son PUSH y POP.

Si ejecutamos POP sobre un STACK vacío, así como leer una CINTA vacía, nos da el carácter blanco ∆.

Podemos agregar un STACK PUSHDOWN y las operaciones PUSH y POP a nuestros nuevos diagramas de AF incluyendo tantos estados como queramos



y los estados



Las aristas que salen de un estado POP son etiquetadas de la misma forma que las aristas de un estado READ, por el momento una para cada carácter que aparezca en el STACK, incluyendo el blanco.

Note que la separación en ramas puede ocurrir en un estado POP pero no en un estado PUSH. Podemos dejar los estados PUSH solo por la única ruta indicada, a pesar que podemos entrar a un estado PUSH desde cualquier dirección.

Cuando un AF ha sido expandido con un STACK y los estados PUSH y POP, podemos llamarlo Autómata Push-Down, abreviado APD.

Estos APD fueron introducidos por Anthony Oettinger en 1961 y Marcel Schützenberger en 1963 y fueron luego estudiados por Robert Every, también en 1963.

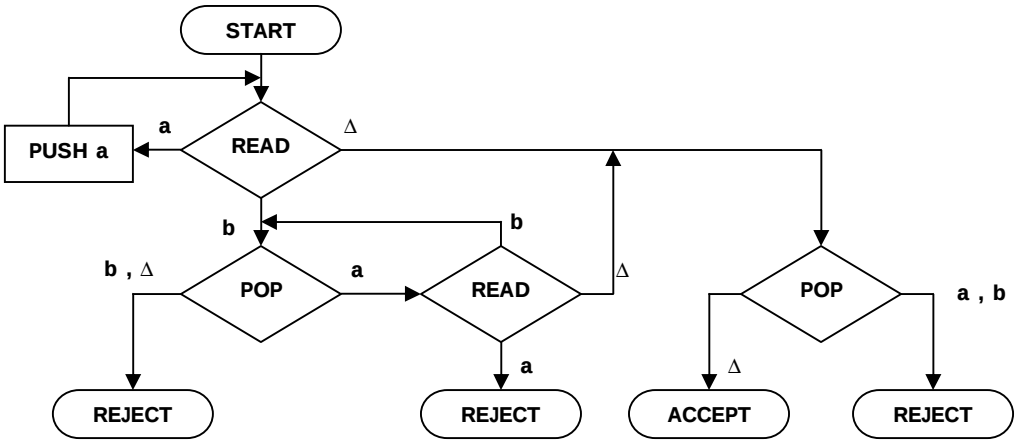
La noción de PUSHDOWN STACK como una estructura de datos existía desde hace tiempo, pero los matemáticos luego reconocieron que al incorporar esta estructura a los AF, las capacidades para reconocer lenguajes aumentaron considerablemente. Schützenberger desarrolló una teoría matemática de lenguajes aplicando los AF y los APD.

La definición precisa se dará pronto, luego de unos ejemplos.

Ejemplo

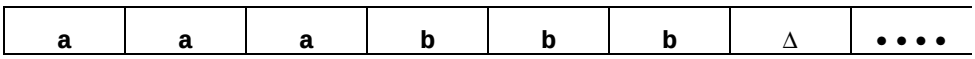
Reconocimiento de una GLC

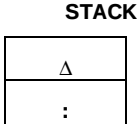
Considere el siguiente APD:



Antes de empezar a analizar esta máquina en general, veámosla en operación sobre la cadena de entrada aaabbb. Comenzamos asumiendo que esta cadena ha sido puesta en la CINTA.

Siempre comenzamos la operación del APD con el STACK vacío como se muestra:





Debemos comenzar por START. Desde allí vamos directamente al primer READ, un estado que lee la primera letra de entrada. Es una a, por lo tanto la tachamos de la

CINTA (ha sido leída) y avanzamos por la arista a desde el estado READ. Esta arista nos lleva al estado PUSH a que nos hace ingresar una a en el STACK.

Ahora la CINTA y el STACK se ven de la siguiente manera:

~~a~~	a	a	b	b	b	∆	• • • •

STACK

a
∆
:

La arista desde el estado PUSH a nos lleva de vuelta a la misma caja READ, por lo que retornamos a este estado. Ahora leemos otra a y procedemos como antes a lo largo de la arista a para ingresarla al STACK. Nuevamente volvemos a la caja READ. Leemos nuestra tercera a y la ingresamos al STACK. La CINTA y el STACK ahora se ven como:

~~a~~	~~a~~	~~a~~	b	b	b	∆	• • • •

STACK

a
a
a
∆
:

Luego del tercer PUSH a, volvemos al READ. Esta vez, sin embargo, leemos la letra b. Esto significa que tomamos la arista b hacia el estado POP que está más abajo. Al leer la b nos queda la CINTA como:

~~a~~	~~a~~	~~a~~	~~b~~	b	b	∆	• • • •

El estado POP toma el elemento superior del STACK. Es una a. Debe ser a o ∆ ya que los únicos caracteres ingresados al STACK en todo el programa son a. Si fuera una ∆ o la imposible elección b, tendríamos que ir a un estado REJECT. Sin embargo, esta vez, cuando sacamos del STACK una letra, es una a, quedando el STACK como:

STACK

a
a
∆
:

Siguiendo el camino a desde POP nos lleva al otro READ. La próxima letra en la CINTA a ser leída es b. Esto deja la CINTA como:

~~a~~	~~a~~	~~a~~	~~b~~	~~b~~	b	∆	• • • •

El camino b desde el segundo estado READ ahora nos lleva nuevamente de vuelta al estado POP. Por lo tanto, tomamos otra a del STACK. El STACK ahora queda con una sola a:

STACK

a
∆
:

La línea a del POP nos lleva al mismo READ. Hay una sola letra en la CINTA, una b. La leemos y dejamos la CINTA vacía, o sea, todos blancos. Sin embargo la máquina aún no sabe que la CINTA está vacía. Descubrirá esto solo cuando trate de leer la CINTA y encuentre una ∆.

~~a~~	~~a~~	~~a~~	~~b~~	~~b~~	~~b~~	∆	• • • •

La letra b que leímos vuelve al estado POP. Entonces tomamos la última a del STACK y lo dejamos vacío, todos blancos.

STACK

a
∆
:

La a nos lleva del POP de la derecha al READ otra vez. Esta vez lo único que podemos leer de la CINTA es un blanco, ∆. La arista ∆ va hacia otro POP a la derecha. Este POP toma una letra del STACK, pero al estar vacío, decimos que extraemos una ∆.

