DESARROLLO DE UN ENTORNO DE SIMULACIÓN PARA AUTÓMATAS DETERMINISTAS

CAROLINA GONZÁLEZ NARANJO
CÉSAR AUGUSTO MONTOYA ROMÁN

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS: ELÉCTRICA, ELECTRÓNICA, FÍSICA, CIENCIAS DE LA COMPUTACIÓN
PROGRAMA DE INGENIERÍA ELÉCTRICA
PEREIRA
2008

DESARROLLO DE UN ENTORNO DE SIMULACIÓN PARA AUTÓMATAS DETERMINISTAS

CAROLINA GONZÁLEZ NARANJO
CÉSAR AUGUSTO MONTOYA ROMÁN

PROYECTO DE GRADO PRESENTADO COMO REQUISITO PARCIAL PARA OPTAR EL TÍTULO DE INGENIERO ELECTRICISTA.

DIRECTOR
INGENIERO MAURICIO HOLGUÍN LONDOÑO

UNIVERSIDAD TECNOLÓGICA DE PEREIRA
FACULTAD DE INGENIERÍAS: ELÉCTRICA, ELECTRÓNICA, FÍSICA, CIENCIAS DE LA COMPUTACIÓN
PROGRAMA DE INGENIERÍA ELÉCTRICA
PEREIRA
2008

Pereira, Noviembre 2008.

Nota de aceptación

_____
_____
_____
_____

_____
Jurado

AGRADECIMIENTOS

A Dios y a nuestros padres por darnos la oportunidad de ser, por el apoyo y la motivación de seguir adelante.

Al Ingeniero Mauricio Holguín Londoño por ser nuestro tutor y guía en este proceso, por estar siempre dispuesto a ayudarnos a encontrar una solución a las dificultades, lo cual nos ha permitido alcanzar nuestra meta.

AL M.Sc Germán Andrés Holguín Londoño por sus correcciones, recomendaciones y sugerencias.

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

NOTACIONES

$\vee$: Disyunción lógica
$\wedge$: Conjunción lógica
$\{\}$: Conjunto vacío
$\{\ \}$: Notación constructora de conjuntos
$|$: Descriptor, se lee "… tal que…"
$\in$: Pertenencia de conjuntos
$\notin$: No pertenencia de conjuntos
$\cap$: Intersección de conjuntos
$\cup$: Unión de conjuntos
$\subset$: Inclusión de conjuntos
$\not\subset$: Negación de inclusión de conjuntos
$\subseteq$: Subconjunto propio

1. INTRODUCCIÓN A LENGUAJES

1.1 ALFABETOS Y CADENAS

El símbolo es una representación distinguible de cualquier información; es una entidad indivisible.

Un alfabeto Σ es un conjunto finito y no vacío de símbolos, con los cuales es posible formar secuencias o cadenas de caracteres, conocidas como palabras.

Una cadena sobre el alfabeto Σ, es una sucesión de caracteres tomados de Σ. La longitud de una cadena c, es el número de caracteres que la forman; se denota por $|c|$ o por $long(c)$. La cadena vacía es la cadena sin elementos y se representa por ε.

La concatenación de las cadenas x y y es aquella que resulta de añadir a la cadena x la cadena y; y se denota por xy. La concatenación de palabras es asociativa, es decir $(xy)z = x(yz)$, pero no conmutativa. La longitud de una concatenación cumple la propiedad $|uv| = |u| + |v|$.[1]

La clausura sobre el alfabeto, Σ^*, es el conjunto de todas las posibles cadenas sobre un alfabeto Σ; el conjunto Σ^*, puede ser infinito, pero enumerable.[2]

1.2 LENGUAJES

Un lenguaje L sobre un alfabeto Σ es un subconjunto de Σ^*, es decir un conjunto de cadenas sobre Σ.

$$L \subseteq \Sigma^*$$

1.2.1 Operaciones sobre lenguajes

- Unión de lenguajes: La unión $L_1 \cup L_2$ de dos lenguajes L_1 y L_2 es el lenguaje formado por las cadenas que pertenecen a L_1 y las cadenas que pertenecen a L_2. La unión de lenguajes constituye un único lenguaje.[5]
  - $L_1 \cup L_2 = \{x : x \in L_1 \vee x \in L_2\}$

[1] La prueba de estas propiedades requiere de una definición formal de las secuencias de caracteres, lo que desviaría el tema.

- Intersección de lenguajes: La intersección $L_1 \cap L_2$ de dos lenguajes L_1 y L_2 es el lenguaje formado por las cadenas que pertenecen simultáneamente a L_1 y a L_2.

$$L_1 \cap L_2 = \{x : x \in L_1 \in x \in L_2\}$$

- Diferencia de lenguajes: La diferencia $L_1 - L_2$ de dos lenguajes L_1 y L_2 es el lenguaje formado por las cadenas que pertenecen a L_1 y no pertenecen a L_2.

$$L_1 - L_2 = \{x : x \in L_1 \wedge x \notin L_2\}$$

- Complementación de un lenguaje: La complementación de un lenguaje sobre un alfabeto Σ es el lenguaje formado por las cadenas que pertenecen al lenguaje universal, Σ^*, pero que no pertenecen al lenguaje L.Es decir, está formado por las cadenas que pertenecen a $\Sigma - L$. [5]

- Concatenación: La concatenación $L_1 \cdot L_2$ de dos lenguajes L_1 y L_2 es el lenguaje formado por las cadenas construidas mediante la concatenación de una cadena de L_1 con una cadena de L_2. La concatenación de lenguajes constituye un monoide.[2]

$$L_1 \cdot L_2 = \{xy : x \in L_1 \wedge y \in L_2\}$$

- Iteración, cierre o clausura de un lenguaje: El cierre de un lenguaje L, o cerradura de Kleene, se representa por L^* y se define como: [2]

$$L^* = \{\varepsilon\} \cup L \cup L^2 \cup L^3 ...$$

- Σ^* es la cerradura de Kleene del alfabeto, tomando los símbolos de éste como palabras de una letra. [5]

1.3 JERARQUÍA DE CHOMSKY

Las clases de lenguajes son conjuntos de lenguajes que comparten una cierta propiedad dada, es decir, conjuntos de conjuntos de secuencias de símbolos.

[2] Este proceso no terminaría nunca, pues L^* es infinito para cualquier L que tenga al menos un elemento.

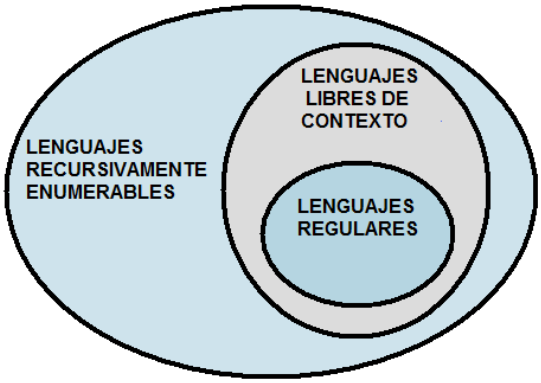


Figura 1 Jerarquía de Chomsky

Chomsky, propuso una jerarquía de lenguajes, donde las clases más complejas incluyen a las más simples: [6]

- Lenguajes Regulares: la clase más pequeña, e incluye a los lenguajes más simples, se llaman así porque sus palabras contienen regularidades o repeticiones de los mismos componentes, estos lenguajes son finitos y pueden ser resultado de la unión o concatenación de otros lenguajes regulares.

- Lenguajes Libres de Contexto: incluyen a los Lenguajes Regulares. Estos lenguajes son generados por las gramáticas Libres de Contexto donde un no-terminal (variable) puede ser sustituido por una cadena de terminales (no variables o constantes) y/o no-terminales, sin tener en cuenta el contexto en el que ocurra.

- Lenguajes Recursivamente Enumerables: incluyen a los Libres de Contexto y por lo tanto a los Lenguajes Regulares. Es un lenguaje formal, conjunto de palabras de longitud finita formadas a partir de un alfabeto finito, para el cual existe una máquina de Turing, sección 4.2.1, que acepta y se detiene con cualquier cadena

Todas estas clases de lenguajes son representables de manera finita, mediante cadenas de caracteres, y cada una está asociada a un tipo de autómata, sección 1.5, capaz de procesar estos lenguajes.

Hay más lenguajes que posibles representaciones finitas, por lo cual hay lenguajes más allá de los Recursivamente Enumerables. Desde un punto de vista práctico, los lenguajes más útiles son aquellos que tienen una representación finita, pues los demás lenguajes son sólo de interés teórico. [2]

1.4 GRAMÁTICA FORMAL

Una gramática es un conjunto de reglas para formar correctamente las frases de un lenguaje. Una regla es una expresión de la forma $\alpha \to \beta$, donde tanto α como β son cadenas de símbolos donde pueden aparecer tanto elementos del alfabeto Σ, llamados constantes, como otros elementos llamados variables y $\to$ es la indicación de reemplazar α por β

Para aplicar una gramática se parte de una variable, llamada símbolo inicial, y se aplican repetidamente las reglas gramaticales, hasta que ya no haya variables en la palabra. En ese momento se dice que la palabra resultante es generada por la gramática, o en forma equivalente, que la palabra resultante es parte del lenguaje de esa gramática.

Las gramáticas formales definen un lenguaje describiendo cómo se pueden generar las cadenas del lenguaje.[5]

Una gramática formal es una cuádrupla $G = (\Sigma, V, R, S)$ donde:

- Σ es un conjunto finito de símbolos terminales
- V es un conjunto finito de símbolos no terminales, variables
- R es un conjunto finito de reglas
- S es el símbolo inicial o axioma

1.5 AUTÓMATAS

Un autómata es un dispositivo que manipula cadenas de símbolos que se le presentan a su entrada, produciendo otras cadenas de símbolos a su salida. El autómata recibe los símbolos de entrada, uno detrás de otro, secuencialmente. El símbolo de salida que en un instante determinado produce un autómata, no sólo depende del último símbolo recibido a la entrada, sino de toda la secuencia o cadena, que ha recibido hasta ese instante.[13]

El estado de un autómata; es toda la información necesaria en un momento dado, para poder deducir, dado un símbolo de entrada en ese momento, cuál será el símbolo de salida.

Conocer el estado de un autómata es lo mismo que conocer toda la historia de símbolos de entrada, así como el estado inicial y el estado en que se encontraba el autómata al recibir el primero de los símbolos de entrada. El autómata tendrá un determinado número de estados, pudiendo ser infinitos, y se encontrará en uno u otro según sea la historia de símbolos que le han llegado.

La configuración de un autómata es su situación en un instante. El movimiento de un autómata es el paso de una configuración a otra. Si un autómata se encuentra en un estado determinado, recibe un símbolo también determinado, produce un símbolo de salida y efectúa un cambio o transición a otro estado, también puede quedarse en el mismo estado.

Los autómatas que son objeto de estudio durante el desarrollo de éste trabajo son abstracciones matemáticas que sólo incluyen el aspecto referente a las secuencias de eventos que ocurren, sin tomar en cuenta la parte física de la máquina, ni tampoco el tipo de movimientos que efectúa.

Clasificación

Una máquina de Turing es un autómata que se mueve sobre una secuencia lineal de datos. En cada instante la máquina puede leer un dato de la secuencia, generalmente un carácter y realiza ciertas acciones con base en una tabla que tiene en cuenta su estado actual (interno) y el último dato leído. Entre las acciones está la posibilidad de escribir nuevos datos en la secuencia; recorrer la secuencia en ambos sentidos y cambiar de estado dentro de un conjunto finito de estados posibles.

Autómata de Pila: Se puede ver como una máquina de Turing que sólo puede leer la cinta en un sentido, aunque puede utilizar una pila para almacenar lo que ha leído.

Autómata Finito: Se puede ver como una máquina de Turing que sólo puede leer la cinta en un sentido y no tiene ningún tipo de dispositivo de almacenamiento.

2. LENGUAJES REGULARES Y AUTÓMATA DE ESTADOS FINITOS

Durante el desarrollo de ésta sección se estudian las máquinas abstractas más simples, los autómatas finitos, las cuales están en relación con los lenguajes regulares.

2.1 LENGUAJES REGULARES

Un lenguaje regular es un tipo de lenguaje formal que puede ser reconocido por un autómata finito, se llama así porque sus palabras contienen regularidades o repeticiones de los mismos componentes; puede ser generado mediante una gramática regular y descrito por una expresión regular.

Los Lenguajes Regulares son los más simples y restringidos dentro de la jerarquía de Chomsky. [6]

Un lenguaje L es regular si y sólo si se cumple una de las siguientes condiciones:

- L es finito
- L es la unión o la concatenación de otros lenguajes regulares R_1 y R_2 $L = R_1 \cup R_2$ o $L = R_1 \cdot R_2$ respectivamente
- L es la cerradura de Kleene de algún lenguaje regular, $L = R*$

Por ejemplo el lenguaje L de palabras formadas por a y b, pero que empiezan con a, como aab, ab, a, $abaa$, es un lenguaje regular.

El alfabeto es $\Sigma = \{a,b\}$ .El lenguaje L puede ser visto como la concatenación de una a con cadenas cualesquiera de a y b; éstas últimas son los elementos de $\{a,b\}*$, mientras que el lenguaje que sólo contiene la palabra a es $\{a\}$; ambos lenguajes son regulares. [3] Entonces su concatenación es $\{a\}\{a,b\}*$, que también es regular.

2.1.1 Expresiones regulares

Las expresiones regulares (ER) se utilizan para denotar conjuntos regulares; un conjunto regular es un conjunto de cadenas que se puede formar mediante las operaciones de unión, concatenación y cierre. [5]

ϕ es la expresión regular que representa al conjunto vacío $\{\}$, que es regular.

[3] $\{a\}$ es finito, por lo tanto regular, mientras que $\{a \cdot b\}*$ es la cerradura de $\{a \cdot b\}$, que es regular por ser finito.

Las expresiones regulares no representan en forma única a un lenguaje; pues pueden existir varias ER para un mismo lenguaje, lo cual no es conveniente, porque al conocer dos ER distintas no se puede estar seguro de que representan dos lenguajes distintos. Por ejemplo, las ER $(a+b)*$ y $(a*b*)*$ representan el mismo lenguaje.

Para comparar directamente dos ER en algunos casos resulta útil aplicar ecuaciones de equivalencia entre las ER, $ER_1 = ER_2$, cuyo significado es que el lenguaje de ER_1 es el mismo que el de ER_2, ya que contienen las mismas palabras.

2.1.2 Gramáticas regulares

Las gramáticas regulares generan los lenguajes regulares. Son las gramáticas más restrictivas.

Una gramática regular es un cuádruplo (Σ, V, R, S) donde:

- Σ es un alfabeto de constantes
- V es un alfabeto de variables
- R es el conjunto de reglas, es un subconjunto finito de $V \times (\Sigma V \cup \Sigma)$
- S es el símbolo inicial, es un elemento de V

Usualmente las reglas no se escriben como pares ordenados (a, aB), sino como $a \to a\beta$.

Aplicar una regla $a \Rightarrow aB$ de una gramática consiste en reemplazar a por aB en una palabra. Al proceso de aplicar una regla se le conoce como paso de derivación y se denota con $\Rightarrow$.

Una cadena $\alpha \in (V \cup \Sigma)*$ es derivable a partir de una gramática (V, Σ, R, S) si hay al menos una secuencia de pasos de derivación que la produce a partir del símbolo inicial S, es decir $S \Rightarrow ... \Rightarrow \alpha$.

El lenguaje $L(G)$ generado por una gramática (V, Σ, R, S) es el conjunto de palabras hechas exclusivamente de constantes, que son derivables a partir del símbolo inicial; $L = \{w \in \Sigma* \mid S \Rightarrow ... \Rightarrow w\}$.

El lado derecho de una regla debe contener un símbolo terminal y, como máximo, un símbolo no terminal. Estas gramáticas pueden ser:

- Lineales a derecha si todas las regla son de la forma:

$A \rightarrow aB$ ó $A \rightarrow a$: $A \in V \cup \{S\}$
$B \in V$
$a \in \Sigma$

En el lado derecho de las reglas el símbolo no terminal aparece a la derecha del símbolo terminal.

- Lineales a izquierda si todas las reglas son de la forma:

$A \rightarrow Ba$ ó $A \rightarrow a$: $A \in V \cup \{S\}$
$B \in V$
$a \in \Sigma$

En el lado derecho de las reglas el símbolo no terminal aparece a la izquierda del símbolo terminal.

En ambos casos, se puede incluir la regla $S \rightarrow \varepsilon$, si el lenguaje que se quiere generar contiene la cadena vacía.

2.2 AUTÓMATA DE ESTADOS FINITOS

2.2.1 Modelado de Sistemas Discretos

El modelado de fenómenos y procesos es una actividad que permite: [19]

- Verificar hipótesis sobre dichos procesos
- Efectuar predicciones sobre el comportamiento futuro
- Hacer simulaciones
- Hacer experimentos del tipo "¿qué pasaría si. . .?", sin tener que actuar sobre el proceso o fenómeno físico

Los eventos discretos son aquéllos en los que se considera su estado sólo en ciertos momentos, separados por intervalos de tiempo, sin importar lo que ocurre en el sistema entre estos momentos.

Usualmente se considera que la realidad es continua, y por lo tanto los sistemas discretos son solamente una abstracción de ciertos sistemas. [19]

Un estado es una situación en la que se permanece un determinado tiempo. En estos modelos se supone que se permanece en los estados un cierto tiempo y que los eventos son instantáneos.

Para elaborar modelos adecuados de un proceso real se requiere seguir los siguientes lineamientos:[17]

- Diferenciar entre los eventos que se consideran instantáneos y aquellos que tienen una duración considerable
- Las condiciones asociadas a los estados deben ser excluyentes
- Las condiciones asociadas a los estados de un modelo deben ser comprensivas, es decir que entre todas ellas cubren todos los casos posibles
- Los estados instantáneos son asociados a los eventos

Para los eventos con duración, es necesario identificar un evento de inicio y otro de terminación.

Estados finales

El propósito de algunos modelos de estados y eventos es el de reconocer secuencias de eventos buenas, de manera que se les pueda diferencias de las secuencias malas.

Los estados finales indican que la secuencia de eventos que llevó hasta ahí puede considerarse como aceptable.

2.2.2 Máquina de Estados Finitos

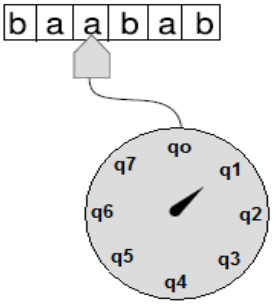


Figura 2. Componentes de una máquina abstracta.

Las máquinas de estados finitos pueden ser visualizadas como dispositivos con los siguientes componentes: Figura 2.

- Una cinta de entrada
- Una cabeza de lectura
- Un control

2.2.3 Funcionamiento de Autómatas Finitos

El funcionamiento de los autómatas finitos consiste en ir pasando de un estado a otro, a medida que va recibiendo caracteres de la palabra de entrada. Este

proceso puede ser seguido en los diagramas de estados; donde el estado inicial se indica mediante una flecha que no tiene nodo origen, los estados finales se representan con un círculo doble, los estados intermedios como círculos y las transiciones como arcos orientados que van de un estado a otro.