Esto significa que debemos seguir la arista ∆, que nos lleva directamente al estado ACCEPT.

Por lo tanto la palabra aaabbb es aceptada por esta máquina.

Se puede observar más de esto. El lenguaje de palabras aceptadas por esta máquina es exactamente:

$$\{a^n b^n, n >= 0\}$$

Hemos mostrado que el lenguaje aceptado por el APD anterior no podría ser aceptado por ningún AF, por lo tanto los APD son más potentes que los AF.

Podemos decir más potentes porque todos los lenguajes regulares pueden ser aceptados por algún APD ya que pueden ser aceptados por algún AF y éstos son como APD sin el uso de un STACK.

Los APD son las máquinas que necesitamos para reconocer GLC. Toda GLC puede ser definida como un lenguaje aceptado por algún APD y el lenguaje aceptado por algún APD puede ser definido por alguna GLC.

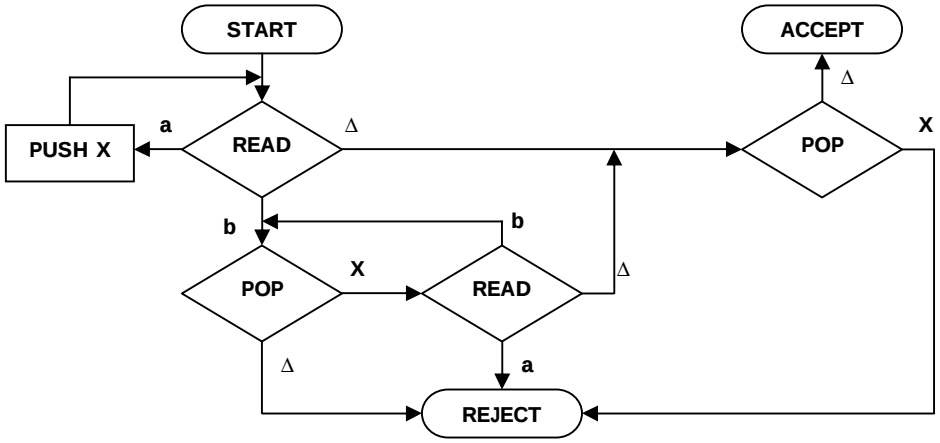
Tomemos un instante para considerar que hace a estas máquinas más poderosas que los AF. La razón fundamental es que tienen una capacidad ilimitada de memoria. Pueden saber donde han estado y cuantas veces.

La causa por la que un AF no puede aceptar el lenguaje a^nb^n es que no puede llevar una cuenta de la cantidad de veces que ha loopeado sobre la arista de letras a, para después compararla con la letra b.

Hay dos puntos que debemos discutir.

1. **El primero es que <u>no necesitamos restricciones sobre usar el mismo alfabeto para cadenas de entrada que para el uso del STACK</u>. En el ejemplo anterior podríamos haber leído una a desde la CINTA y luego agregar una X en el STACK y dejar que X lleve la cuenta del número de a ingresadas. En este caso cuando testeamos el STACK con un estado POP, avanzamos por la arista X o ∆.**

   La máquina entonces se vería como:



   Hemos dibujado esta versión del APD con algunas variaciones menores.

   Los estados READ deben proveer aristas para a, b o ∆. Los estados POP deben proveer aristas para X o ∆. Eliminamos dos estados REJECT y todos van al mismo lugar.

   Cuando definimos el APD se requiere la especificación del alfabeto S de la CINTA y del alfabeto G del STACK, que pueden ser diferentes.

2. **El segundo punto que debemos discutir es la <u>posibilidad de indeterminismo</u>.**

   En nuestra búsqueda de máquinas equivalentes a GLC vimos que un dispositivo de memoria de alguna clase se requiere para aceptar el lenguaje a^nb^n.

Es suficiente el agregado del STACK para aceptar todas las GLC o se necesita algún cambio?

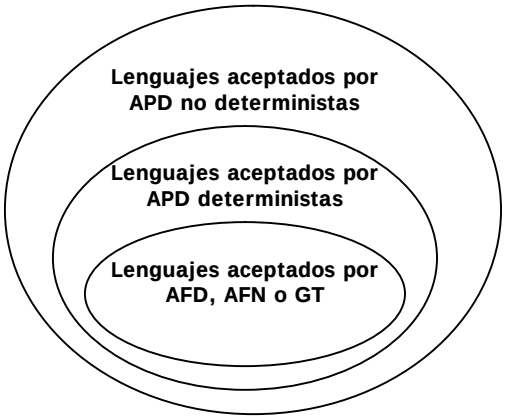
La consideración del lenguaje PALINDROME pronto nos convencerá que la nueva máquina APD necesitará ser indeterminista si queremos corresponderla con una GLC.

- **Un APD determinista es tal que para toda cadena de entrada tiene un único camino a través de la máquina.**
- **Un APD no determinista es aquel en el que en determinadas ocasiones tendremos que elegir entre diferentes posibles caminos a través de la máquina.**

Decimos que una cadena de entrada es aceptada por esta máquina si alguno de los caminos elegidos nos lleva a un estado ACCEPT. Si para todos los posibles caminos la cadena de entrada lleva a REJECT, entonces es rechazada.

Los APD que son equivalentes a GLC son la clase de los no deterministas.

Para los AF vimos que el no determinismo que nos lleva a AFN o GT, no aumenta el poder de la máquina para aceptar nuevos lenguajes. Para los APD esto es diferente. El siguiente diagrama de Venn muestra el poder relativo de estos tres tipos de máquinas.



Antes de dar un ejemplo concreto de un lenguaje aceptado por un APD no determinista que no puede ser aceptado por un APD determinista, consideremos un nuevo lenguaje.

Ejemplo

Reconocimiento de una GLC con un APD determinista

Introduzcamos el lenguaje PALINDROMEX, que acepta todas las palabras de la forma:

s X reverse(s)

Donde s es cualquier cadena de la forma (a+b)*

Las palabras aceptadas por este lenguaje son:

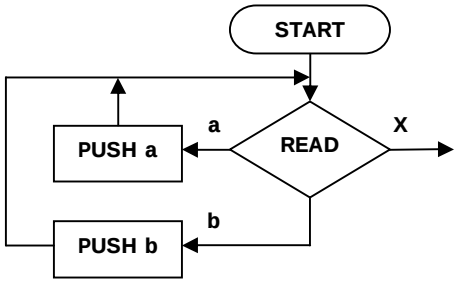
{ X aXa bXb aaXaa abXba baXab bbXbb aaaXaaa aabXbaa .....}

Todas estas palabras son palíndromos ya que se leen igual hacia adelante o atrás.

Contienen una X que marca el centro de la palabra. Podemos construir un APD determinista que acepte el lenguaje PALINDROMEX. Sorprendentemente tiene la misma estructura básica que el APD que teníamos para el lenguaje $\{a^n b^n\}$.

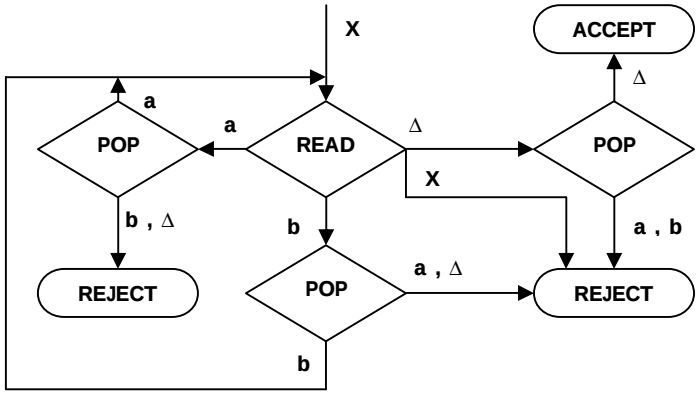
En la primera parte de la máquina el STACK es cargado con las letras de la cadena de entrada. Por conveniencia, cuando leemos una X sabemos que hemos llegado a la mitad de la entrada. Podemos empezar a comparar la mitad que queda en la cadena de entrada con la mitad que está almacenada en el STACK.

Comenzamos almacenando la primera mitad de la cadena en el STACK con la siguiente parte de la máquina:



Si leemos una a, insertamos una a, si leemos una b, insertamos una b, hasta que encontramos una X en la CINTA.

Cuando leemos la X, no la insertamos en el STACK. Se usa para transferirnos a la fase 2. Aquí es donde comparamos la CINTA con el STACK. Para alcanzar el estado ACCEPT, deben ser iguales carácter a carácter hasta hallar los blancos.



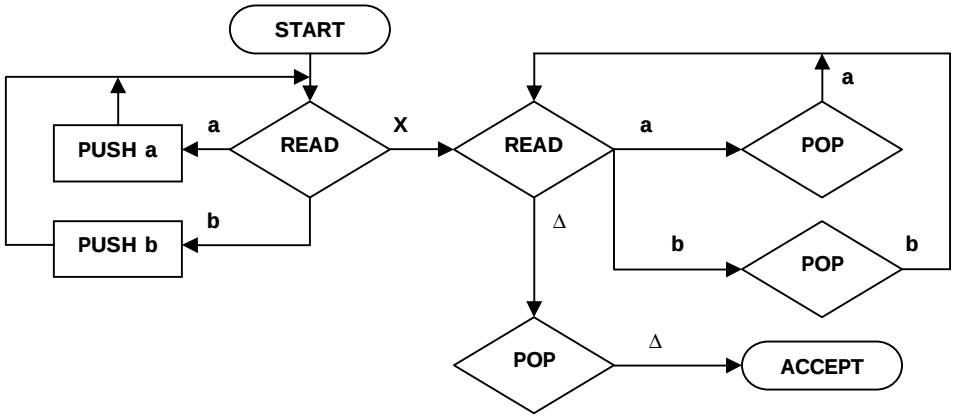
La máquina diseñada es determinista. El alfabeto de entrada aquí es Σ = {a, b, X}, tal que cada estado READ tiene 4 aristas que salen de él.

El alfabeto del STACK tiene dos letras Γ = {a, b}, tal que cada POP tiene 3 aristas que salen de él.

Para cada READ y cada POP hay una única dirección que la entrada puede tomar. Cada cadena de la CINTA genera un único camino a través de este APD.

Podemos graficar un esquema menos complicado para este APD sin los estados REJECT si no nos importa que al ingresar cadenas de entrada inválidas se bloquee el autómata al no tener camino para seguir.

El APD completo (sin REJECT) se dibuja aquí:



Ejemplo

Reconocimiento de una GLC con un APD no determinista

Ahora consideremos que clase de APD podría aceptar el lenguaje ODD-PALINDROME.

Este es el lenguaje que acepta todas las cadenas de a y b que son palíndromos y tienen un número impar de letras. Las palabras en este lenguaje son igual que en PALINDROMEX pero no poseen la letra X en el centro ya que ha cambiado por una a o b.

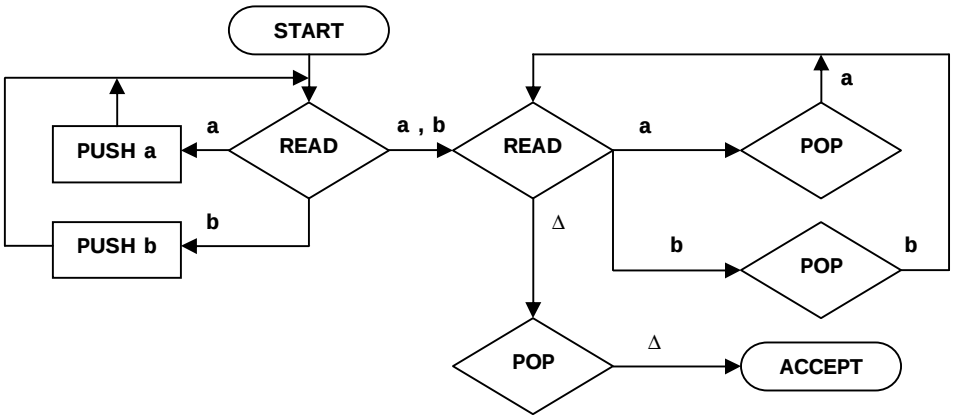
ODDPALINDROME = { a b aaa aba bab bbb}

El problema aquí es que no está la letra del centro, por lo que es más difícil reconocer donde termina la primera mitad y donde comienza la segunda mitad de la cadena.

En realidad, esto es imposible. Un APD, igual que un AF lee la entrada en forma secuencial de izquierda a derecha y no tiene idea de cuantas letras quedan por leer.