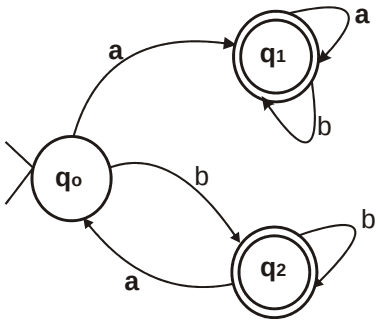


Figura 3. Funcionamiento Autómata Finito

El autómata de la figura 3, de entrada bb, inicia su operación en el estado q_0, que es el estado inicial, al recibir la primera b pasa al estado q_2. Luego, al recibir la segunda b de la palabra de entrada, pasará del estado q_2 a él mismo.

El camino recorrido en el diagrama de estados se pude visualizar como una trayectoria recorrida de estado en estado. Cada nodo del gráfico corresponde a un estado; los estados son el único medio del cual disponen los autómatas finitos para recordar los eventos que ocurren, por ejemplo, que caracteres se han leído hasta el momento; es decir, son máquinas de memoria limitada; para el autómata finito de la figura 3 la trayectoria seguida para la palabra ab consiste en la secuencia de estados: q_o, q_1, q_1.

2.2.4 Definición Formal

Un autómata finito es un modelo matemático de una máquina que acepta cadenas de un lenguaje definido sobre un alfabeto Σ. Consiste en un conjunto finito de estados y un conjunto de transiciones, δ, entre esos estados, que dependen de los símbolos de la cadena de entrada.

El autómata finito acepta una cadena c si la secuencia de transiciones correspondientes a los símbolos de c conduce desde el estado inicial a un estado final.

La función de transición indica a qué estado se va a pasar sabiendo cuál es el estado actual y el símbolo que se está leyendo. δ es una función y no simplemente una relación; esta característica, que permite saber cuál será el siguiente estado, se llama determinismo.

Si para todo estado del autómata existe como máximo una transición definida para cada símbolo del alfabeto, se dice que el autómata es determinístico (AFD). Si a partir de algún estado y para el mismo símbolo de entrada, se definen dos o más transiciones se dice que el autómata es no determinístico (AFND).

Formalmente un autómata finito se define como un quíntuplo $(K, \Sigma, \delta, s, F)$ donde:

- K : conjunto finito de estados
- Σ : alfabeto o conjunto finito de símbolos de entrada
- δ : función de transición de estados, que se define como
  $\delta: K \times \Sigma \to K$ si el autómata es determinístico
  $\delta: K \times \Sigma \to P(K)$ si el autómata es no determinístico; $P(K)$ es el conjunto potencia de K, es decir el conjunto de todos los subconjuntos de K
- s : estado inicial; $s \in K$
- F : conjunto de estados finales o estados de aceptación; $F \in K$

La parte fundamental de un autómata finito es su función de transición y puede ser expresada mediante una tabla; la diferencia entre dicha notación formal y los diagramas de estado es simplemente de notación, pues la información es la misma.

Tanto en los diagramas de estado como en la representación formal hay que respetar las condiciones para tener un autómata válido; en particular, el número de transiciones que salen de cada estado debe ser igual a la cantidad de caracteres del alfabeto, pues δ es una función que está definida para todas las entradas posibles. [3]

Se debe tener en cuenta que debe haber exactamente un estado inicial. En cambio, la cantidad de estados finales puede ser cualquiera, incluso cero, hasta un máximo de $|K|$, cantidad de estados.

En la notación formal hay que seguir las transiciones, que ahora no son representadas como flechas, sino como elementos del conjunto de transiciones.

2.2.5 Palabras aceptadas

Los autómatas finitos pueden ser utilizados para reconocer ciertas palabras y diferenciarlas de otras.

Un autómata finito determinista reconoce o acepta una palabra si se cumplen las siguientes condiciones:

- Se consumen todos los caracteres de dicha palabra de entrada, siguiendo las transiciones y pasando en consecuencia de un estado a otro

- Al terminar la palabra, el estado al que llega es uno de los estados finales del autómata

Así, en el ejemplo de la figura 3, el autómata acepta la palabra bb, pues al terminar de consumirla se encuentra en el estado q_2, el cual es final.

El lenguaje aceptado por una máquina M es el conjunto de palabras aceptadas por dicha máquina.

El autómata de la figura 3 acepta las palabras que empiezan con a, así como las palabras que contienen aa, y también las que terminan en b, como por ejemplo $abab$, $aaaaa$, $baaa$, etc. En cambio, no acepta $baba$ ni bba, $babba$, etc. No acepta la palabra vacía ε. Para que un AFD acepte ε se necesita que el estado inicial sea también final.

2.2.6 Formalización del funcionamiento de los AFD

Las informaciones relevantes para resumir la situación de la máquina en un instante son:

- El contenido de la cinta
- La posición de la cabeza lectora
- El estado en que se encuentra el control

Una configuración es un elemento de $\Sigma^* \times N \times K$, donde el primer elemento es el contenido de la cinta, N describe la posición de la cabeza y K es el estado.

Sólo es necesario incluir en las configuraciones aquellas informaciones que tengan relevancia en cuanto a la aceptación de la palabra al final de su análisis. Como la cabeza lectora no puede desplazarse hacia la izquierda, los caracteres por los que ya pasó no afectarán más el funcionamiento de la máquina. Por lo tanto, es suficiente con considerar lo que falta por leer de la palabra de entrada, en vez de la palabra completa. Esta solución tiene la ventaja de que entonces no es necesario representar la posición de la cabeza, pues ésta se encuentra siempre al inicio de lo que falta por leer.

Una configuración es un elemento de $k \times \Sigma^*$. Por ejemplo, la configuración correspondiente a la figura 3 es $[[q_1, abab]]$.

La relación entre configuraciones, $C \vdash_M C_2$, significa que de la configuración C_1 la máquina M puede pasar en un paso a la configuración C_2 [1]. Definiendo formalmente esta noción se tiene:

$[[q_1, \sigma w]] \vdash_M [[q_2, w]]$ para un $\sigma \in \Sigma$, si y sólo si existe una transición en M tal que $\delta(q_1, \sigma) = q_2$; σ es el caracter que se leyó.

La cerradura reflexiva y transitiva de la relación $\vdash_M$ es denotada por $\vdash^*_M$. Así, la expresión $C \vdash^*_M C_2$ indica que de la configuración C_1 se puede pasar a C_2 en algún número de pasos, que puede ser cero, si $C_1 = C_2$.

Una palabra $w \in \Sigma^*$ es aceptada por una máquina $M = (K, \Sigma, \delta, s, F)$ sólo si existe un estado $q \in F$ tal que $[[s, W]] \vdash^*_M [[q, \varepsilon]]$. No basta con que se llegue a un estado final q, sino que además ya no deben quedar caracteres por leer.

Para probar que el AFD de la figura 3 acepta la palabra $babb$, hay que encontrar una serie de configuraciones tales que se pueda pasar de una a otra por medio de la relación $\vdash_M$. La única forma posible es la siguiente: [4]

$$[[q_0, babb]] \vdash_M [[q_2, abb]] \vdash_M [[q_0, bb]] \vdash_M [[q_2, b]] \vdash_M [[q_2, \varepsilon]]$$

Como $q_2 \in F$, la palabra es aceptada.

Un cálculo en una máquina M es una secuencia de configuraciones $C_1, C_2, ...C_n$, tales que $C \vdash C_{i+1}$.

2.2.7 Diseño de un Autómata Finito determinista

El problema de construir un AFD que acepte exactamente un lenguaje dado, es básicamente un problema de diseño. Para afrontar este problema no es conveniente proceder por ensayo y error, pues hay que considerar muchas posibilidades. Hay dos maneras de equivocarse al diseñar un AFD:

1. Que sobren palabras: el autómata acepta algunas palabras que no debería aceptar.

2. Que falten palabras: hay palabras en el lenguaje considerado que no son aceptadas por el AFD, cuando deberían serlo.

Por ejemplo el autómata de la figura 3 es una solución defectuosa para el lenguaje de las palabras en el alfabeto $\{a, b\}$ que no tienen varias $a's$ seguidas porque:

[4] En los AFD's, para cada palabra de entrada sólo hay una secuencia posible de configuraciones, precisamente porque son deterministas.

1 Hay palabras, como baa, que tiene $a's$ seguidas y sin embargo son aceptadas por el AFD.

2 Hay palabras, como ba, que no tienen $a's$ seguidas y sin embargo no son aceptadas por el AFD.

Como es posible equivocarse de las dos maneras a la vez en un sólo autómata es necesario diseñar los AFD de una manera más sistemática.

El elemento más importante en el diseño sistemático de autómatas a partir de un lenguaje consiste en determinar, de manera explícita, que condición recuerda cada uno de los estados del AFD; la única forma de memoria que tienen los AFD es el estado en que se encuentran. Por lo cual el diseño del AFD inicia con un conjunto de estados que recuerdan condiciones importantes o transiciones que permiten pasar de un estado a otro, lo cual resulta relativamente sencillo cuando se cuenta con los estados y sus condiciones asociadas. [18]

Diseño por conjuntos de estados

En este tipo de diseño se asocian condiciones a grupos de estados y no a estados individuales. De esta manera se aumenta el grado de abstracción en la etapa inicial de diseño, con lo cual es posible afrontar problemas más complejos.

Este método consiste en identificar inicialmente condiciones asociadas al enunciado del problema, aunque éstas no sean suficientemente específicas para asociarse a estados individuales. Este método se describe mediante un ejemplo:

Diseñar un AFD que acepte las palabras del lenguaje en $\{0,1\}$ donde las palabras no contienen la sub-cadena 11 pero sí 00.

A partir del enunciado se identifican las siguientes situaciones:

- Las letras consumidas hasta el momento no contienen ni 00 ni 11
- Contienen 00 pero no 11
- Contienen 11

Estas condiciones cumplen dos requisitos que siempre se deben cumplir en este tipo de diseños:

- Las condiciones deben ser excluyentes, es decir que no deben poder ser ciertas dos o más al mismo tiempo
- Las condiciones deben ser comprensivas, es decir que no falten casos por considerar

Para lograr AFD se sebe clarificar cada grupo de estados, considerando lo que ocurre al recibir cada uno de los posibles caracteres de entrada. La forma en que se subdivide cada grupo de estados en estados individuales es:

- Las letras consumidas hasta el momento no contienen ni 00 ni 11

  1. Inicialmente no se han recibido caracteres
  2. Se acaba de recibir un 0
  3. Se acaba de recibir un 1

- Contienen 00 pero no 11

  1. Se acaba de recibir un 0
  2. Se acaba de recibir un 1

- Contienen 11, no hay sub-condiciones

Esto da un total de 6 estados, cada uno de los cuales tiene una condición específica asociada.

Luego se debe hacer el diseño detallado de las transiciones; el resultado se muestra en la figura 4.

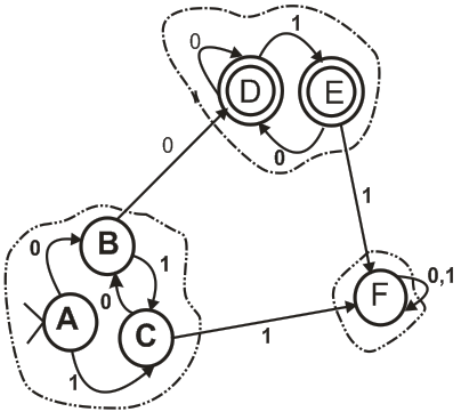


Figura 4. Detalle de estados.

Encontrar directamente las condiciones asociadas a los estados puede ser algo difícil; por ejemplo, encontrar directamente la condición: las letras consumidas hasta el momento no contienen 00 ni 11 y se ha recibido un 0, estado “B” en la figura 4.

En cualquier caso, ya sea que se encuentren directamente las condiciones para cada estado, o primero para grupos de estados, se deben determinar los estados con sus condiciones asociadas, y después se trazan las transiciones.

Diseño por AFD por complemento

En ocasiones, para un cierto lenguaje L, es más sencillo encontrar un AFD para el lenguaje exactamente contrario, complementario $L^c = \Sigma^* - L$. En estos casos, una solución sencilla es hallar primero un AFD para L^c y luego hacer una transformación sencilla para obtener el autómata que acepta L.

Si $M = (K, \Sigma, \delta, s, F)$ es un autómata determinista que acepta un lenguaje regular L, para construir un autómata M^c que acepte el lenguaje complemento de L, esto es, $\Sigma^* - L$, sólo es necesario intercambiar los estados finales de M en no finales y viceversa. Formalmente $M^c = (K, \Sigma, \delta, s, K - F)$. Así, cuando una palabra es rechazada en M, ella es aceptada en M^c y viceversa.

2.2.8 Simplificación de autómatas finitos

En el caso de los AFD, la simplificación es la reducción en el número de estados, pero aceptando el mismo lenguaje que antes de la simplificación, es decir la minimización a la obtención de un autómata con el menor número posible de estados. [8]

Dos estados son equivalentes, $q_1 \approx q_2$, sólo si al intercambiar uno por otro en cualquier configuración no altera la aceptación o rechazo de toda palabra.

Formalmente se dice que dos estados p y q son equivalentes si cuando $[[s, uv]] \vdash^*_M [[q, v]] \vdash^*_M [[r, \varepsilon]]$ y $[[p, v]] \vdash^*_M [[t, \varepsilon]]$ entonces r y t son estados compatibles.

Es decir que si $p \approx q$, al cambiar q por p en la configuración, la palabra va a ser aceptada, se acaba en el estado final t, si y sólo si de todos modos iba a ser aceptada sin cambiar p por q, se acaba en el estado final r.

El único problema con esta definición es que, para verificar si dos estados dados p y q son equivalentes, habría que examinar, para cada palabra posible de entrada, si al intercambiarlos en las configuraciones altera o no la aceptación de esa palabra. Esto es evidentemente imposible para un lenguaje infinito. La definición dice qué son los estados equivalentes, pero no cómo saber si dos estados son equivalentes. Este aspecto es resuelto por la siguiente indicación:

Dado un AFD $M = (K, \Sigma, \delta, q, F)$ y dos estados $q_1, q_2 \in K$, se tiene que $q_1 \approx q_2$ si y sólo si $(K, \Sigma, \delta, q_1, F) \approx (K, \Sigma, \delta, q_2, F)$.

Es decir, para saber si dos estados q_1 y q_2 son equivalentes, se les ubica como estado inicial de sus respectivos autómatas M_1 y M_2, y se procede a comparar dichos autómatas.

Si éstos últimos son equivalentes, quiere decir que los estados q_1 y q_2 son equivalentes.

2.2.9 Autómatas finitos con salida

Los autómatas finitos no sólo realizan la tarea de aceptar o rechazar una palabra, determinando así si pertenece o no a un lenguaje. También es posible definirlos de manera tal que produzca una salida diferente de sí o no. Por ejemplo, en el contexto de una maquina controlada por un autómata, puede haber distintas señales de salida que correspondan a los comandos enviados a la máquina para dirigir su acción. Hay dos formas de definir a los autómatas con salida, según si la salida depende de las transiciones y estados o bien solo del estado en que se encuentra el autómata. En el primer caso, se trata de los autómatas de Mealy, y en el segundo, de los autómatas de Moore, propuestos respectivamente por G. Mealy [13] y E. Moore [15].

Máquina de Moore

En las máquinas de Moore la salida depende únicamente del estado en que se encuentra el autómata. Ésta salida es producida una vez, y cuando se llega a otro estado, o al mismo por efecto de una transición, se produce el símbolo de salida asociado al estado al que se llega.

Las máquinas de Moore se representan gráficamente como cualquier AFD, al que se añade, al lado de cada estado, la salida asociada, que es una cadena de caracteres.

Para formalizar los autómatas de Moore una idea sencilla es añadir a un AFD estándar una función que asocie a cada estado una palabra de salida: λ; y un alfabeto de salida Γ, que puede ser distinto al de entrada. Todos los demás aspectos permanecen igual que en un AFD.

Una máquina de Moore es un séxtuplo $(K, \Sigma, \Gamma, \delta, \lambda, q_0)$, en donde K, Σ y δ son como en los AFD, y q_0 es el estado inicial; además se tiene Γ que es el alfabeto de salida, y λ, que es una función que obtiene la salida asociada a cada estado; la salida es una cadena de caracteres tomados de Γ.

Máquina de Mealy

En las máquinas de Mealy la salida producida depende de la transición que se ejecuta y del estado presente. Por esto, en la notación gráfica los nombres de las

flechas son de la forma σ / ω, donde σ es el caracter que se consume de entrada, y ω es la palabra que se produce en la salida.

Para formalizar las máquinas de Mealy, se aumentan a las transiciones la palabra producida en la salida. Se prefiere definir una función de salida λ, pero a diferencia de las máquinas de Moore, ahora toma como entrada un estado y un caracter de entrada. Es lo mismo que la salida dependa del estado y un caracter, a que dependa de una transición. [15]

Una máquina de Mealy es un séxtuplo $(K, \Sigma, \Gamma, \delta, \lambda, q_0)$, en el que todos los componentes tienen el mismo significado mencionados anteriormente, a excepción de λ, que es una función $\lambda : K \times \Sigma \rightarrow \Gamma*$, toma un elemento de $K \times \Sigma$, que incluye un estado y un caracter de entrada, y produce una palabra formada por caracteres de Γ.

A diferencia de las máquinas de Moore, en las máquinas de Mealy la salida depende de la entrada, además de los estados. Es como asociar la salida a las transiciones, más que a los estados.

Los criterios para diseñar tanto máquinas de Moore como de Mealy son básicamente los mismos que para cualquier otro AFD.

Equivalencias de las máquinas de Moore y Mealy

Aunque muchas veces, para un mismo problema, la máquina de Mealy es más simple que la correspondiente de Moore, ambas clases de máquinas son equivalentes. Si se desprecia la salida de las máquinas de Moore antes de recibir el primer caracter, o sea, con entrada ε, es posible encontrar, para una máquina de Moore dada, su equivalente de Mealy, en el sentido de que producen la misma salida, y viceversa.

La transformación de una máquina de Moore en máquina de Mealy es trivial, pues se hace $\lambda_{Mealy}(q, a) = \lambda_{Moore}(\delta_{Moore}(q, a))$, es decir, se obtiene qué salida producirá una transición de Mealy viendo la salida del estado al que lleva dicha transición en Moore.

La transformación de una máquina de Mealy en Moore es más complicada, pues en general hay que crear estados adicionales. [1]

2.2.10 Autómatas finitos no deterministas (AFN)

Una extensión a los autómatas finitos deterministas es la de permitir que de cada nodo del diagrama de estados salga un número de flechas mayor o menor que $|\Sigma|$. Así, se puede permitir que falte la flecha correspondiente a alguno de los símbolos del alfabeto, o bien que haya varias flechas que salgan de un sólo nodo con el

mismo nombre [8]. Inclusive se permite que las transiciones tengan como nombre palabras de varias letras o hasta la palabra vacía. A estos autómatas finitos se les llama no determinísticos o no deterministas, AFN.

Los AFN tienen menos restricciones que los AFD, resulta que los AFD son un caso particular de los AFN, por lo que todo AFD es de hecho un AFN. Sin embargo, en los autómatas no determinísticos se presenta una dificultad para poder saber qué camino tomar a partir de un estado dado cuando se presenta un símbolo, pues puede haber varias opciones.

2.2.11 Representación formal de los AFN

Un autómata finito no determinista es un quíntuplo $(K, \Sigma, \Delta, s, F)$ donde K, Σ, s y F tienen el mismo significado que para el caso de los autómatas determinísticos, y Δ, es la relación de transición, un subconjunto finito de $K \times \Sigma^* \times K$.

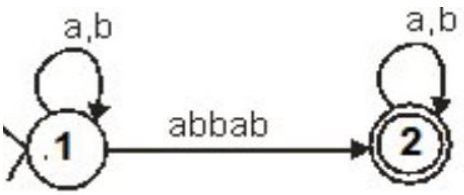


Figura 5 AFN para palabras que contienen $abbab$

El AFN de la figura 5 es representado matemáticamente por el quíntuplo:

$$(\{1,2\},\{a,b\},\{(1,a,1),(1,b,1),(1,abbab,2),(2,a,2),(2,b,2)\},1,\{2\})$$

Δ es una relación, no una función, el segundo elemento, ω, de la relación de transición es una palabra, no un caracter del alfabeto. Esto significa que cada tripleta $(q_1, \omega, q_2) \in \Delta$, que es una transición representada como una flecha que permite pasar de q_1 a $, q_2$ leyendo en la entrada una sub-cadena ω que puede ser palabra vacía.

Una palabra ω es aceptada por un autómata no determinístico si y sólo si existe una trayectoria en su diagrama de estados, que parte del estado inicial y llega a un estado final, tal que la concatenación de los nombres de las flechas es igual a ω.

2.2.12 Equivalencia de AFD y AFN

Los autómatas finitos determinísticos son un subconjunto propio de los no determinísticos, lo que quiere decir que todo AFD es un AFN.[5] Podría entonces

[5] Menos el hecho de que δ es una función y Δ una relación.

pensarse que los AFN son más poderosos que los AFD, en el sentido de que habría algunos lenguajes aceptados por algún AFN para los cuales no habría ningún AFD que los acepte. Sin embargo, en realidad no sucede así. [8]

Para todo AFN N, existe algún AFD D tal que $L(N) = L(D)$.

El método que se usa para pasar de un AFN a un AFD se basa en considerar el conjunto de estados en los que podría encontrarse el AFN al haber consumido una cierta entrada.

Método de los conjuntos de estados

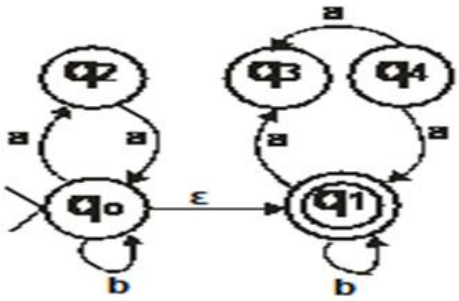


Figura 6. AFN a transformar en AFD

Dado un AFN M, considerando la idea de mantener un conjunto de estados Q_i en los que sería posible estar en cada momento al ir consumiendo las letras de una palabra de entrada.

Para analizar qué sucede cuando el AFN de la figura 6 recibe la palabra $baaaab$ se lleva registro de los conjuntos de estados en los que podría encontrarse el AFN. Inicialmente, podría encontrarse en el estado inicial q_0, pero sin leer ningún caracter podría estar también en el estado q_1, o sea que el proceso arranca con el conjunto de estados $Q_0 = \{q_0, q_1\}$. Al consumirse el primer caracter, b, se puede pasar de q_0 a q_0 o bien a q_1, pasando por el ε, mientras que del q_1 sólo se puede pasar a q_1. Entonces, el conjunto de estados en que se puede estar al consumir la b es $Q_1 = \{q_0, q_1\}$. Y así en adelante:

Entrada	Estados
	$\{q_0, q_1\}$
b	$\{q_0, q_1\}$
a	$\{q_2, q_4\}$
a	$\{q_0, q_1, q_3\}$

a	$\{q_1, q_2, q_4\}$
a	$\{q_0, q_1, q_3, q_4\}$
a	$\{q_1, q_2, q_3, q_4\}$
b	$\{q_1\}$

Tabla 1. Conjunto de estados por los que pasa el AFN de la figura 6.

Como el último conjunto de estados $\{q_1\}$ incluye a un estado final, se concluye que la palabra de entrada puede ser aceptada.

Si se considera a los conjuntos de estados Q_i como una especie de mega-estados de cierto autómata, entonces en realidad se han seguidos los pasos de ejecución de un AFD con mega-estados.

Si en vez de considerar una palabra en particular, como fue $baaaaab$, se considera cada posible caracter que puede llegar al estar en un mega-estado, entonces se podrá completar un AFD, que deberá ser equivalente al AFN dado.

2.3 EQUIVALENCIA DE EXPRESIONES REGULARES Y AF

Teorema de Kleene: Un lenguaje es regular si y sólo si es aceptado por algún autómata finito. [20] [8]

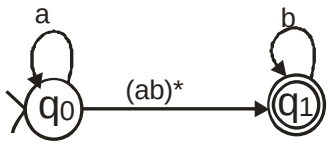


Figura 7. Grafica de transición.

La prueba de que si un lenguaje es regular entonces es aceptado por un AF, consiste en dar un procedimiento para transformar en forma sistemática una expresión regular en un autómata finito que acepte su lenguaje.

El objetivo es hacer una transformación gradual que vaya convirtiendo la ER en AF, para lo cual se requiere utilizar alguna representación de los lenguajes regulares que sea intermedia entre las ER y los AFN.

El uso de las gráficas de transición es una solución. Estas son esencialmente AFN en que los nombres de las flechas tienen expresiones regulares, en lugar de palabras. Las gráficas de transición (GT) son por lo tanto quíntuplos $(K, \Sigma, \Delta, s, F)$ donde $\Delta \in K \times ER \times K$.

En la figura 7 se muestra una GT; en este ejemplo es fácil ver que debe aceptar palabras que tienen primero una sucesión de $a's$, luego repeticiones de ab, y finalmente repeticiones de $b's$. Esta GT se representaría formalmente como el quíntuplo:

$$(\{q_0, q_1\}, \{a, b\}, \{(q_0, a, q_0), (q_0, (ab), \_, q_1), (q_1, b, q_1)\}, q_0, \{q_1\})$$

Los AFN son un subconjunto propio de las GT, puesto que las palabras en los nombres de un AFN pueden ser vistas como expresiones regulares que se representan a sí mismas.

A continuación se describe el procedimiento de transformación de ER a AFN.

A partir de una ER es ligero obtener una GT que acepte el mismo lenguaje. En efecto, sea R una ER; entonces, si

$$G_1 = (\{q_0, q_1\}, \Sigma, \{(q_0, R, q_1)\}, q_0, \{q_1\})$$

entonces $L(G) = L(R)$.

Lo que falta por hacer es transformar gradualmente G_1 en G_2 luego en G_3, etc., hasta llegar a un G_n tal que en las flechas no haya más que caracteres solos, o la palabra vacía. En efecto, $G_n \in AFN$. Este es un proceso de eliminación gradual de los operadores de las ER.

Para eliminar los operadores de las ER en G_i, se aplican reemplazos de ciertas transiciones por otras, hasta que no sea posible aplicar ninguno de estos reemplazos.

Las transformaciones elementales se ilustran en la Tabla 2.

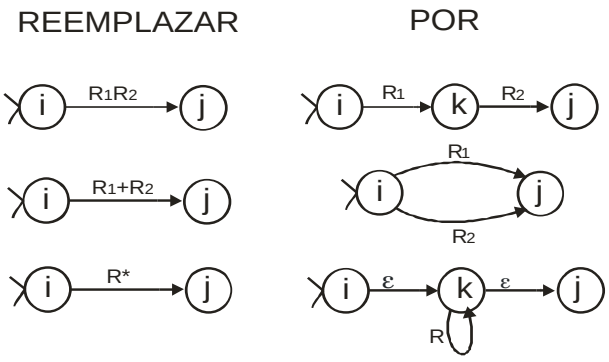


Tabla 2. Eliminación de operaciones para pasar de ER a AF

2.4 EQUIVALENCIA DE GRAMÁTICAS REGULARES Y AF

La clase de los lenguajes generados por alguna gramática regular es exactamente la de los lenguajes regulares y por ende la de los autómatas finitos.[8] La prueba de este teorema consiste en proponer un procedimiento para, a partir de una gramática dada, construir un autómata finito, y viceversa.

Dicho procedimiento es directo, y consiste en asociar a los símbolos no terminales de la gramática, los estados de un autómata. Así, para cada regla $A \to bC$ en la gramática tenemos una transición (A,b,C) en el autómata. Sin embargo, queda pendiente el caso de las reglas $A \to b$. Para estos casos, se tienen transiciones (A,b,Z), donde Z es un nuevo estado para el que no hay un no terminal asociado; Z es el único estado final del autómata.

Similarmente se obtiene a partir de un AFD (K,Σ,δ,s,F), la gramática regular correspondiente. Para cada transición de la forma $((p,\sigma),q) \in \delta$, habrá en la gramática una regla $X_p \to \sigma X_q$, donde X_i es la variable en la gramática que corresponde al estado i del AFD. Queda pendiente cómo obtener las reglas de la forma $X_p \to \sigma$, que son las que permiten terminar una derivación. La aplicación de este tipo de reglas debe corresponder al consumo del último caracter de una palabra aceptada en el AFD. Al terminar una palabra aceptada en un AFD, necesariamente se debe estar en un estado final. De ahí se concluye que hay que incorporar a la gramática, por cada transición $((p,\sigma),q)$, donde $q \in F$, una regla adicional $X_p \to \sigma$, además de la regla $X_p \to \sigma X_q$.

2.5 LIMITACIONES DE LOS LENGUAJES REGULARES

Los AF están limitados a los estados de que disponen como único medio para recordar la serie de símbolos recibidos hasta un momento dado. Se debe considerar que, en un AF, el único plano de los símbolos recibidos es el estado en que se encuentra. Por lo mismo, varias secuencias distintas de caracteres que llevan a un mismo estado son consideradas como indistinguibles. Esta limitación de los AF los hace finalmente incapaces de distinguir las palabras aceptables de las no aceptables en ciertos lenguajes, más complicados que los lenguajes regulares. [4]

Por ejemplo, para el lenguaje $\{a^n b^n\}$ no es posible construir un autómata finito que lo acepte, ni representarlo por una expresión regular o gramática regular. En efecto, se supone que un AFD está recorriendo una palabra $\{a^n b^n\}$, entonces al terminar el grupo de $a's$ el autómata debe recordar cuántas encontró, para poder comparar con el número de $b's$. Como la cantidad de $a's$ que puede haber en la primera mitad de la palabra no es constante, dicha cantidad no puede recordarse con una cantidad de memoria fija, como es la de los autómatas finitos.

3. LENGUAJES LIBRES DE CONTEXTO Y AUTÓMATA DE PILA

Puesto que los autómatas finitos no son suficientemente poderosos para aceptar los lenguajes libres de contexto (LLC), se puede agregar algo a los AF de manera que se incremente su poder de cálculo.

En esta sección se verán los autómatas de pila y su relación con los lenguajes libres de contexto. Los autómatas de pila tienen memoria representada en forma de una pila lo que les permite reconocer una clase de lenguajes más grande: los lenguajes libres de contexto.

Un autómata de pila es una máquina con un número finito de estados, una cinta de entrada y una pila. El autómata lee cadenas de símbolos de longitud finita de la cinta de entrada que es infinita. El comportamiento de la máquina está determinado por el estado en que se encuentra, el símbolo en el tope de la pila y el símbolo en la cinta de entrada. Para cada estado y dependiendo del tope de la pila, al leer los símbolos de la cinta de entrada cambia de estado, compila, guarda, o descompila, borra, de la pila y avanza en la cinta de entrada.

Estas máquinas se utilizan para reconocer lenguajes; es decir, para leer cadenas y determinar si pertenecen o no a un lenguaje dado. También sirven para describir el comportamiento de ciertas máquinas.

3.1 LENGUAJES LIBRE DE CONTEXTO (LLC)

Los Lenguajes Libres de Contexto (LLC) forman una clase de lenguajes más amplia que los Lenguajes Regulares, de acuerdo con la Jerarquía de Chomsky. Estos lenguajes son importantes tanto desde el punto de vista teórico, por relacionar las llamadas Gramáticas Libres de Contexto (GLC) con los Autómatas de Pila, como desde el punto de vista práctico, ya que casi todos los lenguajes de programación están basados en los LLC.

El formato de las reglas en las GLC es menos rígido que en las gramáticas regulares, y así toda gramática regular es GLC pero no viceversa.

3.1.1 Formalización de las gramáticas libres de contexto

Una gramática libre de contexto es un cuádruplo (V, Σ, R, S) donde:

- V es un alfabeto de variables, también llamadas no terminales
- Σ es un alfabeto de constantes, también llamadas terminales Suponemos que V y Σ son disjuntos, esto es, $V \cap \Sigma = \varepsilon$

- R , el conjunto de reglas, es un subconjunto finito de $V \times (V \cup \Sigma)^*$
- S , el símbolo inicial, es un elemento de V

Una GLC G es correcta con respecto al lenguaje dado L cuando el lenguaje generado por G no contiene palabras que estén fuera de L es decir, $L(G) \subseteq L$, donde $L(G)$ denota el lenguaje generado por G. Análogamente, se dice que G es completa cuando G es capaz de generar al menos las palabras de L, es decir, $L \subseteq L(G)$**.** Al diseñar gramáticas, es posible cometer dos clases de errores:

1. Que sobren palabras, es decir, que la gramática genere algunas palabras que no debería generar. En este caso, la gramática sería incorrecta.

2. Que falten palabras, es decir, que haya palabras en el lenguaje considerado para las que no hay ninguna derivación. En este caso, la gramática sería incompleta.

Las pruebas que permiten establecer la correspondencia entre un lenguaje y una gramática dados requieren dos partes:[8]

1. Prueba de corrección, que garantiza que todas las palabras que se producen al utilizar la gramática efectivamente corresponden a la descripción del lenguaje dado.

2. Prueba de completez, que se asegura de que al producir palabras con la gramática, no falten palabras del lenguaje dado.

Muchas veces es posible hacer modificaciones sencillas a una gramática conocida para obtener la del lenguaje requerido.

3.1.2 GLC para unión de lenguajes

Muchos lenguajes pueden ser expresados en forma útil como la unión de otros dos lenguajes, para los cuales se conocen las gramáticas que los generan. El lenguaje $\{a^n b^m \mid n \neq m\}$ se puede expresar como la unión de los lenguajes: [8]

$$\{a^n b^m \mid n \neq m\} = \{a^n b^m \mid n < m\} \{a^n b^m \mid n > m\}$$

La manera de combinar dos gramáticas con símbolos iníciales S_1 y S_2 respectivamente, para producir la unión de los lenguajes originales, consiste en crear un nuevo símbolo inicial S (S_1 y S_2 dejan de ser iníciales), tomar las reglas tanto de una gramática como de otra, y añadir dos nuevas reglas, $S \to S_1$ y $S \to S_2$, para que el nuevo símbolo inicial sea capaz de generar cualquiera de los dos antiguos símbolos iníciales; a partir del primer paso, se continua la derivación utilizando alguna de las dos gramáticas originales, sin utilizar las reglas de la otra.

Para garantizar esto último se supone que las dos gramáticas originales no tienen ninguna variable en común.

Formalmente la gramática que genera la unión de lenguajes es: Sean $G_1 = (V_1, \Sigma_1, R_1, S_1)$ y $G_2 = (V_2, \Sigma_2, R_2, S_2)$ dos GLC; se puede suponer, sin pérdida de generalidad, que las variables de G_1 y G_2 son disjuntas. La GLC que genera $L(G_1) \cup L(G_2)$ es:

$$G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, R_1 \cup R_2 \cup \{S \to S_1, S \to S_2\}, S).$$

En efecto, para una palabra $\omega \in L(G_1)$ la derivación comienza aplicando $S \to S_1$, y después se continua con la derivación a partir de S_1. Similarmente se hace para una palabra $\omega \in L(G_2)$.

Para el lenguaje $\{a^n b^m \mid n \neq m\} = \{a^n b^m \mid n < m\}\{a^n b^m \mid n > m\}$, las gramáticas originales tendrían reglas:

$\{a^n b^m \mid n > m\}$	$\{a^n b^m \mid n < m\}$
1. $S_1 \to aS_1 b$	4. $S_2 \to aS_2 b$
2. $S_1 \to aS_1$	5. $S_2 \to S_2 b$
3. $S_1 \to a$	6. $S_2 \to b$

Tabla 3. Gramáticas del lenguaje $\{a^n b^m \mid n \neq m\} = \{a^n b^m \mid n < m\}\{a^n b^m \mid n > m\}$

La mezcla de gramáticas se da cuando es necesario combinar dos gramáticas, pero permitiendo que las gramáticas a combinar tengan un mismo símbolo inicial. La GLC mezclada contendría simplemente la unión de todas las reglas de las dos gramáticas.

3.1.3 Árbol de derivación

Las GLC tienen la propiedad de que las derivaciones pueden ser representadas en forma de árbol, conocido como árbol de derivación.

Sea $G = (V, \Sigma, R, S)$ una GLC. Entonces un árbol de derivación cumple las siguientes propiedades [2]:

- Cada nodo tiene un nombre, estados
- La raíz tiene un nombre S, estado inicial
- Los nombres de los nodos que no son hojas debe estar en V, no terminales, y las de las hojas en $\Sigma \cup \{\varepsilon\}$, terminales

- Si un nodo n tiene nombre A, y los nodos $n_{1...}n_m$ son sus hijos (de izquierda a derecha), con nombres respectivamente $A_{1...}A_m$, entonces $A \to A_{1...}A_m \in R$.

Al realizar un recorrido en orden del árbol de derivación se recupera la cadena a partir de la cual se construyó dicho árbol. Por lo cual compilar una cadena de caracteres se logra construyendo el árbol de derivación, a partir del producto de éste; el producto es la cadena de caracteres que resulta de concatenar los caracteres terminales encontrados en los nombres de los nodos hoja, haciendo el recorrido en orden del árbol de derivación.

Se dice que una gramática es ambigua si y sólo si alguna palabra del lenguaje que genera tiene más de un árbol de derivación. La ambigüedad es una propiedad de la gramática, no de su lenguaje generado. Para un mismo lenguaje puede haber una gramática ambigua y una no ambigua.

Existen técnicas para eliminar la ambigüedad de una GLC; en general estas técnicas consisten en introducir nuevos no-terminales de modo que se eliminen los árboles de derivación no deseados.

Dichas técnicas de eliminación de ambigüedad no son siempre aplicables, y de hecho hay algunos LLC para los que es imposible encontrar una gramática libre de contexto no ambigua; estos lenguajes se llaman inherentemente ambiguos. Un ejemplo, dado en [8] junto con la prueba correspondiente, es el siguiente:

$$L = \{a^n b^n c^m d^m\}\{a^n b^m c^m d^m\}, r \geq 1, m \geq 1$$

3.1.4 Derivaciones izquierda y derecha

En una gramática no ambigua G, a una palabra $\omega \in L(G)$ corresponde un sólo árbol de derivación A_G; aun así, puede haber varias derivaciones para obtener ω a partir del símbolo inicial, $S \Rightarrow ... \Rightarrow \omega$. Una manera de hacer única la forma de derivar una palabra consiste en restringir la elección del símbolo que se va a expandir en curso de la derivación.

Si se tiene en cierto momento de la derivación la palabra $(S())(S)$, en el paso siguiente se puede aplicar alguna regla de la gramática ya sea a la primera o a la segunda de las S. En cambio, si se restringe a aplicar las reglas sólo al no terminal que se encuentre más a la izquierda en la palabra, entonces habrá una sola opción posible.