Pero no estamos completamente perdidos. El algoritmo puede cambiarse para ajustarse a nuestras necesidades introduciendo no determinismo.

Consideremos:



Esta máquina es la misma que la máquina anterior excepto que ha cambiado la X por a,b.

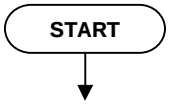
Esta máquina ahora es no determinista ya que el estado READ izquierdo tiene dos caminos para elegir en el caso de la letra a y dos caminos para la letra b.

Si elegimos el camino correcto, en el momento justo la cadena de entrada será aceptada y en caso contrario rechazada.

Definición

Un Autómata Push-Down (APD) es una colección de 8 cosas:

1. **Un alfabeto Σ de letras de entrada.**
2. **Una CINTA de entrada (infinita en una dirección). Inicialmente la cadena de letras de entrada se ubica en la CINTA comenzando en la celda i. El resto de la CINTA tiene blancos.**
3. **Un alfabeto Γ de caracteres del STACK.**
4. **Un STACK pushdown (infinito en una dirección). Inicialmente el STACK está vacío (contiene todos blancos).**
5. **Un estado START que contiene solo aristas de salida, no aristas de entrada.**



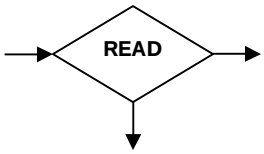
6. **Estados de Detención de dos clases: algunos ACCEPT y algunos REJECT. Tienen aristas de entrada y no aristas de salida.**

7. **Un número finito de estados PUSH sin ramificación que introducen caracteres en la parte superior del STACK. Son de la forma:**

donde X es cualquier letra en Γ

8. **Un número finito de estados con ramificación de dos clases:**

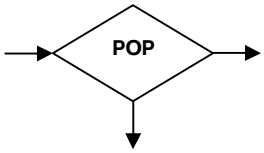
i) Estados que leen la próxima letra no usada de la CINTA



que pueden tener aristas de salida etiquetadas con letras de Σ y el carácter blanco Δ, sin restricciones de duplicación de etiquetas y sin obligación de que haya una etiqueta para cada letra de Σ, o una Δ.

y

ii) Estados que leen el carácter superior del STACK



que pueden tener aristas de salida etiquetadas con las letras de Γ y el carácter blanco Δ, nuevamente sin restricciones.

Ejecutar una cadena de letras de entrada en un APD significa comenzar desde el estado START y seguir las aristas no etiquetadas y las etiquetadas que generen un camino a través del diagrama. El camino terminará en un estado de detención o hará crash en algún estado que no posea ramificación ante la lectura de un carácter de la entrada desde la CINTA.

Una cadena de entrada con un camino que termina en un estado ACCEPT se dice que es aceptada.

Una cadena de entrada que sigue una lista de caminos se dice que es aceptada si al menos uno de esos caminos llega a un estado ACCEPT.

El conjunto de todas las cadenas aceptadas por un APD se llama el Lenguaje Aceptado por el APD o el Lenguaje Reconocido por el APD.

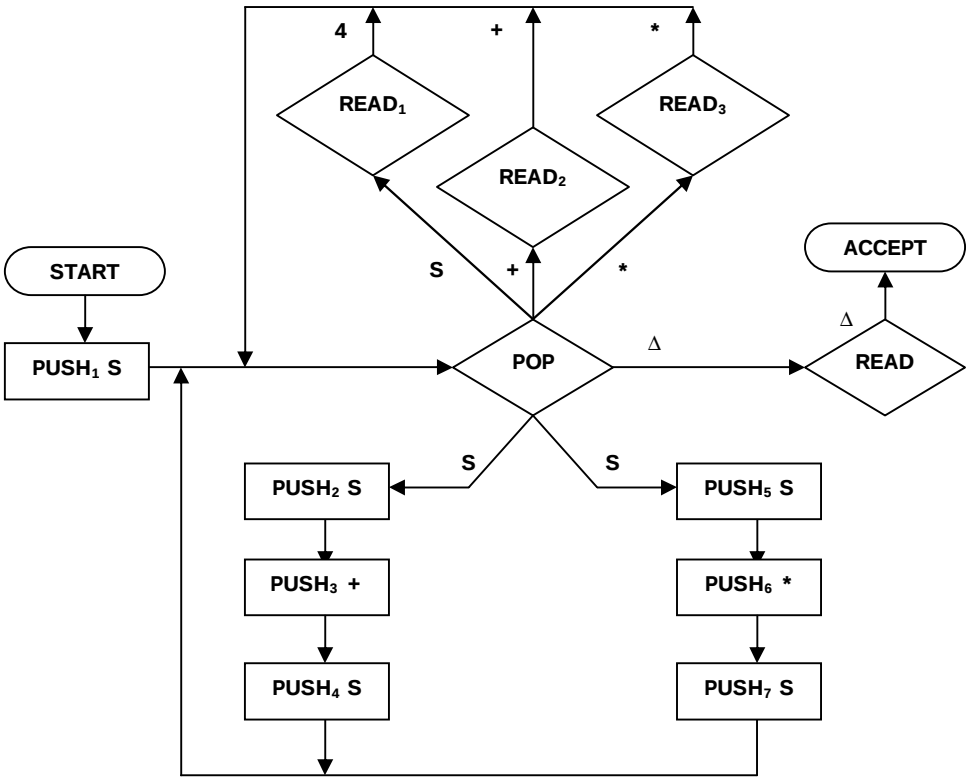
Ejemplo

Considere el lenguaje generado por la GLC:

S à S + S | S * S | 4

Los terminales son +, * y 4 y el único no terminal es S.

El siguiente APD acepta este lenguaje:



Máquinas de Turing

Lenguaje definido por	Aceptor Correspondiente	Ejemplo de aplicación
Expresión Regular	Autómata Finito Grafo de Transición	Editores de texto, circuitos secuenciales
Gramática Libre de Contexto	Autómata PushDown	Sentencias de lenguajes de programación Compiladores
Gramática de tipo 0	Máquina de Turing Máquina de Post	Computadoras

Vemos en la entrada inferior a la derecha de la tabla que estamos por cumplir la promesa hecha en la introducción. Pronto daremos un modelo matemático para la familia completa de computadoras modernas.

Este modelo nos habilitará no solo para estudiar algunas limitaciones teóricas sobre las tareas que pueden realizar las computadoras, también será un modelo que podemos usar para mostrar aquellas operaciones que pueden ser hechas por computadoras.

Hay una progresión definida en las filas de la tabla. Todos los lenguajes regulares son lenguajes libres de contexto, y veremos que todos los lenguajes libres de contexto son lenguajes de Máquinas de Turing.

Históricamente, el orden de invención de estas ideas es:

- **Los lenguajes regulares y AF fueron desarrollados por Kleene, Mealy, Moore, Rabin y Scout en los años 50.**
- **Las GLC y los APD fueron desarrollados posteriormente, por Chomsky, Oettinger, Schützenberger, y Evey, mayormente en los años 60.**
- **Las máquinas de Turing y su teoría fueron desarrolladas por Alan Mathison Turing y Emil Post entre los años 30 y 40.**

Sorprende que estas fechas están fuera de orden ya que el trabajo de Turing fue previo a la invención de la computadora misma.

Turing no estaba analizando un espécimen que tenía frente a sí, estaba inventando la bestia. Esta es otra demostración de que no hay nada más práctico que una buena teoría abstracta.

Ya que las Máquinas de Turing serán nuestro último modelo de computadoras, necesariamente tendrán capacidades de salida. La salida es muy importante, tan importante que un programa sin sentencias de salida parecería totalmente inútil porque nunca daría a los humanos los resultados de sus cálculos. Probablemente hayamos escuchado decir que la única sentencia que ningún programa puede dejar de tener es una sentencia de salida.

Esto no es exactamente cierto. Considere el siguiente programa (escrito en ningún lenguaje particular):

```
READ X
IF X=1 THEN END
IF X=2 THEN DIVIDE X BY 0
IF X>2 THEN GOTO STATEMENT 4
```

Asumamos que la entrada es un entero positivo.

- **Si el programa termina naturalmente, entonces sabemos que X vale 1.**
- **Si termina creando overflow o es interrumpido por algún mensaje de error de cálculo ilegal (crash), entonces sabemos que X vale 2.**
- **Si encontramos que nuestro programa terminó porque excedía nuestro tiempo de proceso en la computadora, entonces sabemos que X es mayor que 2.**

Veremos en un momento que las mismas ideas se aplican a Máquinas de Turing.

Definición

Una Máquina de Turing, denotada MT, es una colección de seis cosas:

1. **Un alfabeto Σ (sigma) de letras de entrada, que por cuestión de claridad no contiene el símbolo blanco Δ.**
2. **Una CINTA dividida en una secuencia de celdas numeradas cada una conteniendo un carácter o un blanco. La palabra de entrada se presenta a la máquina una letra por celda comenzando por la celda de más a la izquierda, llamada celda i. El resto de la CINTA se completa inicialmente con blancos Δ.**

3. **Una CABEZA de CINTA que puede en un paso leer el contenido de una celda de la CINTA, reemplazarlo con algún otro carácter, y reposicionarse en la próxima celda a la derecha o a la izquierda de aquel que ha sido recién leído.**

   Al principio del proceso, la CABEZA de CINTA siempre empieza leyendo la entrada de la celda i. La CABEZA de CINTA nunca puede moverse hacia la izquierda de la celda i. Si se le dan órdenes de hacerlo, la máquina falla.
4. **Un alfabeto Γ (gamma), de caracteres que pueden ser impresos en la CINTA por la CABEZA de CINTA. Estos pueden incluir Σ. Aún cuando permitamos a la CABEZA de CINTA imprimir un Δ llamamos a esto borrar y no incluir el blanco como una letra del alfabeto Γ.**
5. **Un conjunto finito de estados que incluye exactamente un estado START desde el cual comienza la ejecución (y en el que podemos reingresar durante la ejecución) y algunos (quizá ninguno) estados de HALT que causan que la ejecución termine**

cuando se ingresa en ellos. Los otros estados no tienen funciones, solo nombres: 1, 2, 3, ...

6. **Un programa, que es un conjunto de reglas que nos dicen, en base a la letra que la CABEZA de CINTA ha leído, como cambiar de estado, que imprimir, donde mover la CABEZA de CINTA.**

   Diseñamos al programa como una colección de aristas direccionadas que conectan estados. Cada arista está etiquetada con un triplete de información:

   (letra, letra, dirección)

   - **La primera letra (Δ o una letra de Σ o Γ) es el carácter que la CABEZA de CINTA lee desde la celda a la que está apuntando.**
   - **La segunda letra (también Δ o una letra de Γ) es lo que la CABEZA de CINTA imprime en la celda antes de dejarla.**
   - **El tercer componente, la dirección, le dice a la CABEZA de CINTA si moverse una celda a la derecha, R, o una celda a la izquierda, L.**

No se hace ninguna estipulación sobre si cada estado tiene una arista que sale de él para cada letra posible en la CINTA.

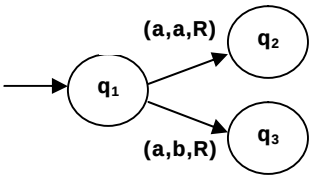
Si estamos en un estado y leemos una letra que no ofrece ninguna elección de camino posible hacia otro estado, fallamos; esto significa que terminamos la ejecución en forma fallida.

Para terminar la ejecución de una entrada exitosamente debemos llegar a un estado de HALT. La palabra en la CINTA de entrada se dice que es aceptada por la MT.

Un fallo también ocurre cuando estamos en la primera celda de la CINTA y tratamos de mover la CABEZA de CINTA hacia la izquierda.

Por definición *todas las máquinas de Turing son deterministas*. Esto significa que no hay un estado q que tenga dos o mas aristas que salen de él etiquetadas con la misma primera letra.

Por ejemplo:



no está permitido.

Una máquina Turing es un mecanismo computacional hipotético formado por una unidad de control de estado finito acoplada a una cinta infinita. Cada paso en una computación de máquina Turing consiste en escribir un símbolo en la cinta, correr la cinta hacia la izquierda o hacia la derecha y adquirir un nuevo estado interno. La acción particular que se realiza en cada paso está determinada por el estado actual de la máquina y el símbolo rastreado en ese momento. A causa de la naturaleza simple y mecánica de sus computaciones, las máquinas Turing pueden ciertamente ser consideradas como algoritmos eficaces.