Elegir el no terminal más a la izquierda es arbitrario; igual se puede elegir el no terminal más a la derecha.

Se llama derivación izquierda de una palabra ω a una secuencia $S \Rightarrow \omega_1 \Rightarrow ... \Rightarrow \omega_n \Rightarrow \omega$ donde, para pasar de ω_i a ω_{i+1}, se aplica una regla al no terminal de ω_i que se encuentre más a la izquierda. Similarmente se puede definir una derivación derecha.

Para la gramática no ambigua con reglas $S \to AB$, $A \to a$, $B \to b$, la palabra ab se produce con la derivación izquierda:

$$S \Rightarrow AB \Rightarrow aB \Rightarrow ab$$

También se puede producir con la derivación derecha:

$$S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$$

Teorema: Para una gramática no ambigua G, y una palabra $\omega \in L(G)$, existe solamente una derivación izquierda $S \Rightarrow^* \omega$. [8]

Prueba: La derivación izquierda corresponde a un recorrido en pre orden del árbol de derivación, expandiendo los no terminales que se van encontrando en el camino. Se sabe que existe un sólo recorrido en pre orden para un árbol dado.

3.1.5 Gramáticas Libres y sensitivas al Contexto

Las GLC deben su nombre a una comparación con otro tipo de gramáticas, las llamadas sensitivas al contexto, donde para una regla $\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2$, el símbolo A sólo puede generar β cuando se encuentra rodeado por el contexto $\alpha_1 ... \alpha_2$. En cambio, en las GLC no es necesario especificar un contexto, por esto se llaman libres de contexto.

Las gramáticas sensitivas al contexto son estrictamente más poderosas que las GLC; un ejemplo es el lenguaje de las cadenas de la forma $a^n b^n c^n$, para el que no hay ninguna GLC.

3.1.6 Transformación de las GLC y Formas Normales

En muchas situaciones se considera conveniente modificar las reglas de la gramática, de manera que cumplan las reglas con propiedades tales como no producir la cadena vacía del lado derecho. Cuando se habla de modificar las reglas de la gramática, se entiende que esto debe hacerse sin modificar el lenguaje generado.

Por ejemplo, la presencia de reglas que producen vacío en la gramática puede ser fuente de dificultades tales como la ambigüedad, o la posibilidad de tener derivaciones arbitrariamente largas.

Tomando por ejemplo la siguiente gramática para los paréntesis bien balanceados:

1. $S \to SS$
2. $S \to (S)$
3. $S \to \varepsilon$

Con esta gramática es posible hacer derivaciones arbitrariamente largas de una palabra tan sencilla como $()$, el subíndice de las flechas indica la regla utilizada:

$$S \Rightarrow_1 SS \Rightarrow_1 SSS \Rightarrow_1 ... \Rightarrow_3 SSS \Rightarrow_3 SS \Rightarrow_3 S \Rightarrow_2 (S) \Rightarrow_3 ()$$

Si se pudiera tener una gramática equivalente, pero sin reglas que produzcan la cadena vacía, ya no sería posible hacer derivaciones arbitrariamente largas. Esto puede ser una ventaja a la hora de determinar si una palabra se deriva o no de una gramática.

Eliminación de las reglas $A \to \varepsilon$

Considerando la gramática para los paréntesis bien balanceados. Si se quiere una GLC equivalente, pero sin emplear producciones vacías una opción sería analizar en reversa la derivación de donde viene la S que queremos cambiar por ε. Sólo hay otras dos reglas en la gramática, de modo que esa S tuvo que ser generada ya sea por $S \to (S)$ o por $S \to SS$. En el caso de $S \to (S)$, una solución es, en lugar de hacer la derivación

$$S \Rightarrow ... \Rightarrow \alpha S \beta \Rightarrow \alpha(S)\beta \Rightarrow \alpha(),\ \alpha \in \Sigma^*, \beta \in (\Sigma \cup V)^*$$

Mejor hacer directamente la derivación

$$S \Rightarrow ... \Rightarrow \alpha S \beta \Rightarrow \alpha()\beta \text{)β}$$

Agregando una regla $S \Rightarrow ()$ a la gramática. Y en caso de que la S provenga de la regla $S \Rightarrow SS$, se puede cambiar la derivación

$$S \Rightarrow ... \Rightarrow \alpha S \beta \Rightarrow \alpha SS \beta \Rightarrow \alpha S \beta$$

Por la derivación

$$S \Rightarrow ... \Rightarrow \alpha S \beta \Rightarrow \alpha S \beta$$

Usando una nueva regla $S \to S$, o, simplemente reemplazarla por

$$S \Rightarrow ... \Rightarrow \alpha S \beta$$

Sin ninguna regla adicional, la parte de la derivación $\alpha S \beta \Rightarrow \alpha S S \beta \Rightarrow \alpha S \beta$ desaparece por completo, pues no sirve de nada.

Resumiendo, la idea que permite eliminar las reglas $A \rightarrow \varepsilon$ es la de irse un paso atrás, para examinar de dónde provino el no-terminal A que queremos eliminar, y por cada regla $B \rightarrow \alpha A \beta$ de la gramática agregar una regla $B \rightarrow \alpha \beta$, en que directamente ya se reemplazó A por ε. Una vez hecho esto, se pueden suprimir todas las reglas de la forma $A \rightarrow \varepsilon$, pues resultan redundantes.

En efecto, la GLC original generaba la palabra vacía ε, mientras que la GLC transformada no la genera. Desde luego, el hecho de que una GLC contenga reglas de la forma $A \rightarrow \varepsilon$ no significa que el lenguaje contenga forzosamente a la palabra vacía.

En caso de que el lenguaje en cuestión realmente contenga a la palabra vacía, no es posible estrictamente eliminar todas las producciones vacías sin alterar el significado de la gramática. En estos casos se expresa el lenguaje como la unión $\{\varepsilon\} \cup L(G)$, donde G' es la gramática transformada.

3.1.7 Limitaciones de los LLC

Verificar que un lenguaje dado no es LLC, puede ser muy útil, para evitar el trabajo de tratar inútilmente de diseñar GLCs de lenguajes que no tienen ninguna. Una herramienta para esto es aplicar el llamado teorema de bombeo.

Teorema de bombeo

Existe para cada $G \in GL$ un número k tal que toda $\omega \in L(G)$, donde $|\omega| > k$, puede ser escrita como $\omega = uvxyz$ de tal manera que v y y no son ambas vacías, y que $uv^n y^n z \in L(G)$ para cualquier $m \geq 0$.

Este teorema dice que siempre hay manera de introducir discretamente sub-cadenas a las palabras de los LLC. Sirve para probar que ciertos lenguajes no son LLC. [1]

Prueba: Basta con probar que hay una derivación

$$S \Rightarrow^* uAz \Rightarrow^* uvAyz \Rightarrow^* uvxyz = \omega$$

pues al aparecer el mismo no-terminal en dos puntos de la derivación, es posible insertar ese segmento de la derivación cuantas veces se quiera, incluyendo cero.

Esa parte de la derivación, que tiene la forma $uAz \Rightarrow *uvAyz$, es una especie de ciclo sobre el no-terminal A.

Para probar que existen en la derivación ciclos de la forma $uAz \Rightarrow *uvAyz$, la idea será verificar que el tamaño vertical del árbol, su profundidad, es mayor que la cantidad de no-terminales disponibles. En consecuencia, algún no-terminal debe repetirse.

Primero, la cantidad de no-terminales para una gramática (V, Σ, R, S) es $|V|$.

Ahora se examina el problema de verificar si los árboles de derivación pueden tener una profundidad mayor que $|V|$.

Sea $m = \max(\{|\alpha| \mid A \to \alpha \in R\})$. Un árbol de profundidad p tiene a lo más m^r hojas, y por lo tanto un árbol A_ω para ω, con $|\omega| > m^r$ tiene profundidad mayor que R.

Así, toda palabra de longitud mayor que $M^{|V|}$ tendrá necesariamente una profundidad mayor que $|V|$, y por lo tanto, algún no-terminal estará repetido en la derivación; sea A ese no-terminal. Se representa el árbol de derivación en la figura 8.

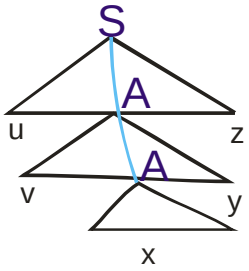


Figura 8. Árbol de derivación.

Como se ve, hay un subárbol del árbol de derivación, el triángulo intermedio en la figura 8, en el que el símbolo A es la raíz y también una de las hojas. Ese subárbol puede ser insertado o eliminado cuantas veces se quiera, y quedará siempre un árbol de derivación válido; cada vez que dicho subárbol sea insertado, las sub-cadenas v e y se repetirían una vez más. Esto completa la prueba. En la figura se aprecia porqué es importante que v e y no sean ambas vacías.

3.1.8 Propiedades de decisión de los LLC

Hay ciertas preguntas sobre los lenguajes libres de contexto y sus gramáticas que son posibles contestar, mientras que hay otras preguntas que no se pueden contestar en el caso general.

Iniciando con dos preguntas que sí se pueden contestar con seguridad y en un tiempo finito, es posible dar un algoritmo tal que, siguiéndolo paso por paso, se llega a concluir un sí o un no. Estos algoritmos se llaman algoritmos de decisión, pues permiten decidir la respuesta a una pregunta. Las preguntas que se contestarán son las siguientes:

Dadas una gramática G y una palabra ω, es posible decidir si $\omega \in L(G)$ cuando las reglas de G cumplen la propiedad: "Para toda regla $A \to \alpha$, $|\alpha| > 1$, o bien $\alpha \in \Sigma$, es decir, el lado derecho tiene varios símbolos, o si tiene exactamente un símbolo, éste es terminal."[20]

Prueba: La idea para probar el teorema es que cada derivación incrementa la longitud de la palabra, porque el lado derecho de las reglas tiene en general más de un símbolo. En vista de que la longitud de la palabra crece con cada paso de derivación, sólo hay que examinar las derivaciones hasta una cierta longitud finita. Por ejemplo, la gramática de los paréntesis bien balanceados cumple con la propiedad requerida:

1. $S \to ()$
2. $S \to SS$
3. $S \to (S)$

Como en esta gramática el lado derecho mide 2 o más símbolos, la aplicación de cada regla reemplaza un símbolo por dos o más. Por lo tanto, para saber si hay una derivación de la palabra $()(())$ que mide 6 símbolos, sólo se necesita examinar las derivaciones izquierdas de 5 pasos a lo más y que terminan en una palabra hecha únicamente de terminales. Estas derivaciones son:

1 paso:
$S \Rightarrow S$

2 pasos:
$S \Rightarrow (S) \Rightarrow (())$

3 pasos:
$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow ((()))$
$S \Rightarrow SS \Rightarrow ()S \Rightarrow ()()$

4 pasos:
$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow ((()))$
$S \Rightarrow (S) \Rightarrow (SS) \Rightarrow (()S) \Rightarrow (()())$
$S \Rightarrow SS \Rightarrow ()S \Rightarrow ()(S) \Rightarrow ()$

5 pasos:

$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow ((((S)))) \Rightarrow (((((\,)))))$

$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow ((SS)) \Rightarrow (((\,)S)) \Rightarrow (((\,)(\,)))$

$S \Rightarrow (S) \Rightarrow (SS) \Rightarrow (\,)S) \Rightarrow (\,)(S)) \Rightarrow (\,)(\,)))$

$S \Rightarrow (S) \Rightarrow (SS) \Rightarrow ((S)S) \Rightarrow (((\,))S) \Rightarrow (((\,))(\,))$

$S \Rightarrow SS \Rightarrow \,)S \Rightarrow \,)(S) \Rightarrow \,)((S)) \Rightarrow \,)(((\,)))$

$S \Rightarrow SS \Rightarrow \,)S \Rightarrow \,)SS \Rightarrow \,)\,)S \Rightarrow \,)\,)(\,)$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (\,))S \Rightarrow (\,))(S) \Rightarrow (\,))(\,))$

$S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (((\,)))S(((\,)))(\,)$

$S \Rightarrow SS \Rightarrow SSS \Rightarrow \,)SS) \Rightarrow \,)\,)S \Rightarrow \,)\,)(\,)$

Con base en este grupo de derivaciones es simple probar que la palabra $(\,)(\,)($ de 6 caracteres de longitud no pertenece al lenguaje generado por la gramática, pues si así fuera, estaría entre alguna de las palabras derivadas en 5 pasos o menos.

En el caso general se incluyen reglas de la forma $A \to \alpha$, con $\alpha \in \Sigma$. Para empezar se observa que las reglas de la forma $A \to \alpha$ producen exclusivamente un terminal, por lo que, en el peor caso, se aplicaron tantas reglas de este tipo como letras tenga la palabra generada. Por ejemplo, sea la gramática de las expresiones aritméticas: [8]

1. $E \to E + E$
2. $E \to E \times E$
3. $E \to x$
4. $E \to y$

Esta gramática tiene reglas, como $E \to x$ y $E \to y$ que tienen en su lado derecho un caracter. Entonces, dada una expresión aritmética como $x \times y + x$, que tiene 5 símbolos, a lo más se usan ese tipo de reglas en 5 ocasiones, se ve que en una derivación de $x \times y + x$ ese tipo de reglas se usa exactamente en 3 ocasiones.

Para generar 5 terminales con reglas de la forma $A \to \alpha a$ se requieren 5 no-terminales. Esos 5 no-terminales se generan con las reglas de la forma $A \to \alpha$, donde $|\alpha| > 1$. En el peor de los casos, $|\alpha| = 2$, por lo que se requerirán 4 pasos de derivación para llegar a los 5 no-terminales. Eso da un total de 9 pasos de derivación. Así, si se quiere determinar en forma segura si la palabra $x \times y + x$ pertenece o no al lenguaje generado por la gramática, sólo se tiene que examinar las derivaciones de longitud menor o igual a 9.

En general, para una palabra ω de longitud l hay que examinar las derivaciones de longitud hasta $2 \times l - 1$. Si la palabra se encuentra al final de alguna de esas

derivaciones, la palabra pertenece al lenguaje, y en caso contrario no pertenece al lenguaje. Esto termina la prueba del teorema.

En el enunciado del teorema no se está restringiendo a las GLC que satisfacen la condición: para toda regla $A \to \alpha$, $|\alpha| > 1$, o bien $\alpha \in \Sigma$, es decir, el lado derecho tiene varios símbolos, o si tiene exactamente un símbolo, éste es terminal. Cabe preguntar si esto constituye una limitación, en el sentido de que hay muchas GLC que no cumplen dicha condición. La respuesta es no, pues existe un procedimiento para pasar de una GLC arbitraria a una GLC que satisfaga la condición del teorema.

3.2 AUTÓMATA DE PILA

3.2.1 Funcionamiento de los autómatas de pila, AP

La pila funciona de manera que el último caracter que se almacena en ella es el primero en salir. Un aspecto crucial de la pila es que sólo se puede modificar su tope, que es el extremo por donde entran o salen los caracteres. Los caracteres a la mitad de la pila no son accesibles sin quitar antes los que están encima de ellos. La pila tiene un alfabeto propio, que puede o no coincidir con el alfabeto de la palabra de entrada. Esto se justifica porque puede ser necesario introducir en la pila caracteres especiales usados como separadores, según las necesidades de diseño del autómata.

Al iniciar la operación de un AP, la pila se encuentra vacía. Durante la operación del AP, la pila puede ir recibiendo y almacenando caracteres, según lo indiquen las transiciones ejecutadas. Al final de su operación, para aceptar una palabra, la pila debe estar nuevamente vacía.

(a) Incorrecto (b) Correcto

Figura 9. AP para el lenguaje $a^n b^n$

En los AP las transiciones de un estado a otro indican, además de los caracteres que se leen de la cadena de entrada, también lo que se saca del tope de la pila, y lo que se guarda en ella.

Antes de formalizar los AP, se utilizará una notación gráfica, como en los AP de las figuras 9 (a) y (b). Para las transiciones se usa la notación $\omega / \alpha / \beta$, donde ω

es la entrada, secuencia de caracteres, que se leen, α es lo que se saca de la pila, y β lo que se guarda en ella.

Se supone que primero se ejecuta la operación de sacar de la pila y luego la de guardar. Al igual que los AF, los AP tienen estados finales, que permiten distinguir cuando una palabra de entrada es aceptada.

Para que una palabra de entrada sea aceptada en un AP se deben cumplir todas las condiciones siguientes:

1. La palabra de entrada se debe haber leído totalmente.
2. El AP se debe encontrar en un estado final.
3. La pila debe estar vacía.

3.2.2 Diseño de AP

El problema de diseño de los AP consiste en obtener un AP M que acepte exactamente un lenguaje L dado.

Aunque en el caso de los AP no hay metodologías tan generalmente aplicables como era el caso de los autómatas finitos, siguen siendo válidas las ideas básicas del diseño sistemático, en particular establecer claramente qué es lo que recuerda cada estado del AP antes de trazar transiciones. Para los AP, adicionalmente se debe establecer una estrategia clara para el manejo de la pila.

A la hora de diseñar un AP se debe repartir lo que requiere ser recordado entre los estados y la pila. Distintos diseños para un mismo problema pueden tomar decisiones diferentes en cuanto a qué recuerda cada cual.

Ejemplo, Diseñar un AP que acepte exactamente el lenguaje con palabras de la forma $a^n b^n$, para cualquier número natural n.

Una opción es la de utilizar la pila como contador para recordar la cantidad de $a's$ que se consumen, y luego confrontar con la cantidad de $b's$. Una primera versión de este diseño utiliza un sólo estado q, con transiciones $a/\varepsilon/a$ y $b/a/\varepsilon$ de q a sí mismo, como en la figura 9(a).

Para verificar el funcionamiento del autómata, se puede simular su ejecución, listando las situaciones sucesivas en que se encuentra, mediante una tabla. Las columnas de la tabla de ejecución para un AP son: el estado en que se encuentra el autómata, lo que falta por leer de la palabra de entrada, y el contenido de la pila.

Por ejemplo, la tabla de ejecución del AP de la figura 9, para la palabra $aabb$, es:

Estado	Por leer	Pila
q	$aabb$	Z
q	abb	A
q	bb	Aa
q	b	A
q	ε	Z

Tabla 4. Tabla de ejecución.

Se concluye que el AP puede aceptar palabras como $a^n b^n$. Sin embargo, hay un problema: el AP también acepta palabras como $abab$, que no tienen la forma deseada. El problema viene de que no hemos recordado cuando se terminan las a y principian las b, por eso ha sido posible mezclarlas en $abab$. Una solución es utilizar los estados para memorizar las situaciones de estar consumiendo a o estar consumiendo b. El diagrama de estados correspondiente se muestra en la figura 9(b).

3.2.3 Definición formal Autómata pila

Un autómata de pila es un séxtuplo $(K, \Sigma, \Gamma, \Delta, s, F)$, donde:

- K es un conjunto de estados.
- Σ es el alfabeto de entrada
- Γ es el alfabeto de la pila
- $s \in K$ es el estado inicial
- $F \subseteq K$ es un conjunto de estados finales
- $\Delta \subseteq (K \times \Sigma^* \times \Gamma^*) \times (K \times \Gamma^*)$ es la relación de transición

La relación de transición aplica a cada estado, cada símbolo de entrada incluyendo la cadena vacía y cada símbolo del tope de la pila en un conjunto de posibles movimientos. Cada movimiento parte de un estado, un símbolo de la cinta de entrada y un símbolo del tope de la pila. El movimiento en sí consiste en un cambio de estado, en la lectura del símbolo de entrada y en la sustitución del símbolo del tope de la pila por una cadena de símbolos.

Las transiciones se interpretan de la siguiente forma: si $((q, \beta, \varphi), (p, \rho)) \in \Delta$ esto se lee así:

Si está en el estado $q \in K$, leyendo un elemento $\beta \in \Sigma$ de una cadena c y con $\varphi \in \Gamma$ en el tope de la pila, lea β descompile φ, compile $\rho \in \Gamma$ y pase al estado $p \in K$.

Existen transiciones más específicas, definidas a continuación:[21]

- $((q,\beta,\varepsilon),(p,\rho))$ si está en el estado q y la cadena que está leyendo tiene como prefijo β entonces, lea β, compile ρ y pase al estado p
- $((q,\beta,\rho),(p,\varepsilon))$ si está en el estado q y la cadena que está leyendo tiene como prefijo β y en el tope de la pila está ρ, entonces lea β, descompile ρ y pase al estado p sin compilar nada
- $((q,\beta,\varepsilon),(p,\varepsilon))$ si está en el estado q y la cadena que está leyendo tiene como prefijo β, lea β, pase a p y no compile nada
- $((q,\beta,\rho),(p,\rho\mu))$ si está en el estado q, lee β, y ρ está en el tope de la pila, pase a p compilando μ sobre ρ
- $((q,\beta,\rho),(p,\rho))$ si está en el estado q, lee β, y ρ está en el tope de la pila, pase a p dejando ρ en el tope de la pila

3.2.4 Relación entre AF y AP

Todo lenguaje aceptado por un AF es también aceptado por un AP, puesto que los AP son una extensión de los AF. [20]

Prueba: Sea (K,Σ,Δ,s,F) un AF; el AP $(K,\Sigma,\varnothing,\Delta',s,F)$, con $\Delta'=\{((p,u,\varepsilon),(q,\varepsilon))|\ (p,u,q)\in\Delta\}$ acepta el mismo lenguaje.

3.2.5 Relación entre AP y LLC

Se verifica el resultado por el que se inició el estudio de los AP, es decir, comprobar si son efectivamente capaces de aceptar los LLC.

Los autómatas de pila aceptan exactamente los LLC.

La prueba se puede dividir en dos partes:

1. Si M es un AP, entonces $L(M)$ es un LLC.
2. Si L es un LLC, entonces hay un AP M tal que $L(M)=L$

Se presenta únicamente la prueba con la parte 2, que se considera de mayor relevancia práctica. La otra parte de la prueba, que también es un procedimiento de conversión, puede consultarse en la referencia [19].

Sea una gramática $G=(V,\Sigma,R,S)$, entonces un AP M que acepta exactamente el lenguaje generado por G se define como sigue:

$$M=(\{p,q\},\Sigma,V\cup\Sigma,\Delta,p,\{q\})$$

donde Δ contiene las siguientes transiciones:

1. $((p,\varepsilon,\varepsilon),(q,S))$
2. $((q,\varepsilon,A),(q,x))$ para cada $A \to x \in R$
3. $((q,\sigma,\sigma),(q,\varepsilon))$ para cada $\sigma \in \Sigma$

Ejemplo.- Obtener un AP que acepte el LLC generado por la gramática con reglas:

1. $S \to aSa$
2. $S \to bSb$
3. $S \to c$

Las transiciones del AP correspondiente están dadas en la tabla siguiente:

1	$(p,\varepsilon,\varepsilon)$	(q,S)
2	(q,ε,S)	(q,aSa)
3	(q,ε,S)	(q,bSb)
4	(q,ε,S)	(q,c)
5	(q,a,a)	(q,ε)
6	(q,b,b)	(q,ε)
7	(q,c,c)	(q,ε)

Tabla 5. Transiciones del AP

El funcionamiento de este AP ante la palabra $abcba$ aparece en la siguiente tabla:

Estado	Falta leer	Pila
p	$abcba$	ε
q	$abcba$	S
q	$abcba$	aSa
q	$bcba$	Sa
q	$bcba$	$bSba$
q	cba	Sba
q	cba	cba
q	ba	ba
q	a	a
q	ε	ε

Tabla 6. Funcionamiento AP

Si se observa las transiciones del AP, se ve que solamente tiene dos estados, p y q, y que el primero de ellos desaparece del cálculo en el primer paso; de esto se concluye que el AP no utiliza los estados para recordar características de la entrada, y por lo tanto reposa exclusivamente en el almacenamiento de caracteres en la pila. En efecto, se puede ver que las transiciones del tipo 2, transiciones 2-4 del ejemplo, lo que hacen es reemplazar en la pila una variable por la cadena que

aparece en el lado derecho de la regla correspondiente. Dado que la única transición de tipo 1, transición 1 del ejemplo, coloca el símbolo inicial en la pila, a continuación lo que hacen las reglas de tipo 2 es realmente efectuar toda la derivación dentro de la pila de la palabra de entrada, reemplazando un lado izquierdo de una regla por su lado derecho. Una vez hecha la derivación de la palabra de entrada, la cual estaría dentro de la pila, sin haber aún gastado un sólo caracter de la entrada, se puede comparar caracter por caracter con la entrada, por medio de las transiciones de tipo 3.

Existe un problema técnico: al leer la palabra $abcba$, no se están aplicando las reglas en el orden descrito, esto es, primero la transición del grupo 1, luego las del grupo 2 y finalmente las del grupo 3, sino que más bien en la cuarta línea de la tabla se consume un caracter a, aplicación que corresponde a una transición del grupo 3, seguida de la aplicación de una transición del grupo 2. Esto no es casualidad; lo que ocurre es que las variables no pueden ser reemplazadas por el lado derecho de una regla si dichas variables no se encuentran en el tope de la pila; debido a que los AP sólo pueden acceder al caracter que se encuentra en el tope de la pila. Por esto, se hace necesario, antes de reemplazar una variable por la cadena del lado derecho de una regla, llevar dicha variable hasta que aparezca en el tope de la pila, lo cual puede hacerse consumiendo caracteres de la pila y de la entrada, mediante la aplicación de transiciones del tipo 3

De la construcción del AP descrito, se concluye con la siguiente proposición:

$$S \Rightarrow^* \omega \text{ si y sólo si } [[p, \omega, \varepsilon]] \vdash^*_{M(G)} [[q, \varepsilon, \varepsilon]]$$

donde $M(G)$ denota al AP construido a partir de la gramática G por el procedimiento descrito.

La prueba que para todo AP hay una gramática equivalente, se encuentra en [11].

La equivalencia de los AP y de las GLC permite aplicar todas las propiedades de los LLC para resolver problemas de diseño de AP.

4. LENGUAJE RECURSIVAMENTE ENUMERABLES Y MÁQUINA DE TURING

4.1 LENGUAJES RECURSIVAMENTE ENUMERABLES

Si L es un lenguaje decidible existe una máquina M, que siempre se detiene y que responde sí ante una palabra cualquiera $w \in L$. Los lenguajes decidibles también son conocidos como lenguajes recursivos

Un lenguaje L es recursivamente enumerable si existe una MT M que cuando es alimentada con alguna palabra $w \in L$, se detiene diciendo sí, pero su comportamiento no está definido cuando $w \notin L$

Todo lenguaje recursivo (decidible) es recursivamente enumerable, ya que un lenguaje recursivo es un caso particular de los lenguajes recursivamente enumerables.

El concepto de enumerabilidad recursiva nace del hecho de que si se tiene un lenguaje recursivamente enumerable, es posible construir una MT que corre sin entrada y que genera como salida a todas las palabras del lenguaje. Evidentemente, esta máquina no para sí el lenguaje es infinito, pero toda palabra w del lenguaje es generada como salida en tiempo finito.

4.2 MÁQUINA DE TURING

Turing propuso en los años 30 [22] un modelo de máquina abstracta, como una extensión de los autómatas finitos, que resultó ser de una gran simplicidad y poderío a la vez. La máquina de Turing es particularmente importante porque es la más poderosa de todas las máquinas abstractas conocidas

Han habido diversos intentos de encontrar otros modelos de máquinas u otros formalismos que sean más poderosos que las Maquinas de Turing, en el mismo sentido que las Máquinas de Turing son más poderosas que los Autómatas Finitos y los Autómatas de Pila; (se dice que un tipo de máquina MA es más poderoso que un tipo MB cuando el conjunto de lenguajes aceptados por alguna máquina en MB es un subconjunto propio de los aceptados por MA), pero todos los intentos han sido infructuosos al encontrarse que dichas extensiones son equivalentes en poder de cálculo a la MT original [11]; muchas de estas extensiones dotan de mayor flexibilidad al diseño de una Máquina de Turing que resuelve un problema en particular.

A. Turing propuso, en la llamada Tesis de Turing, que la noción de una máquina de Turing es una idealización matemática útil para probar que ciertas tareas no son automatizables o que ciertas funciones no son computables, una función es

computable si y sólo si hay una máquina de Turing que la computa, y que no podrá haber una máquina abstracta que calcule algo que la Máquina de Turing no pueda calcular [12].

4.2.1 Funcionamiento de la Máquina de Turing

La máquina de Turing tiene, como los autómatas que se han visto antes, un control finito, una cabeza lectora y una cinta donde puede haber caracteres y donde eventualmente viene la palabra de entrada. La cinta es de longitud infinita hacia la derecha, hacia donde se extiende indefinidamente, llenándose los espacios con el caracter blanco. La cinta no es infinita hacia la izquierda, por lo que hay un cuadro de la cinta que es el extremo izquierdo.

En la MT la cabeza lectora es de lectura y escritura, por lo que la cinta puede ser modificada en curso de ejecución. Además, en la MT la cabeza se mueve bidireccionalmente, izquierda y derecha, por lo que puede pasar repetidas veces sobre un mismo segmento de la cinta.

La operación de la MT consta de los siguientes pasos:

1. Lee un caracter en la cinta.
2. Efectúa una transición de estado.
3. Realiza una acción en la cinta.

Las acciones que puede ejecutar en la cinta la MT pueden ser:

1. Escribe un símbolo en la cinta.
2. Mueve la cabeza a la izquierda o a la derecha.

La palabra de entrada en la MT está escrita inicialmente en la cinta. Como la cinta es infinita, inicialmente toda la parte de la cinta a la derecha de la palabra de entrada está llena del caracter blanco, $(\_)$.

Por definición, al iniciar la operación de la MT, la cabeza lectora está posicionada en el caracter blanco, $(\_)$, a la izquierda de la palabra de entrada, el cual es el cuadro más a la izquierda de la cinta.

En la MT se llega al final de un cálculo cuando se alcanza un estado especial llamado *halt*, (detener), en el control finito, como resultado de una transición. Se representará al *halt* por h. Al llegar al *halt*, se detiene la operación de la MT, y se acepta la palabra de entrada. Así, en la MT no hay estados finales. En cierto sentido el *halt* sería entonces el único estado final, sólo que además detiene la ejecución.[1]

Cuando se quiere que una palabra no sea aceptada, desde luego se debe evitar que la MT llegue al *halt*. Esto se puede asegurar haciendo que la MT caiga en un ciclo infinito.

El lenguaje aceptado por una MT es simplemente el conjunto de palabras aceptadas por ella.

Al diseñar una MT que acepte un cierto lenguaje, en realidad se diseña el autómata finito que controla la cabeza y la cinta, el cual es un autómata con salida. Así, se puede usar la notación gráfica utilizada para aquellos autómatas para indicar su funcionamiento. En particular, cuando se traza una flecha que va de un estado p a un estado q con nombre σ / L, quiere decir que cuando el caracter leído por la cabeza de la MT, es σ, la cabeza lectora hace un movimiento a la izquierda, indicada por el caracter L (*left*); similarmente cuando se tiene una flecha con σ / R el movimiento es a la derecha.

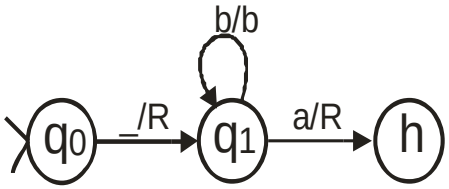


Figura 10. MT que acepta palabras que empiezan con a

Ejemplo: Diseñar una MT que acepte las palabras en $\{a,b\}$ que comiencen con a. La solución se muestra en la figura 10. Si la primera letra es una a, la palabra se acepta, y en caso contrario se hace que la MT caiga en un ciclo infinito, leyendo y escribiendo b. La acción inmediatamente antes de caer en el *halt* es irrelevante; igual se podía haber puesto a / a o a / R como nombre de la flecha.

Ejemplo: Probar que hay lenguajes que no son libres de contexto, pero que pueden ser aceptados por una máquina de Turing, el lenguaje $a^n b^n c^n$, que se sabe que no es LLC.

La estrategia para el funcionamiento de dicha MT consiste en recorrer la palabra, descontando en cada una de ellas una a, una b y una c; para descontar esos caracteres simplemente se reemplazan por un caracter _. Cuando ya no se encuentre ninguna a, b o c en algún recorrido, si queda alguna de las otras dos letras la palabra no es aceptada; en caso contrario se llega a *halt*. Es útil, antes de emprender el diseño de una MT, tener una idea muy clara de cómo se quiere que funcione. Para eso se detalla el funcionamiento con un ejemplo para la palabra $aabbcc$. La posición de la cabeza se indica sombreada.

_	a	a	b	b	c	c	_
_	a	a	b	b	c	c	_
_	*	a	b	b	c	c	_
_	*	a	b	b	c	c	_
_	*	a	b	b	c	c	_
_	*	a	*	b	c	c	_
...	...	...	...	...	...	...	...
_	*	a	*	b	*	c	_
...	...	...	...	...	...	...	...
_	*	a	*	b	*	c	_
...	...	...	...	...	...	...	...
_	*	a	*	b	*	c	_
...	...	...	...	...	...	...	...

Tabla 7. Funcionamiento de una MT leyendo la palabra $aabbcc$.

4.2.2 Formalización de la MT

Una MT es un quíntuplo $(K, \Sigma, \Gamma, \delta, s)$ donde:

- K es un conjunto de estados tal que $h \in K$
- Σ es el alfabeto de entrada, donde $\_ \in \Sigma$
- Γ es el alfabeto de la cinta, donde $\_ \in \Gamma$ y $\Sigma \subseteq \Gamma$
- $s \in K$ es el estado inicial
- $\delta : (K - \{h\} \times \Gamma) \rightarrow K \times (\Gamma \cup \{L, R\})$ es la función de transición

La expresión de la función de transición se entiende como: la función de transición del control finito debe considerar como entradas el estado actual, que es un elemento de K, pero que no puede ser h, así como el caracter leído en la cinta, que es elemento de Γ. Por eso a la izquierda de la flecha aparece la expresión $\delta : (K - \{h\} \times \Gamma)$ $\delta : (K - \{h\} \times \Gamma)$. Luego, el resultado de la función de transición debe incluir el siguiente estado, que es elemento de K. Otro resultado de la función de transición son las acciones a ejecutar por la MT, una escritura, y un movimiento, que puede ser a derecha, izquierda o conservar la posición actual.[1] La acción mover cabeza a la izquierda se representa por el símbolo L, R para la derecha y H para conservar la posición actual sobre la cinta. En el caso de la escritura se indica el caracter que se escribe, el cual es un elemento de Γ. Desde luego, para que no haya confusión se requiere que ni L ni R ni H estén en Γ.Resumiendo, el resultado de la función de transición debe ser un elemento de $K \times (\Gamma \cup \{L, R, H\})$.

Así, si $\delta(q, a) = (b, L, p)$, donde $b \in \Gamma$, esto quiere decir que estando la MT en el estado q con la cabeza lectora sobre un caracter a, la función de transición

enviará al autómata a un estado p, escribirá el caracter b y hará un movimiento a la izquierda

4.2.3 Configuración

En las MT la configuración resume la situación en que se encuentra la MT en cualquier punto intermedio de un cálculo, de manera tal que con sólo las informaciones contenidas en la configuración se pueda reconstruir dicha situación y continuar el cálculo. [25]

Las informaciones necesarias para resumir la situación de una MT en medio de un cálculo son:

- Estado en que se encuentra la MT
- Contenido de la cinta
- Posición de la cabeza

El problema es cómo representar formalmente cada uno de los tres componentes de la configuración.

No hay problema con el estado en que se encuentra la MT, que es directamente un elemento de K. Respecto al contenido de la cinta, existe la dificultad de que como es infinita, no se puede representar toda por una cadena de caracteres, que siempre será de tamaño finito. La solución es tomar en cuenta únicamente la parte de la cinta hasta antes de donde empieza la sucesión infinita de blancos, pues esta última no contiene ninguna información útil.

El siguiente problema es cómo caracterizar la posición de la cabeza lectora. La solución es representar la posición por un número entero que indica la posición actual con respecto a alguna referencia. Sin embargo, se adopta la solución que consiste en dividir la cinta dentro de la configuración en tres pedazos:

- La parte de la cinta a la izquierda de la cabeza, que es un elemento de Γ^*
- El cuadro en la posición de la cabeza lectora, que es un elemento de Γ
- La parte de la cinta a la derecha de la cabeza lectora, hasta antes de la sucesión de blancos que se extiende indefinidamente a la derecha

La parte a la derecha de la cabeza lectora es un elemento de Γ^*, pero se puede hacer una mejor caracterización de ella considerando que el último caracter de ella no es blanco. Así, sería un elemento de $\Gamma^*(-\{\_\})$. Esta expresión no incluye la cadena vacía, la cual puede producirse cuando todos los caracteres a la derecha de la cabeza son blancos. La solución es añadir este caso, por lo que finalmente la parte a la derecha de la cabeza lectora es un elemento de $\Gamma^*(-\{\_\}) \cup \{\varepsilon\}$.

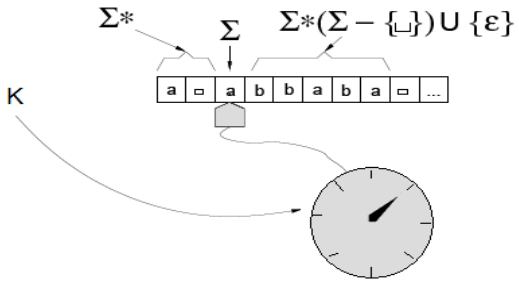


Figura 11. Configuración en MT

Finalmente, la configuración es un elemento de:

$$K \times \Gamma^* \times \Gamma \times (\Gamma^*(\Gamma - \{\_\})\{\varepsilon\}$$

Las configuraciones se indican encerradas entre dobles corchetes, como en $[[q, aa, a, bb]]$ que indica que la MT en cuestión se encuentra en el estado q, habiendo a la izquierda de la cabeza una cadena aa, bajo la cabeza una a, y a su derecha, antes de la secuencia infinita de blancos, una cadena bb. Para simplificar aún más la notación, se puede indicar por un caracter subrayado la posición de la cabeza lectora; así en vez de tener cuatro componentes la configuración tendrá únicamente dos.

Relación entre configuraciones

Se define una relación binaria $C_1 \vdash C_2$ que nos indica que la MT puede pasar de la configuración C_1 a la configuración C_2.