Hay varias maneras en que el modelo básico de la máquina Turing puede ser modificado sin que se afecten sus capacidades últimas de computación. Entre las generalizaciones del modelo que no aumentan su poder computacional están el uso de cintas con varias pistas, el uso de más de una cabeza lectora o el uso de más de una cinta. Entre las restricciones que no disminuyen el poder del modelo básico están el uso de cintas infinitas en sólo una dirección, la limitación a un solo símbolo no blanco o la limitación a solo dos estados internos. Así, aunque una variante del modelo de máquina Turing puede resultar más eficiente o conveniente que otra para una aplicación dada, las capacidades fundamentales de las máquinas Turing no dependen de la variante particular adoptada. (Hennie 77, pp. 23-24)

.....

Una *máquina Turing* consiste en una cinta indefinidamente larga acoplada a una unidad de control finita.... La cinta, que actúa como la memoria de la máquina, está dividida en cuadrados. Cada cuadrado puede ser estampado con un solo símbolo del alfabeto finito designado o puede estar en blanco.... La unidad de control puede correr la cinta para los dos lados al través de la cabeza lectora, pero en un momento particular la cabeza lectora puede examinar, o *rastrear*, solamente un cuadrado de la cinta.

La operación de una máquina Turing se caracteriza por la siguiente secuencia de eventos. La máquina recibe inicialmente una cinta en la cual algún número finito de cuadrados están estampados con símbolos y el resto permanece en blanco. Un determinado cuadrado de la cinta es designado para estar inicialmente bajo la cabeza lectora, y la unidad de control es forzada a asumir un cierto *estado inicial* predesignado. La máquina pasa entonces por una *computación* que consiste en una serie de movidas básicas. Esta computación puede continuar indefinidamente, o puede terminar después de un número finito de movidas. Si de hecho termina, el patrón de símbolos que queda en la cinta se toma como el resultado de la computación.

Una máquina Turing comienza con una cinta inicial y realiza una transformación en esa cinta. Si la máquina se detiene, el resultado de la transformación es el contenido de la cinta en el momento del detenimiento. Si la máquina nunca se detiene, el resultado no está definido. (Kain 72, pp. 83-86)

.....

En su artículo de 1936, A. M. Turing define la clase de máquinas abstractas que hoy llevan su nombre. Una *máquina Turing* es una máquina de estado finito asociada con una clase especial de ambiente –su *cinta*– en el cual puede guardar (y más tarde recobrar) secuencias de símbolos.... Son máquinas muy simples. En cada momento la parte que es una máquina de estado finito obtiene su estímulo de *entrada* mediante la *lectura* del símbolo escrito en cierto punto a lo largo de la cinta. La *respuesta* de la máquina puede *cambiar tal símbolo* y también mover la máquina una pequeña distancia en alguna de las dos direcciones a lo largo de la cinta. El resultado es que el estímulo para el nuevo ciclo de operación vendrá de un "cuadrado" distinto de la cinta, y la máquina puede así leer un

símbolo que haya sido escrito ahí mucho antes. Esto significa que la máquina tiene acceso a una especie de memoria exterior rudimentaria, además de la que es suplida por su parte de estado finito. Y como no ponemos límite en la cantidad de cinta disponible, esta memoria tiene en efecto una capacidad infinita.... (Minsky 67, p. 107)

.....

Una máquina Turing es una máquina de estado finito asociada con un almacén externo o memoria. Esta memoria tiene la forma de una secuencia de *cuadrados*, marcados en una *cinta* lineal. La máquina está acoplada a una cinta mediante una *cabeza*, situada en cada momento sobre algún cuadrado de la cinta. La cabeza tiene tres funciones, todas las cuales se ejercen en cada ciclo de operación de la máquina de estado finito. Estas funciones son *leer* el cuadrado de la cinta rastreado en ese momento, *escribir* en el cuadrado rastreado y *mover* la máquina a un cuadrado adyacente (que pasa a ser el cuadrado rastreado en el siguiente ciclo de operación). (Minsky 67, p. 117)

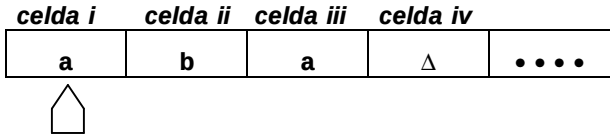
.....

Lo que está en la cinta cuando la máquina para dependerá de un modo complicado de lo que la cinta tenía al comenzar. Por eso podemos decir que el resultado de la computación presente en la cinta es una *función* de la entrada. Exactamente cuál función sea, eso dependerá, por supuesto, de la máquina Turing que se use. Entonces, podemos pensar que una máquina Turing *define* una función o la *computa*, o incluso que *es una función*. (Minsky 67, p. 132)

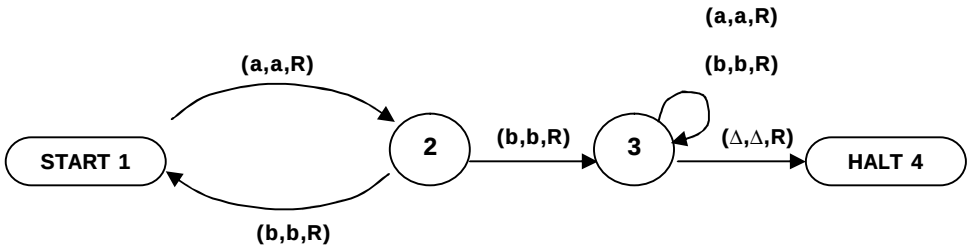
Ejemplo

Reconocimiento de un Lenguaje Regular

La siguiente es la CINTA de una máquina de Turing que correrá sobre la entrada aba.



El programa para esta MT es dado como un grafo dirigido con aristas etiquetadas como se muestra a continuación:



Note que el loop en el estado 3 tiene dos etiquetas. Las aristas desde el estado 1 al estado 2 podrían haber sido dibujadas como una arista con dos etiquetas.

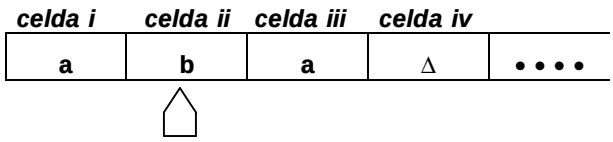
Comenzamos, como siempre, con la CABEZA de CINTA leyendo la celda i y el programa en el estado START, que está etiquetado como estado 1. Escribimos esto como:

1
aba

El número de arriba es el número del estado en el que estamos. Debajo se encuentran los contenidos de la CINTA. Es posible encontrar símbolos Δ dentro de esta cadena. Subrayamos el carácter de la celda que está por ser leída.