La relación $\vdash$ en $C \times C$, donde C es el conjunto de configuraciones, se define por casos:

Caso escritura:

$[[p, \omega, a, u]] \vdash [[q, \omega, b, u]]$

Si y sólo si $\delta(p, a) = (q, b)$, donde $b \in \Gamma$

Caso de movimiento a la izquierda, parte derecha no vacía:

$[[p, \omega d, a, u]] \vdash [[q, \omega, d, au]]$

Si y sólo si $\delta(p, a) = (q, L)$, donde $a \neq \_$ o bien $u \neq \varepsilon$

Caso de movimiento a la izquierda, parte derecha vacía:

$[[p, \omega d, \_, \varepsilon]] \vdash [[q, \omega, d, \varepsilon]]]$

Si y sólo si $\delta(p, \_) = (q, L)$

Caso de movimiento a la derecha, parte derecha no vacía:

$[[p,\omega,a,du]\vdash [[q,\omega a,d,u]]$

Si y sólo si $\delta(p,a)=(q,R)$

Caso de movimiento a la derecha, parte derecha vacía:

$[[p,\omega,a,\varepsilon]\vdash [[q,\omega a,\_,\varepsilon]]$

Si y sólo si $\delta(p,a)=(q,R)$

Configuración colgada

En el caso de que la cabeza lectora se encuentre en el cuadro de la cinta más a la izquierda, y se trate de hacer un movimiento a la izquierda, se produce un error llamado configuración colgada, que tiene como consecuencia que la MT no pueda seguir funcionando, y desde luego no podrá ser aceptada la palabra de entrada.

Formalmente, si se tiene una configuración de la forma $[[p,\varepsilon,a,u]]$ y la transición es $\delta(p,a)\omega=(q,L)$, no existe una configuración C tal que $[[p,\varepsilon,a,u]\vdash C$.

En general se evitará el uso intencional de las configuraciones colgadas, de modo que si no se quiere que una palabra sea aceptada, se hará que la MT se cicle en vez de colgarse.

4.2.4 Cálculos en MT

En las MT un cálculo es una secuencia C1, $C_1,C_2...C_n$ de configuraciones tal que $C\vdash C_{i+1}$. Un cálculo puede ser visto en términos computacionales como un gráfico de ejecución, que describe de una manera muy exacta la forma en que una MT responde ante una entrada en particular.

Por ejemplo, sea la MT siguiente: $K=\{s\}$, $\Sigma=\{a,\_\}$, $\delta(s,a)=(s,R)$, $\Sigma\delta(s,\_)=(h,\_)$. Ante la configuración $[[s,a,a,aa]]$ se presenta el cálculo siguiente:

$$[[s,a\boxed{a}aa]\vdash [[s,aa\boxed{a}a]\vdash [[s,aaa\boxed{a}]\vdash [[s,aaaa\boxed{\_}]\vdash [[s,aaaa\boxed{h}]]$$

Se puede llegar de una configuración C_i a C_j, para $i\le j$ en cero o varios pasos; esto se indica en forma compacta utilizando la cerradura reflexiva y transitiva de la relación $\vdash$, denotada por $\vdash^*$, quedando $C\vdash^* C_j$.

4.2.5 Palabra aceptada

Formalmente las nociones de palabra aceptada y lenguaje aceptado son:

Una palabra $\omega \in \Sigma^*$, es aceptada por una MT M si $[[s, \varepsilon, \_, \omega]] \vdash^* [[h, \alpha, a, \beta]]$ donde $a, \beta \in \Gamma$. Como se ve, el único criterio para que la palabra de entrada ω se acepte es que se llegue a *halt* en algún momento, independientemente del contenido final de la cinta, el cual es visto como basura.

4.2.6 MT para cálculos de funciones

Se ha visto las MT como analizadoras de palabras cuyo fin es determinar si la palabra de entrada pertenece o no al lenguaje aceptado. Sin embargo, las MT también pueden ser utilizadas para calcular resultados u operaciones a partir de la entrada. En vez de considerar como basura el contenido de la cinta al llegar al *halt*, se podría verlo como un resultado calculado. Para poder interpretar sin ambigüedad el contenido final de la cinta como resultado, se requiere que cumpla con un formato estricto, caracterizado por los siguientes puntos: [23]

- La palabra de salida no debe contener ningún caracter blanco (_)
- La palabra de salida comienza en el segundo caracter de la cinta, teniendo a su izquierda un blanco y a su derecha una infinidad de blancos
- La cabeza estará posicionada en el primer blanco a la derecha de la palabra de salida

Se puede apreciar que el formato para la palabra de salida es muy similar al de la palabra de entrada, salvo que en la primera, la cabeza está posicionada en el caracter a la derecha de la palabra.

Ejemplo. Suponiendo la función reverse, que invierte el orden en que aparecen las letras en la palabra de entrada; así, $reverse(aabb) = bbaa$. Si inicialmente el contenido de la cinta es de la forma $\boxed{a}abb...$, donde el caracter resaltado indica la posición de la cabeza, la cinta al final debe quedar como: $\_bbaa\boxed{\_}...$

Es muy importante seguir este formato, y no caer en ninguno de los siguientes errores:

- Aparece algún espacio blanco dentro del resultado, como en la cinta $\_bbaa\_abt\boxed{\_}...$
- El resultado no está posicionado empezando en el segundo cuadro de la cinta, como en $\_\_\_bbaa\boxed{\_}...$
- La cabeza no está ubicada exactamente en el cuadro a la derecha del resultado, como en la cinta $\_bba\boxed{a}...$
- Aparece basura (caracteres no blancos) en la cinta, a la derecha o izquierda del resultado, como en la cinta $\_\_\_bbaa\_\_\_b\_...$

Para precisar estas nociones, se utiliza la noción formal de configuración: Una MT calcula un resultado $u \in \Sigma*$ a partir de una entrada $\omega \in \Sigma*$ si:

$$[[s, \varepsilon, \_, \omega]] \vdash *[[h, u, \_, \varepsilon]]$$

Las funciones en matemáticas sirven para describir la relación entre un resultado y una entrada. Se puede relacionar esta noción con la definición anterior: Una MT M calcula una función f: $\Sigma* \rightarrow \Sigma*$ si para toda entrada ω, M calcula un resultado u tal que $f(\omega) = u$.

Si hay una MT que calcula una función f, se dice que f es Turing-calculable.

Ejemplo. Construir una máquina de Turing que reste dos números naturales en unario, esto es, $f(x, y) = x - y$. Como las MT reciben un sólo argumento, para realizar una función de dos argumentos como la resta en realidad se recibe un sólo argumento que contiene un símbolo para separar dos partes de la entrada. La cabeza lectora al final debe estar posicionada en el blanco a la derecha del residuo. En caso de que el sustraendo sea mayor que el minuendo, el resultado es cero.

La estrategia para construir esta MT es ir descontando cada 1 del minuendo contra otro 1 del sustraendo, reemplazando ambos por un caracter arbitrario sea *. Cuando se termine el sustraendo, se borran los caracteres inútiles de manera que queden sólo los restos del minuendo. Para evitar tener que recorrer el residuo, se descuentan caracteres del minuendo de derecha a izquierda. Es decir, se tendría una secuencia de configuraciones de la cinta como las siguientes, la última línea indica la configuración en la que debe dar *halt*.

_	1	1	1	"-"	1	1	_
_	1	1	1	"-"	1	1	_
...	...	...	...	...	...	...	...
_	1	1	1	"-"	1	1	_
_	1	1	1	"-"	1	1	_
_	1	1	1	"-"	1	*	_
...	...	...	...	...	...	...	...
_	1	1	*	"-"	1	1	_
...	...	...	...	...	...	...	...
_	1	*	*	"-"	*	*	_
...	...	...	...	...	...	...	...
_	1	_	_	_	_	_	_

Tabla 8. Configuración de la cinta de una MT que resta dos numero naturales en unario.

4.2.7 Problemas de decisión

Un caso particular de funciones es aquel en que el resultado sólo puede ser sí o no. Si se representa el sí con 1 y el no con 0, se está considerando funciones $g:\Sigma^* \to \{1,0\}$. En este caso, la MT sirve para decidir si la entrada tiene una propiedad P o no la tiene.

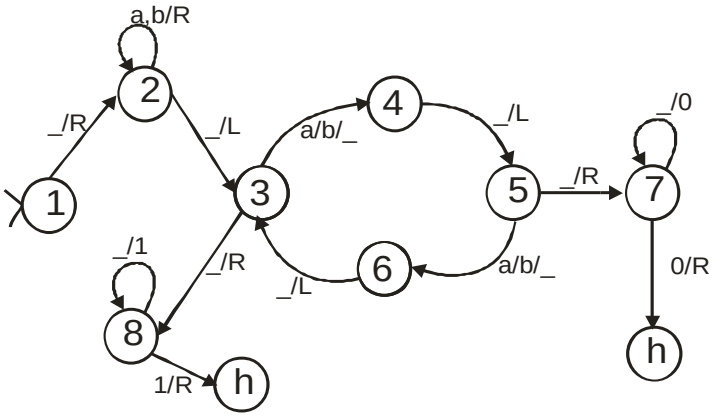


Figura 12. MT que decide si la entrada es de longitud par.

Un diseño para la MT que decide si una entrada en el alfabeto $\Sigma\{a,b\}$ es de longitud par aparece en la figura 12. La estrategia en este diseño es primero recorrer la cabeza al extremo derecho, y luego ir borrando los caracteres de entrada, de derecha a izquierda, y recordando mediante los estados 3 y 5 si la cantidad de letras es, hasta el momento, par o impar, respectivamente. Al terminar de borrar la palabra de entrada, según que se haya terminado en el estado 3 o 5, se escribe 1 o 0 en la cinta, y se llega a *halt*.

Se dice que un lenguaje L es Turing-decidible si hay alguna MT que entrega un resultado 1 si la entrada ω está en L, y un resultado 0 en caso contrario. [1]

Para que una MT entregue como resultado 1 o 0, es condición indispensable que la palabra de entrada haya sido aceptada. Esto tiene la consecuencia siguiente:

Un lenguaje es Turing-decidible solamente si es Turing-aceptable

Si un lenguaje no es Turing-decidible se dice que es indecidible

Relación entre aceptar y decidir

Las siguientes propiedades que relacionan Turing-decidible con Turing-aceptable son útiles para comprender mejor ambas nociones:

1. Todo lenguaje Turing-decidible es Turing-aceptable.

2. Si L es Turing-decidible, L^c es Turing-decidible.
3. L es decidible si y sólo si L y L^c son Turing-aceptables.

La prueba de 1 es muy sencilla, para decidir un lenguaje L, la MT debe primero llegar al *halt* para toda palabra de $\omega \in L$, con lo que necesariamente acepta ω.

El punto 2 es sencillo, pues dada una MT M que decide el lenguaje L, se produce una máquina M_0 que decide L^c cambiando en M el resultado 1 por 0 y viceversa.

La prueba de 3 es más complicada. No se probará que si L y L^c son Turing aceptables entonces L es decidido por alguna MT, sino más bien que hay un procedimiento mecánico para decidir L. Por la llamada Tesis de Church, ambos enunciados son equivalentes. Suponiendo que se tiene dos MT, M y M^c, que aceptan respectivamente los lenguajes L y L^c Se pone a funcionar ambas máquinas "en paralelo", analizando ambas la misma palabra ω. Ahora bien, si $\omega \in L$, eventualmente M llegará al *halt*. Si $\omega \notin L$, entonces $\omega \in L^c$, y en algún momento M^c se detendrá. Ahora se considera una MT adicional M^*, que "observa" a M y a M^c y que si M se para, entrega una salida 1, mientras que si M^c, se para, entrega una salida 0. Para toda palabra ω, M^* decidirá 1 o 0, por lo que el lenguaje es decidible.

4.3 MT EN LA JERARQUÍA DE CHOMSKY

En conclusión, las MT no son capaces de aceptar todos los lenguajes posibles en 2^{Σ^*}. Este hecho puede ser establecido a partir de la enumerabilidad de las MT: pues las MT son cuádruplos (K, Σ, δ, s) y, por lo tanto elementos de un producto cartesiano, al ser enumerable cada uno de los componentes necesariamente el cuádruplo es también enumerable. En efecto:

- Los conjuntos de los estados posibles son enumerables si estandarizamos los nombres de los estados a q_0, q_1, q_2, etc., lo cual evidentemente no altera ningún aspecto del funcionamiento de la MT
- Similarmente, un alfabeto estándar σ_0, σ_1, σ_2, etc., puede codificar cualquier alfabeto en particular. Así, también los alfabetos son enumerables
- La función de transición es parte de otros productos cartesianos de estados y caracteres, por lo que es también enumerable

Los estados iníciales trivialmente son enumerables, siguiendo la estandarización del primer punto.

Ahora bien, al ser enumerables las MT, resulta que no puede mapearse un elemento de 2^{Σ^*} con una MT distinta, y por lo tanto hay lenguajes que no tienen una MT que los acepte.

Desde luego, el resultado anterior no ayuda a localizar exactamente qué lenguajes no son aceptados por ninguna MT; esto ya se había hecho para algunos lenguajes en la sección precedente.

Resulta útil entonces ubicar la parte de los lenguajes que sí pueden aceptar las MT con respecto a otras clases de lenguajes, siguiendo la estructura de clases de lenguajes llamada jerarquía de Chomsky. [7]

Recordando la jerarquía de Chomsky, que clasifica los lenguajes en categorías, y la forma en que se asocian distintos tipos de máquinas a dichas categorías de lenguajes:

TIPO DE AUTÓMATA	LENGUAJE QUE SE PROCESA	GRAMATICA QUE LO GENERA
Autómatas Finitos	Lenguajes Regulares	Gramáticas Regulares
Autómatas de Pila	Lenguajes Libres de Contexto	Gramáticas Libres de Contexto
Autómatas linealmente acotados	Lenguajes sensitivos al Contexto	Gramáticas Sensitivas al contexto
Máquina de Turing decidiendo	Lenguajes Recursivos	
Máquina de Turing aceptando	Lenguajes Recursivos Enumerables	Gramáticas no Restringidas

Tabla 9. Jerarquía de Chomsky

En esta tabla se han diferenciado la clase de lenguajes que pueden ser decididos por una MT, que son llamados recursivos, de los lenguajes que pueden ser aceptados por una MT, que son los "recursivamente enumerables, aunque no hemos definido ninguno de ellos más que por su relación con las MT. [6]

De acuerdo con la presentación de la jerarquía de Chomsky, las MT son equivalentes en poder de cálculo a las gramáticas no restringidas. La prueba de esto puede ser consultada en diversas referencias [11], [1].

[6] "Recursivamente enumerables" es solamente otro nombre para Turing aceptable, usado en textos como [1].

Así, de acuerdo con la Tesis de Church, los lenguajes recursivamente enumerables son el más extenso conjunto de lenguajes que pueden ser algorítmicamente analizados.

4.4 INTRODUCCIÓN A LA COMPUTABILIDAD

En la teoría de la computabilidad, las funciones Turing-computables o funciones computables son, según la tesis de Church-Turing, aquellas funciones que se pueden calcular utilizando una máquina de cálculo. Esta noción puede ser relativa a cualquier conjunto A de números naturales, o equivalente a una función cualquiera f de los números naturales a los números naturales, por medio de máquinas de Turing extendidas. Estas funciones pueden ser llamadas A-computables o f-computables respectivamente.

Las funciones computables se usan para discutir sobre computabilidad sin tener que referirse a ningún modelo de computación en concreto, como la máquina de Turing o la máquina de registros. En este apartado, se pueden utilizar los axiomas de Blum para definir una teoría de complejidad computacional sobre el conjunto de las funciones computables. Según la tesis Church-Turing, la clase de las funciones computables es equivalente a la clase de las funciones definidas por funciones recursivas, calculo de λ (lambda), o algoritmos de Markov.

El objetivo de la teoría de la computabilidad es concretar la noción de función calculable, función/expresión cuyos valores se pueden calcular de forma automática a partir de un algoritmo, y ha encontrado resultados positivos y negativos (resultados de no computabilidad o de indecibilidad).

La teoría de la computabilidad se puede caracterizar por la búsqueda de respuestas a las siguientes cuestiones:

1. ¿Qué pueden hacer los computadores actuales sin limitaciones de espacio, tiempo o dinero?
2. ¿Cuáles son estas limitaciones relacionadas con el cálculo?

Para resolver las anteriores cuestiones, se realizaron los estudios de ciertos modelos de computación, los cuales, arrojaron el concepto de algoritmo. Estos estudios realizados a principios del siglo 20 fueron elaborados, entre otros, por Post, Church, Turing, Kleene y Gödel. Estos trabajos han influido profundamente en el desarrollo de la computación tanto en aspectos teóricos como en aspectos prácticos, como pueden ser la posibilidad de interpretar programas, la existencia de computadores de propósito general, la representación de lenguajes mediante estructuras formales basados en reglas de producción y la dualidad entre hardware y software.

4.4.1 Tesis de Church

A. Turing propuso, en la Tesis de Turing, que todo aquello que puede ser calculado, podrá ser calculado en una MT, y que no podrá haber una máquina abstracta que calcule algo que la MT no pueda calcular [12]. Más aún, A. Church, inventor del cálculo lambda, uno de los sistemas competidores de la MT, propuso la conjetura de que en realidad no puede haber ningún modelo de cómputo más poderoso que los desarrollados hasta entonces, que incluían la MT, su cálculo lambda, así como otras máquinas abstractas, como la máquina de Post.

Hasta hoy la llamada “tesis de Church” no ha podido ser probada ni refutada.
La tesis de Church, sin embargo, no se considera un teorema que pudiera ser eventualmente probado, sino simplemente una hipótesis de trabajo.

4.4.2 Comparación de las MT con otras máquinas

Se pueden considerar comparaciones de la MT con:

1. Extensiones a la MT
    a) MT con varias cintas, varias cabezas
    b) MT con no determinismo
2. Otras máquinas de cinta
3. Otros paradigmas, máquinas de Post, Gramáticas

Estas comparaciones pueden ser encontradas en la referencia [11].

4.4.3 Límites de la MT

Hay problemas que no se pueden resolver como una secuencia determinista de operaciones elementales, que es lo esencial de las MT. Estos problemas son llamados algorítmicamente irresolubles.

Concentrando la atención en problemas del tipo: dados una palabra ω y, la descripción de, un lenguaje L, decidir si $\omega \in L$, que son llamados problemas de pertenencia de palabras. Se dice que un lenguaje L es decidible si hay una MT para decidir el problema de la pertenencia de palabras. Muchos otros problemas que no son del tipo mencionado pueden sin embargo expresarse en términos de éstos mediante una transformación adecuada; por ejemplo, el problema de determinar si dos gramáticas G_1 y G_2 son equivalentes, puede expresarse de la manera siguiente: Para toda $\omega \in L(G_1)$, decidir si $\omega \in L(G_2)$.

El problema del paro de MT

Una vez que se cuenta con un primer problema irresoluble, la prueba de que otros problemas son irresolubles consiste en probar que éstos pueden ser reducidos al problema de referencia. Este primer problema irresoluble es el del paro de la MT.

El problema del paro de la MT consiste en determinar algorítmicamente, esto es, mediante una MT, si una MT dada M va a parar o no cuando analiza la palabra de entrada ω.

Como una MT analiza el comportamiento de otra, se requiere que esta última sea dada como entrada a la primera; esto puede ser hecho mediante una codificación de la MT que debe analizarse. Una manera simple de codificar una MT es considerando la cadena de símbolos de su representación como cuádruplo (K,Σ,δ,s). $d(M)$ es la codificación de una MT M.

Teorema: No existe ninguna MT tal que, para cualquier palabra ω y cualquier MT M, decida si $\omega \in L(M)$.

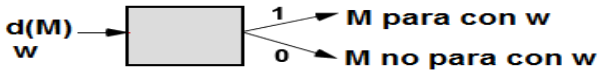
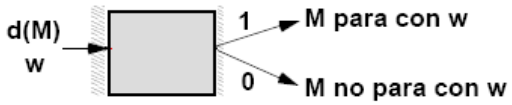
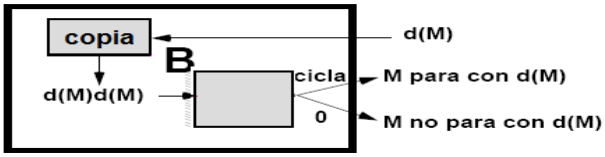


Figura 13. Problema de paro de una MT

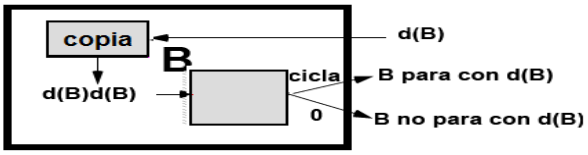
En la figura 13 se muestra cómo debería funcionar la MT que resolvería el problema del paro.



(a)Máquina A



(b) Máquina B



(c) Contradicción

Figura 14. Prueba de paro de MT.

Prueba: [7] Por contradicción. Sea A la MT de la figura 14(a). Entonces se construye otra MT B, como se representa en la figura 14(b), esto es, se tiene una única entrada con la codificación $d(M)$ de la MT M, y se pasa esta palabra a una MT copiadora, que duplica la entrada $d(M)$. La salida de la copiadora será dos veces $d(M)$. Esto es pasado como entrada a una máquina A' que es A modificada [8] de la siguiente manera: a la salida 1 de A la cambiamos de forma que en vez de dar el *halt* se cicle; esto siempre puede hacerse. Ahora bien, comparando A con A' se ve que la salida 1 corresponde al hecho de que M para con $d(M)$.

Finalmente se supone que se aplica la máquina B a una entrada formada por la misma máquina codificada, esto es, $d(B)$. Entonces cuando B se cicla, esto corresponde a la salida que indica que "B se para con $d(B)$", lo cual es contradictorio. Similarmente, B entrega un resultado 0, esto es, se para, en el caso que corresponde a "B no se para con $d(B)$", que también es contradictorio. Esto se ilustra en la figura 14(c).

Utilizando el problema del paro de la MT como referencia, se ha probado que otros problemas son también insolubles. Entre los más conocidos, se tienen:

- El problema de la equivalencia de las gramáticas libres de contexto
- La ambigüedad de las GLC
- El problema de la pertenencia de palabras para gramáticas sin restricciones

Las pruebas de estos resultados se encuentran en las referencias [11], [1].

[7] Esta prueba es debida a M. Minsky [14], aunque la primera prueba data de Turing [22].

[8] Obsérvese que la segunda repetición de $d(M)$ es de hecho la palabra ω que se supone que es sometida a M.

5. DESARROLLO DEL ENTORNO DE SIMULACIÓN PARA AUTÓMATAS DETERMINISTAS

Después de estudiar la teoría relacionada con máquinas abstractas, se muestra el desarrollo de un entorno de simulación, reconocedor de lenguajes, para autómatas deterministas en el ambiente gráfico LabVIEW™.

Este entorno de simulación está compuesto por las máquinas de estados finitos, de pila y de Turing, a continuación se muestra el desarrollo de cada uno de ellos.

5.1 AUTÓMATA DE ESTADOS FINITOS:

El funcionamiento de un autómata finito es básicamente una serie de estados en los que se avanza de uno a otro por medio de unas transiciones, las cuales son las líneas que unen dichos estados y que representan los diferentes caracteres que se pueden leer de la palabra de entrada. El buen funcionamiento de este autómata depende de realizar bien las transiciones, escoger el estado inicial y la buena elección del estado final o de aceptación.

Se puede mostrar su funcionamiento más claramente mediante un ejemplo, el cual consiste en realizar un autómata de estados finitos que acepte las palabras que contengan la cadena 0122.

Se realiza el diagrama de estados comenzando en el estado inicial y buscando el camino correcto hasta llegar al estado final o de aceptación, luego se observa qué otras posibilidades hay en el camino (caminos incorrectos) y se llevan a estados que no son de aceptación.

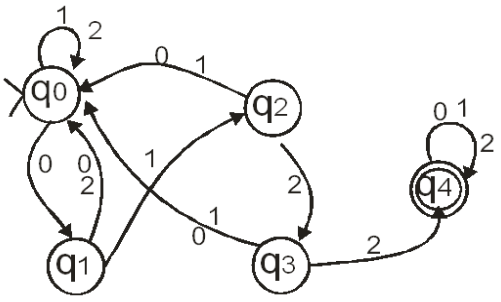


Figura 15. Diagrama de estados.

En este diagrama se observa el estado inicial (0), el estado final (4) y se determinan las transiciones, que son las siguientes:

	0	1	2	3	4
0	1	0	0	0	4
1	0	2	0	0	4
2	0	0	3	4	4

Tabla 10. Transiciones del autómata reconocedor de palabras que contienen la cadena 0122.

Estas transiciones se leen así:

Estando en el estado 0 (estado inicial) y leyendo un cero (0) de la cadena de entrada, la acción respectiva es avanzar al estado 1, si estando en este mismo estado el caracter leído es 1 ó 2, la acción respectiva es permanecer en el estado 0, así se continua hasta llegar al estado 4.

Con base en esto se construyó el simulador para autómatas finitos en LabVIEW™, cuyo algoritmo,
Figura **17**, se trata de saber en qué estado se encuentra el autómata y que caracter está leyendo de la palabra de entrada, con estos dos datos realiza la acción respectiva la cual puede ser avanzar a un nuevo estado ó quedarse en el mismo, acción declarada anteriormente en las transiciones. Este proceso implementado en LabVIEW™ se puede visualizar de una forma más clara en el siguiente diagrama, Figura 16.

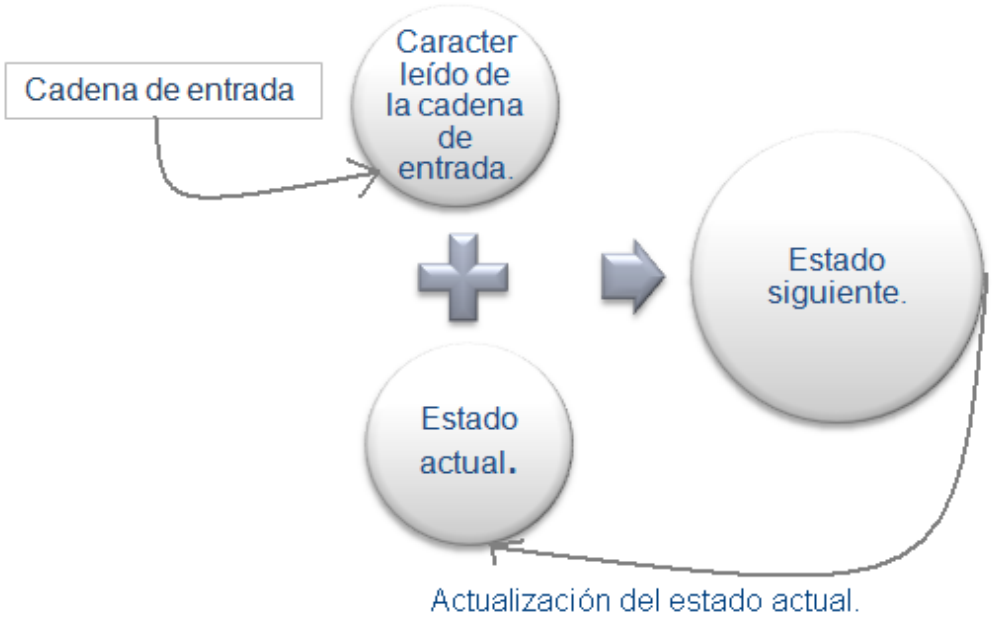


Figura 16. Diagrama de bloques autómata finito

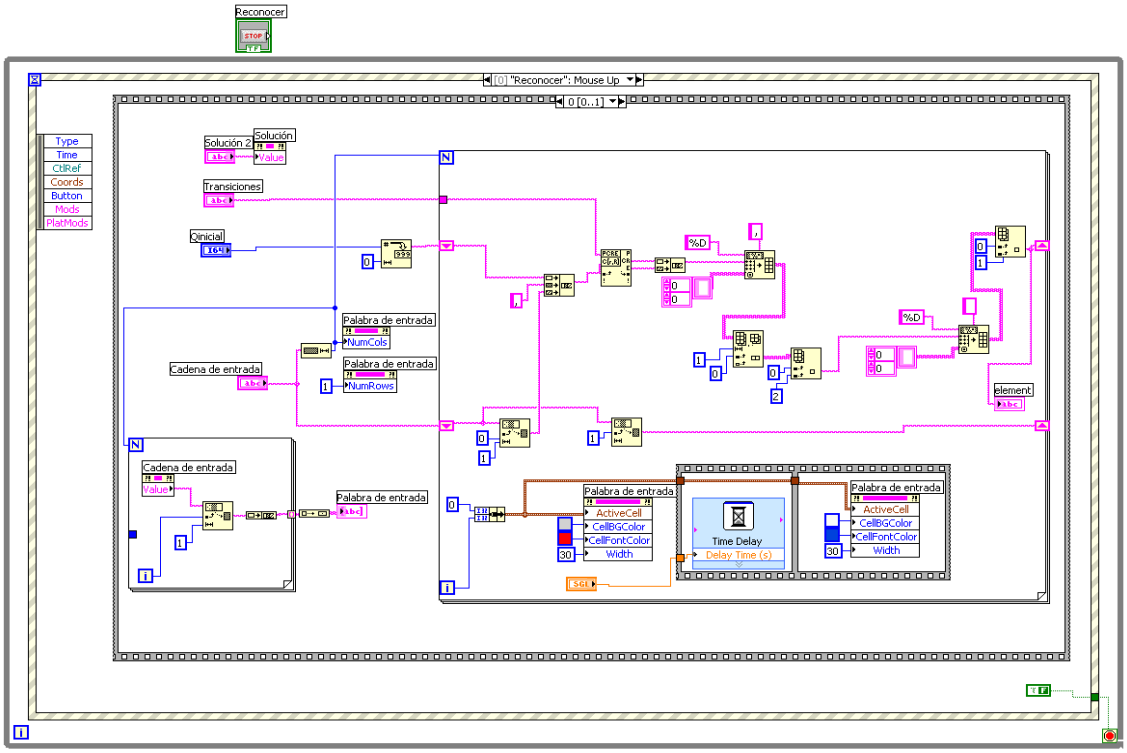


Figura 17. Diagrama de bloques autómata de estados finitos

Las transiciones ya determinadas se deben ingresar en el panel frontal del simulador, Figura 18, como se muestra a continuación:

0,0, 1
0,1, 0
0,2, 0
1,0, 0
1,1, 2
1,2, 0
2,0, 0
2,1, 0
2,2, 3
3,0, 0
3,1, 0
3,2, 4
4,0, 4
4,1, 4
4,2, 4

Estado actual,caracter leído,(espacio)estado siguiente

Figura 18. Panel frontal autómata de estados finitos

También se deben ingresar el estado inicial, el final y la cadena de entrada, cada uno en su respectivo campo, al haber ingresado estos datos se debe presionar el botón RECONOCER para iniciar el proceso de aceptación de la cadena de entrada, el cual termina cuando ésta sea leída en su totalidad por el autómata, ya que este sólo tiene la opción de avanzar hacia la derecha. Si al finalizar dicho proceso el autómata se encuentra en el estado final, la palabra es aceptada con éxito en caso contrario no es aceptada.

5.2 AUTÓMATA DE PILA

El funcionamiento de un autómata de pila también se puede ver como una serie de estados en los cuales se avanza de uno a otro mediante transiciones, pero que a diferencia de los de estados finitos cuenta con un dispositivo de memoria, por lo que las transiciones son diferentes ya que los datos a tener en cuenta no son solamente el estado actual y el carácter leído sino que a estos dos se le suma el caracter del tope de la pila, y en cuanto a la acción respectiva también se le debe agregar la acción a realizar en la pila.

Se puede mostrar su funcionamiento más claramente mediante un ejemplo, el cual es realizar un autómata de pila que cuente $1's$ separados por un signo $(=)$, aceptando las palabras que tengan el mismo número de $1's$ a ambos lados del signo.

Se realiza el diagrama de estados comenzando en el estado inicial y buscando el camino correcto hasta llegar al estado final ó de aceptación, luego se observa qué otras posibilidades hay en el camino (caminos incorrectos) y se llevan a estados que no son de aceptación.

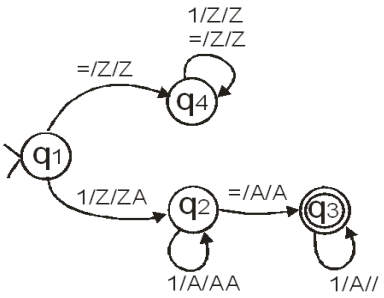


Figura 19. Diagrama de estados autómata de pila

En este diagrama se observa el estado inicial (1), el estado final (3) y se determinan las transiciones, que son las siguientes:

	1	2	3	4
$=, Z$	$4, Z$	--	--	$4, Z$
$=, A$	--	$3, A$	--	--
$1, Z$	$2, ZA$	--	--	$4, Z$
$1, A$	--	$2, AA$	$3, /$	--

Tabla 11. Transiciones del autómata de pila que acepta palabra con el mismo números de $1´s$ a ambos lados del $=$.

Estas transiciones se leen así:

Estando en el estado 1 (estado inicial), leyendo un uno (1) de la cadena de entrada y teniendo Z (símbolo de vacío) en la pila, las acciones respectivas son avanzar al estado 2 y guardar en la pila el caracter A enseguida de Z, si estando en este mismo estado el caracter leído es el signo igual $(=)$, las acciones respectivas es avanzar al estado 4 y no guardar ni borrar nada de la pila, así se leen todas las transiciones. Los espacios donde se encuentre (--) son transiciones que no existen, esto significa que no es posible tener la combinación de estos tres elementos (estado actual, caracter leído y caracter del tope de la pila).

Un caso particular es la acción de borrar un caracter de la pila, la cual se realiza con el símbolo $(/)$, por ejemplo estando en el estado 3, leyendo un uno (1) y teniendo una A en el tope de la pila se permanece en este estado y se borra el dato disponible en la pila.

Con base en esto se construyó el simulador para autómatas de pila en LabVIEW™, cuyo algoritmo,
Figura **21**, se trata de saber en qué estado se encuentra el autómata, que caracter está leyendo de la palabra de entrada y que caracter está en el tope de la pila, con

estos tres datos se realizan las dos acciones respectivas las cuales son avanzar a un nuevo estado ó quedarse en el mismo y la acción en la pila, la cual puede ser guardar, borrar ó dejar la pila como está, estas acciones son declaradas en las transiciones. Este proceso implementado en LabVIEW™ se puede visualizar de una forma más clara en el siguiente diagrama, Figura 20

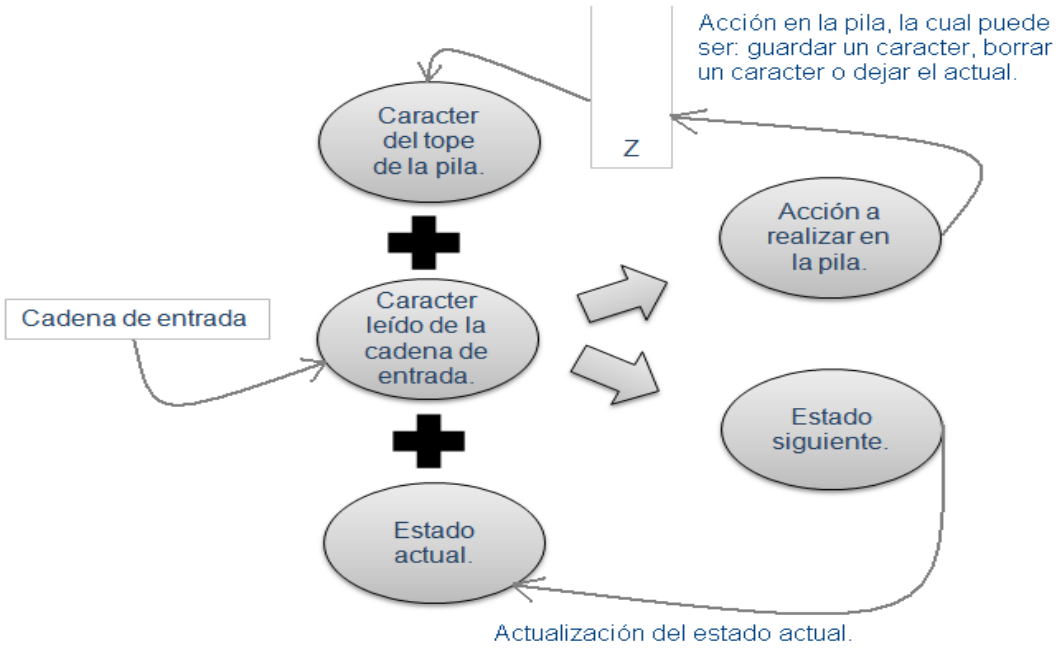


Figura 20. Diagrama de bloques autómata de pila

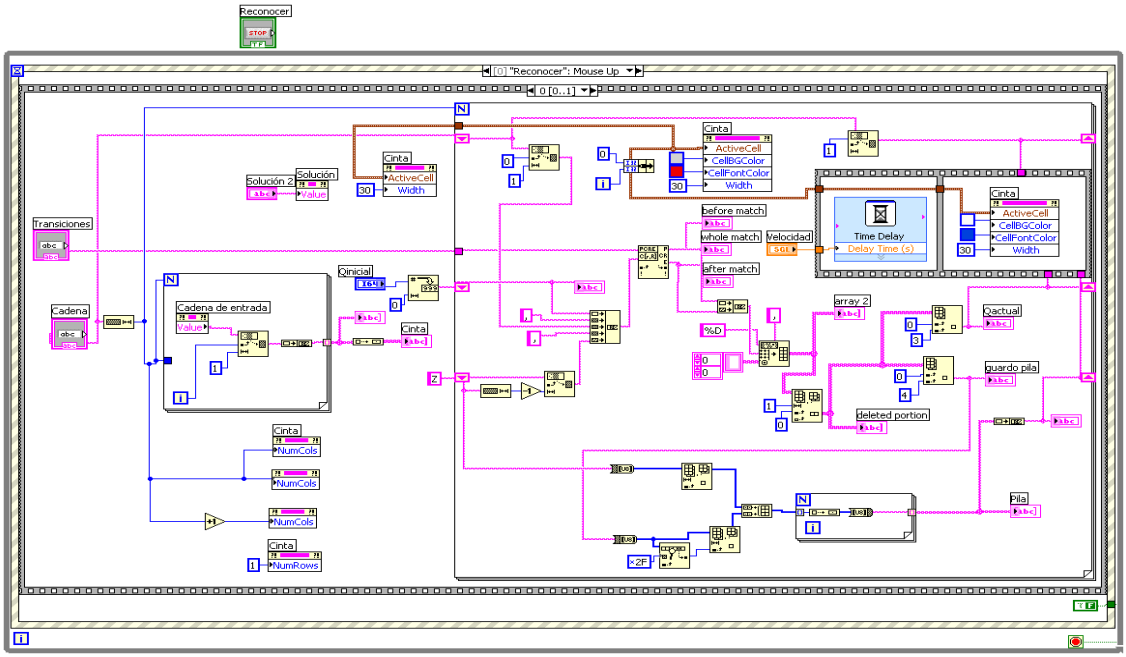


Figura 21. Diagrama de bloques autómata de pila

Las transiciones ya determinadas se deben ingresar en el panel frontal del simulador, Figura 22, como se muestra a continuación:

1,=,Z,4,Z
1,1,Z,2,ZA
2,=,A,3,A
2,1,A,2,AA
3,1,A,3,/
4,=,Z,4,Z
4,1,Z,4,Z

Estado actual,caracter leído,carácter del tope de la pila,estado siguiente,acción a realizar en la pila

Figura 22. Panel frontal autómata de pila

También se deben ingresar el estado inicial, el final y la cadena de entrada, cada uno en su respectivo campo, al haber ingresado estos datos se debe presionar el botón RECONOCER para iniciar el proceso de aceptación de la cadena de entrada, el cual termina cuando ésta sea leída en su totalidad por el autómata, ya que este sólo tiene la opción de avanzar hacia la derecha. Si al finalizar dicho proceso el autómata se encuentra en el estado final y la pila se encuentra con su símbolo de vacío la palabra es aceptada con éxito en caso contrario no es aceptada.