En este punto de nuestro ejemplo, la CABEZA de CINTA lee la letra a y seguimos por la arista (a,a,R) hacia el estado 2. Las instrucciones de esta arista a la CABEZA de CINTA son *"lea una a, imprima una a, muévase a la derecha"*.

La CINTA ahora se ve como:



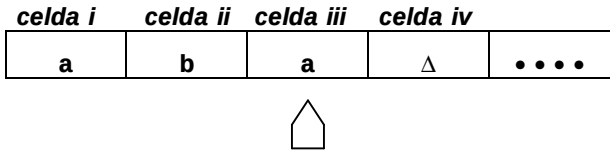
Podemos registrar el proceso de ejecución escribiendo:

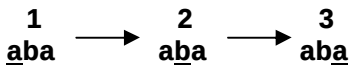
1 ⟶ **2**
aba **ab**a

Ahora estamos en el estado 2. Seguimos la arista etiquetada por (b,b,R). La CABEZA de CINTA reemplaza la b con una b y se mueve a la derecha una celda.

La idea de reemplazar una letra consigo misma puede parecer tonta, pero unifica la estructura de las instrucciones de las Máquinas de Turing, para que todas tengan el mismo formato (las de reemplazo y las de desplazamiento).

En este momento la situación es la siguiente:





Pasamos al estado 3, leemos una a, y hacemos un loop.

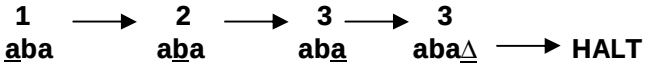
Finalmente leemos un Δ y nos movemos al estado 4.

La cadena de entrada aba ha sido aceptada por esta Máquina de Turing.

Esta MT particular no cambia ninguna de las letras de la CINTA, por lo que al final de la ejecución la CINTA permanece:

abaΔ

En resumen, la ejecución completa puede ser expresada como la siguiente traza:



Qué cadenas de entrada acepta esta MT?

Cualquier letra, a o b, nos lleva al estado 2. Desde allí al estado 3 leemos una letra b. Finalmente se genera un loop hasta que la CABEZA de CINTA encuentra una Δ.

El lenguaje aceptado por esta MT está formado por todas las palabras que comienzan con una a o b, luego tienen una b y finalmente tienen cualquier número de a o b.

Este es un lenguaje regular porque puede ser definido también por la expresión regular:

(a+b)b(a+b)*

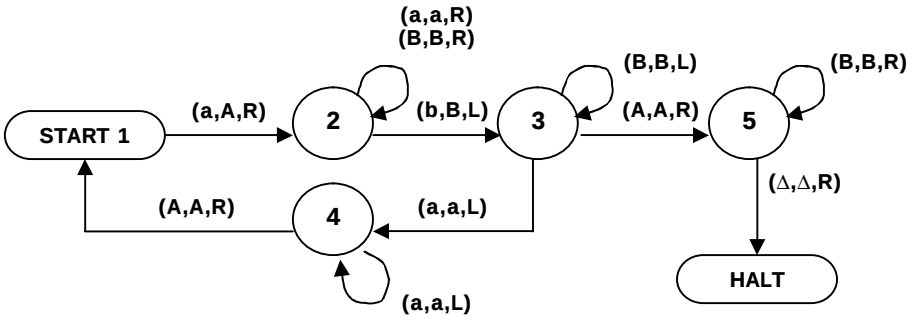
Esta MT es también una *reminiscencia de los AF*, ya que hace un solo paso sobre la cadena de entrada, moviendo su CABEZA de CINTA siempre hacia la derecha y nunca cambiando una letra que ha leído.

Ejemplo

Reconocimiento de una GLC, aceptada por un APD determinista

Considere la siguiente MT.

Esta MT acepta el lenguaje $\{a^n b^n$, con n>=1$\}$



Técnicamente, el alfabeto de entrada es Σ = {a , b} y el alfabeto de salida es Γ = {a , A , B}.

El siguiente cuadro muestra los contenidos de la CINTA en cada paso del procesamiento de la cadena aaabbb. Recuerde que durante el seguimiento la posición de la CABEZA de CINTA se indica mediante el subrayado de la letra que va a ser leída.

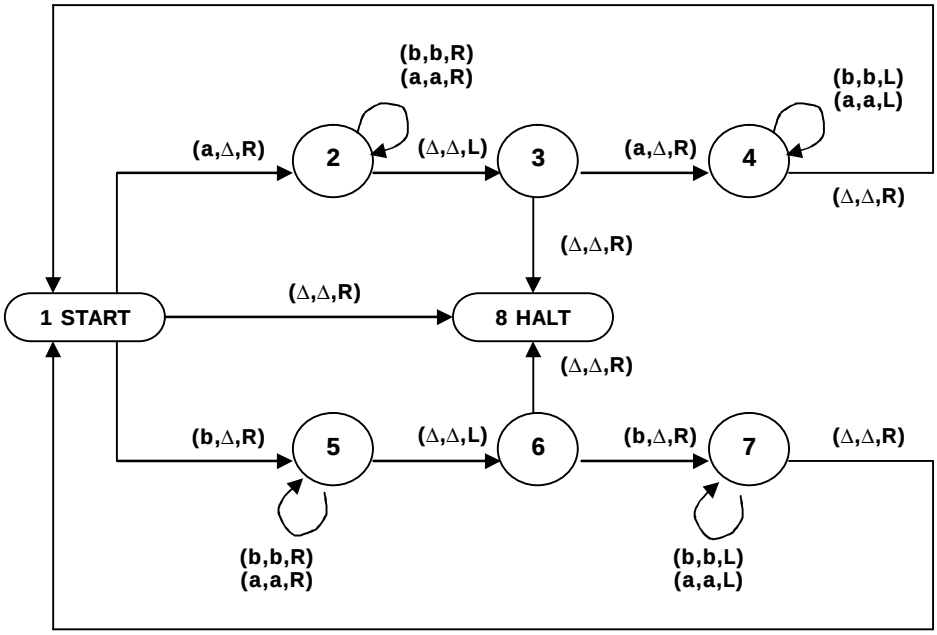
1° parte	2° parte
aaabbb	A**A**aBBb
A**a**abbb	AA**a**BBb
Aa**a**bbb	AAA**B**Bb
Aaa**b**bb	AAAB**B**b
Aa**a**Bbb	AAABB**b**
A**a**aBbb	AAAB**B**B
AaaBbb	AAA**B**BB
A**a**aBbb	AA**A**BBB
AA**a**Bbb	AAA**B**BB
AAa**B**bb	AAAB**B**B
AAaB**b**b	AAABB**B**
AAa**B**Bb	AAABBB**Δ**
AA**a**BBb	HALT

Ejemplo

Reconocimiento de una GLC, aceptada por un APD no determinista

Considere la siguiente MT.

Esta MT acepta el lenguaje PALINDROME



El siguiente cuadro muestra los contenidos de la CINTA en cada paso del procesamiento de la cadena ababa. Recuerde que durante el seguimiento la posición de la CABEZA de CINTA se indica mediante el subrayado de la letra que va a ser leída.

1° parte		2° parte	
1	**a**baba	5	Δ**b**abΔ
2	Δ**b**aba	5	ΔΔ**a**bΔ
2	Δb**a**ba	5	ΔΔa**b**Δ
2	Δba**b**a	5	ΔΔab**Δ**
2	Δbab**a**	6	ΔΔa**b**Δ
2	Δbaba**Δ**	7	ΔΔ**a**ΔΔ
3	Δbab**a**	7	Δ**Δ**aΔΔ
4	Δba**b**Δ	1	ΔΔ**a**ΔΔ
4	Δb**a**bΔ	2	ΔΔ**Δ**ΔΔ
4	Δ**b**abΔ	3	ΔΔΔ**Δ**Δ
4	**Δ**babΔ	8	**HALT**

Nuestro primer ejemplo era solamente un AF convertido, y el lenguaje que aceptaba era Regular.

El segundo ejemplo aceptaba un lenguaje Libre de Contexto y No Regular y la MT dada empleaba alfabetos separados para escribir y leer.

La tercer máquina aceptaba un lenguaje que era también Libre de Contexto, pero que solo podía ser aceptado por un APD No Determinístico, sin embargo la MT diseñada es Determinística.

Hemos visto que podemos usar la CINTA para más que un STACK PUSHDOWN. En los últimos dos ejemplos recorrimos hacia delante y atrás la CINTA para hacer observaciones y cambios en la cadena. Veremos más adelante que la CINTA puede ser usada para más tareas aún. Puede ser usada como un espacio de trabajo para cálculo y salida.

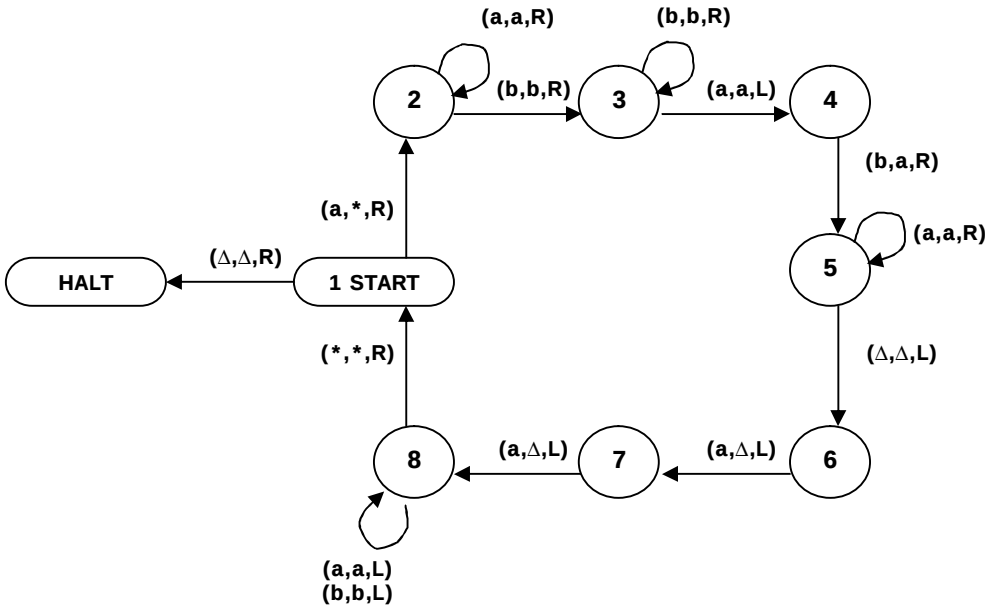
Ejemplo

Reconocimiento de una Gramática Dependiente del Contexto,

que no puede ser aceptada por un APD

Considere la siguiente MT.

Esta MT acepta el lenguaje $\{a^n b^n a^n$, con $n>=0\}$



El siguiente cuadro muestra los contenidos de la CINTA en cada paso del procesamiento de la cadena aaabbbaaa. Recuerde que durante el seguimiento la posición de la CABEZA de CINTA se indica mediante el subrayado de la letra que va a ser leída.

1° parte		2° parte		3° parte	
1	**a**aabbbaaa	8	*aab**b**aa	8	**ab**a
2	*a**a**bbbaaa	8	*aa**b**baa	8	**ab**a
2	*a**a**bbbaaa	8	*a**a**bbaa	8	**a**ba
2	*aa**b**bbaaa	8	*a**a**bbaa	8	**_aba
3	*aab**b**baaa	8	*aabbaa	1	**a**ba
3	*aabb**b**aaa	1	*a**a**bbaa	2	***b**a
3	*aabbb**a**aa	2	**a**bbaa	3	***b**a
4	*aabb**b**aaa	2	**a**bbaa	4	***b**a
5	*aabba**a**aa	3	**ab**baa	5	***a**a
5	*aabbaa**a**a	3	**abb**aa	5	***aaΔ
5	*aabbaaa**a**	4	**ab**baa	6	***a**a
5	*aabbaaaaΔ	5	**aba**aa	7	***a**
6	*aabbaaa**a**	5	**abaa**a	8	***_
7	*aabbaa**a**	5	**abaaaΔ	1	***Δ
8	*aabba**a**	6	**abaa**a	Halt	***ΔΔ
8	*aabb**a**a	7	**aba**a		

Luego de diseñar la máquina y siguiendo el recorrido son obvias algunas consideraciones:

1. **Las únicas palabras aceptadas son de la forma {$a^n b^n a^n$, con n>=0}.**
2. **Cuando la máquina se detiene, la CINTA contiene tantos * como letras b había en la entrada.**
3. **Si la entrada era $a^n b^n a^n$, la CABEZA de CINTA estará en la celda (n + 2) al detenerse la máquina.**

Este ejemplo sugiere que las MT son más poderosas que los APD, ya que los APD no pueden aceptar este lenguaje.