5.3 MÁQUINA DE TURING

La máquina de Turing como los autómatas explicados anteriormente, también se puede ver como una serie de estados en los cuales se avanza de uno a otro mediante unas transiciones, pero que a diferencia de dichos autómatas tiene la capacidad de mover su cabeza de lectura/escritura atreves de la cinta no sólo hacia la derecha sino también hacia la izquierda ó quedarse en la misma posición.

Esta máquina consta, gracias a su capacidad de ser bidireccional, con la posibilidad de modificar el contenido de la cinta durante su recorrido sobre ésta. Por lo cual esta máquina no es posible desarrollarla a partir de un diagrama de estados, ya que éste no nos brinda la información necesaria para construir las transiciones, las cuales constan de dos datos indispensables para la ejecución de las acciones respectivas, estos datos son el estado actual y el carácter leído de la cinta en su posición actual, los cuales llevan a ejecutar las acciones respectivas: caracter a escribir en la posición actual, dirección del próximo movimiento de la cabeza de lectura/escritura y el estado siguiente.

Es de mucha importancia aclarar la forma de ingresar la cadena de entrada en particular en esta máquina, ya que como se sabe es bidireccional y la cinta es infinita, y también la forma de paro ya que dicha máquina puede modificar el contenido de la cinta y podría quedarse en ella infinitamente, por eso se explica muy bien estos pasos y el funcionamiento de la Máquina de Turing mediante un ejemplo sencillo, el cual es realizar una Máquina de Turing que sume $1's$.

Se realiza el diagrama de estados, aunque como se mencionó anteriormente no es posible determinar las transiciones a partir de este, ya que con dicho diagrama sólo se controla el movimiento de la cabeza de lectura/escritura dependiendo del caracter leído y el estado actual.

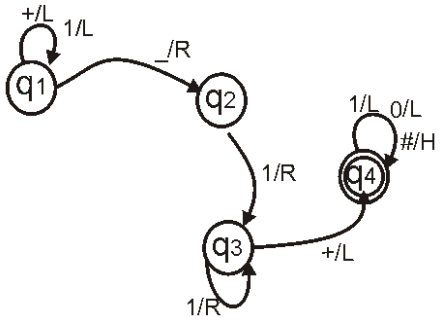


Figura 23. Diagrama de estados que contrala la cabeza de lectura/escritura de la máquina de Turing que suma $1's$.

En el diagrama de estados, Figura 23, se puede observar el estado inicial 1, el final 4 que para esta máquina no es necesariamente el estado de aceptación.

Las transiciones se realizan según el objetivo a conseguir para cada programa simulado, siguiendo las respectivas instrucciones. Las transiciones para este ejemplo son:

	1	2	3	4
+	$+, L, 1$	--	$1, L, 4$	--
1	$1, L, 1$	$0, R, 3$	$1, R, 3$	$1, L, 4$
_	$\#, R, 2$	--	--	--
0	--	--	--	$=, L, 4$
#	--	--	--	$\_, H, 4$

Tabla 10. Transiciones de la máquina de Turing que suma $1´s$

Estas transiciones se leen así:

Estando en el estado 1 (estado inicial) y leyendo un uno (1) de la cinta, las acciones respectivas son escribir un 1 en la posición actual, su próximo movimiento es hacia la izquierda (L) y permanecer en el estado 1, si estando en este mismo estado el caracter leído es el signo $(\_)$, las acciones respectivas son escribir el signo $\#$, su próximo movimiento es hacia la derecha (R) y avanzar al estado 2, así se leen todas las transiciones. Los espacios donde se encuentre (--) son transiciones que no existen, esto significa que no es posible tener la combinación de estos dos elementos (estado actual y caracter leído).

Con base en esto se construyó el simulador para Máquina de Turing en LabVIEW™, cuyo algoritmo, Figura 25, se trata de saber en qué estado se encuentra la máquina y que caracter está leyendo de la cinta, con estos dos datos se realizan las tres acciones respectivas las cuales son escribir el nuevo caracter en la posición actual, desplazar la cabeza de lectura/escritura sobre la cinta y avanzar a un nuevo estado ó quedarse en el mismo, estas acciones son declaradas en las transiciones, vale aclarar que primero escribe el nuevo caracter y luego hace el movimiento indicado. Este proceso implementado en LabVIEW™ se puede visualizar de una forma más clara en el diagrama de la Figura 24.

Como se mencionó anteriormente es muy importante el correcto ingreso de la cadena de entrada, la cual debe estar precedida por un $(\_)$ pues la cinta es infinita y este signo permite conocer la posición más a la izquierda de la palabra de entrada, la cual por definición comienza en la segunda celda de la cinta. También es necesario finalizar la cadena con otro $(\_)$ para identificar la posición más a la derecha de la palabra de entrada sobre la cinta, e indicar que las siguientes celdas de la cinta infinita se encuentran vacías.

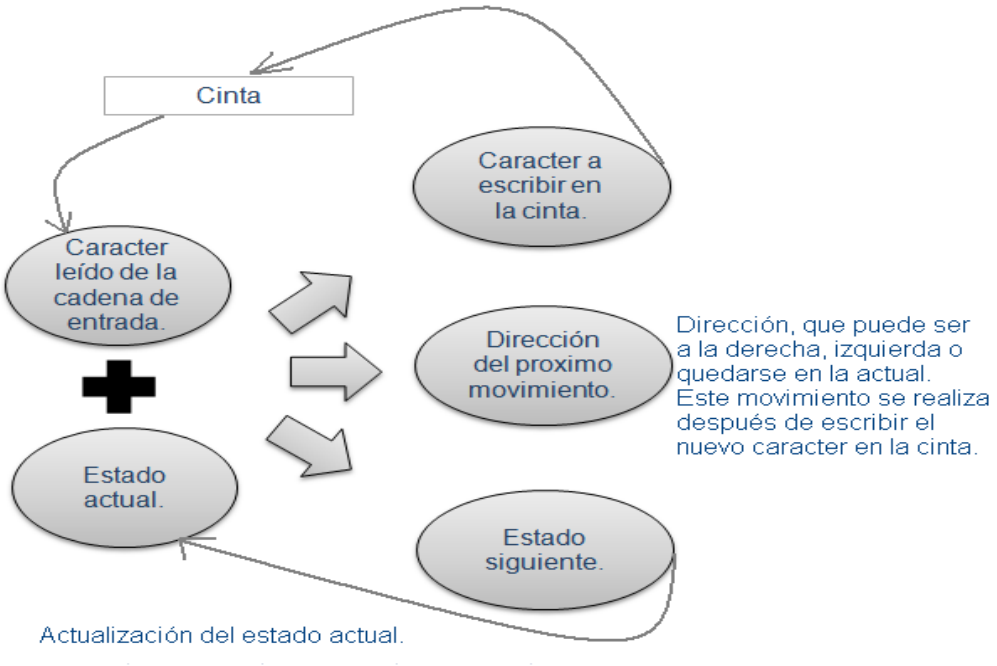


Figura 24. Diagrama de bloques máquina de Turing

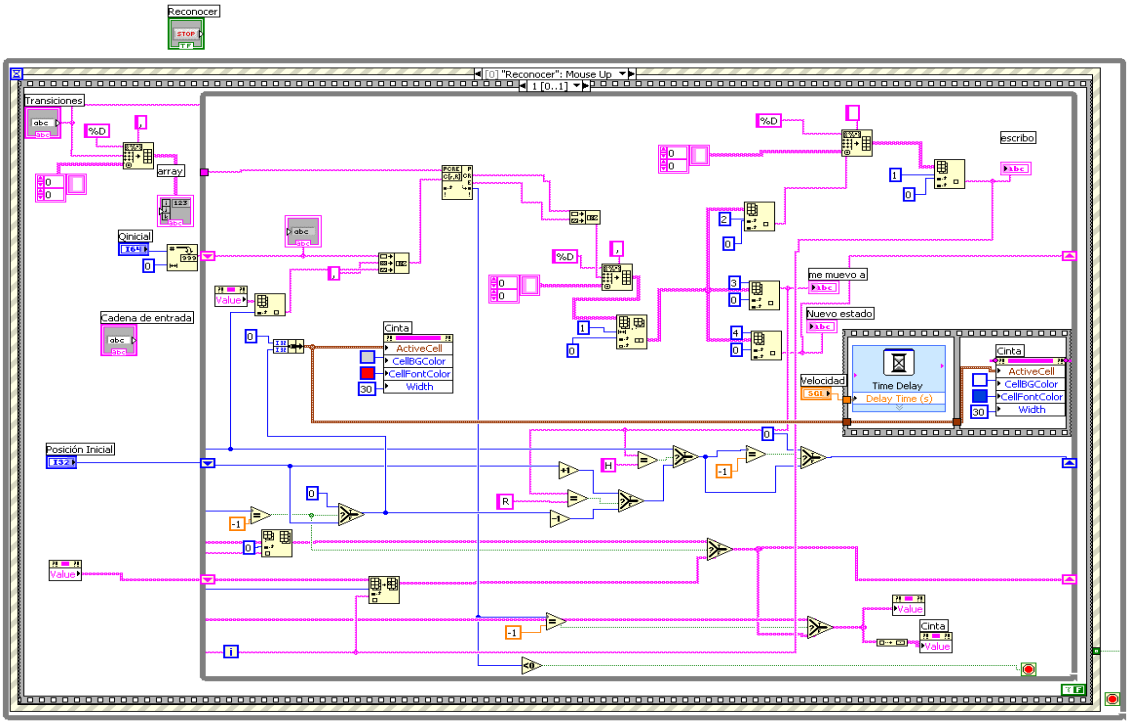


Figura 25. Diagrama de bloques para la máquina de Turing que suma $1's$.

Las transiciones ya determinadas se deben ingresar en el panel frontal del simulador, Figura 26, como se muestra a continuación:

1,+, +,L,1
1,1, 1,L,1
1,_, #,R,2
2,1, 0,R,3
3,+, 1,L,4
3,1, 1,R,3
4,1, 1,L,4
4,0, =,L,4
4,#, _,H,4

Estado actual,caracter leído,(espacio)caracter a escribir,dirección del próximo movimiento de la cabeza de lectura/escritura,estado siguiente

Figura 26. Panel frontal de la máquina de Turing que suma $1's$.

También se deben ingresar el estado inicial, la posición inicial de la cabeza de lectura/escritura sobre la palabra de entrada y la cadena de entrada, cada uno en su respectivo campo, al haber ingresado estos datos se debe presionar el botón RECONOCER para iniciar el proceso de aceptación de la cadena de entrada, el cual termina cuando la máquina llega a un *halt*, para el simulador desarrollado dicho *halt* se alcanza cuando no se encuentre una transición, esto significa que los dos elementos necesarios no coinciden en ninguna de ellas (estado actual y carácter leído).

6. ANÁLISIS DE RESULTADOS

Con este proyecto se enmarcó el poderío matemático y generalidad de la Máquina de Turing entre las máquinas abstractas equivalentes a la jerarquía de lenguajes formales que desarrolló Noam Chomsky en su obra Teoría de las Gramáticas Transformacionales, por medio del desarrollo de un simulador de autómatas; que permite representar el funcionamiento de un reconocedor de lenguajes que determina si una palabra, pertenece o no a un leguaje dado; como una herramienta pedagógica que permite mostrar como la máquina de Turing abarca el conjunto de todas las máquinas abstractas.

6.1 CAPACIDADES Y LIMITACIONES DE AUTÓMATAS

A continuación se muestran las capacidades y limitaciones de cada máquina por medio de un ejemplo, que confronta la teoría en cuestión y permite resaltar la generalidad y poderío de la Máquina de Turing, y muestra el funcionamiento validez y confiabilidad del entorno de simulación desarrollado.

6.1.1 Máquina de estados finitos

$\{a^2b^2\}$ es un tipo de lenguaje formal, pues sus palabras contienen regularidades o repeticiones de los mismos componentes, como se mencionó en la sección 1.2 los lenguajes regulares pueden ser reconocidos por un autómata finito y son los más simples y restringidos dentro de la jerarquía de Chomsky.

El funcionamiento de los autómatas finitos consiste en ir pasando de un estado a otro, a medida que va recibiendo caracteres de la palabra de entrada, y en un conjunto de transiciones, δ, entre esos estados, que dependen de los símbolos de la cadena de entrada, como se muestra en el diagrama de estados en la Figura 27, donde el camino de aceptación es el que toma la secuencia de estados $q_0q_1q_3q_4q_5$, otro camino diferente a este es de no aceptación a la palabra de entrada, en la Figura 28 el autómata sigue el diagrama de estados pues al leer el primer caracter toma el estado 1 y al leer el segundo el estado 3, así hasta leer toda la cadena de entrada.

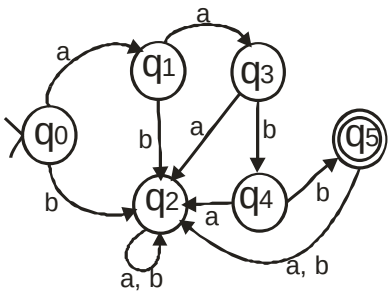


Figura 27. Diagrama de estado del automata reconocedor del lenguaje a^2b^2

Figura 28. Funcionamiento del autómata a^2b^2

Figura 29. Aceptación del la palabra $aabb$.

El autómata finito determinista acepta la cadena $aabb$, Figura 29, porque se consumen todos los caracteres de dicha palabra de entrada y la secuencia de transiciones correspondientes a los símbolos de $aabb$ conduce desde el estado inicial a el estado final.

La trayectoria seguida para la palabra $aabb$, que puede visualizarse en el diagramada de estados, Figura 27, consiste en la secuencia de estados: q_1, q_3, q_4; q_5. Los estados son el único medio que disponen los autómatas finitos para recordar qué caracteres se han leído hasta el momento, son máquinas de memoria limitada; esta limitación los hace finalmente incapaces de distinguir las palabras aceptables de las no aceptables en ciertos lenguajes, más complicados que los lenguajes regulares.

Para el lenguaje $\{a^n b^n\}$ no es posible construir un autómata finito que lo acepte, ni representarlo por una expresión regular o gramática regular, pues al terminar de leer el grupo de $a's$ el autómata debe recordar cuántas encontró, para poder comparar con el número de $b's$. Como la cantidad de $a's$ que puede haber en la primera mitad de la palabra no es constante, dicha cantidad no puede recordarse con una memoria fija, como es la de los autómatas finitos.

6.1.2 Máquina de pila

$\{a^n b^n\}$ es un lenguaje libre de contexto, el cual forma una clase de lenguaje más amplia que los lenguajes regulares, de acuerdo con la Jerarquía de Chomsky.

Como los autómatas finitos no son suficientemente poderosos para aceptar los lenguajes libres de contexto se les agregó una memoria en forma de pila que incrementa su poder de cálculo. La pila tiene un alfabeto propio, que puede o no coincidir con el alfabeto de la palabra de entrada.

El movimiento del autómata de pila consiste en un cambio de estado, en la lectura del símbolo de entrada y en la sustitución del símbolo del tope de la pila por una cadena de símbolos, como se observa en el diagrama de estados de la Figura 30.

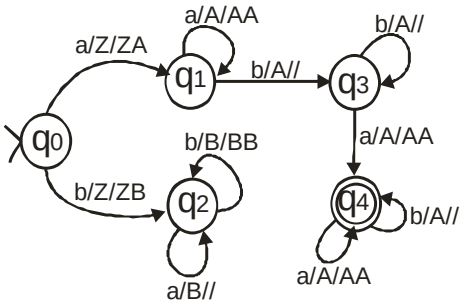


Figura 30. Diagrama de estado del automata reconocedor del lenguaje $\{a^n b^n\}$

Como se muestra en la Figura 31 al iniciar la operación la pila se encuentra vacía, Z carácter que indica que la pila está vacía, durante la operación la pila almacena o borra caracteres, según lo indiquen las transiciones ejecutadas, la pila funciona de manera que el último caracter que se almacena en ella es el primero en salir; los caracteres a la mitad de la pila no son accesibles sin quitar antes los que están encima de ellos, como se muestra en el diagrama de estados de la Figura 30; donde en el estado q_1 al leer una a de la cinta y con la pila vacía, toma el estado q_2 y guarda una A en la pila, y así toma los siguientes estados hasta leer todos los caracteres de la cadena de entrada, en la Figura 32 se muestra como el autómata sigue el diagrama de estados pues en el estado 4, lee la segunda b de la cadena de entrada y con AA en la pila, descompila una A y conserva el estado 4.

Figura 31. Pila vacia.

Figura 32. Pila borrando caracteres.

Para diseñar el autómata de pila que acepta exactamente el lenguaje con palabras de la forma $a^n b^n$, para cualquier número natural n, se utilizó la pila como contador para recordar la cantidad de $a's$ que se consumen, y luego confrontar con la cantidad de $b's$, y los estados para memorizar las situaciones de estar consumiendo a o estar consumiendo b.

Al final de la operación de aceptación de la palabra, la pila está vacía, se han leído todos los caracteres de la palabra de entrada y la secuencia de transiciones conduce desde el estado inicial al estado final, como se muestra para la palabra $aaabbb$ en la Figura 33.

Figura 33. Aceptación de la palabra $aaabbb$.

Las GLC deben su nombre a una comparación con las gramáticas, sensitivas al contexto, donde una regla genera un símbolo cuando se encuentra rodeado por un contexto.

Las gramáticas sensitivas al contexto son estrictamente más poderosas que las gramáticas libres de contexto; un ejemplo es el lenguaje de las cadenas de la forma $a^n b^n c^n$, para el que no existe una gramática libre de contexto

6.1.3 Máquina de Turing

Para probar que hay lenguajes que no son libres de contexto, pero que pueden ser aceptados por una máquina de Turing, se trabaja con el lenguaje $a^n b^n c^n$, que se sabe que no es un lenguaje libre de contexto.

Al diseñar la MT que acepta un el lenguaje $a^n b^n c^n$, se diseñó el autómata finito con salida que controla la cabeza y la cinta, como se muestra en la Figura 34 donde la transición que va del estado 1 a el estado 2 con nombre $\_/R$, quiere decir que cuando el caracter leído por la cabeza de la MT, es $\_$, la cabeza lectora hace un movimiento a la derecha, indicada por el caracter R.

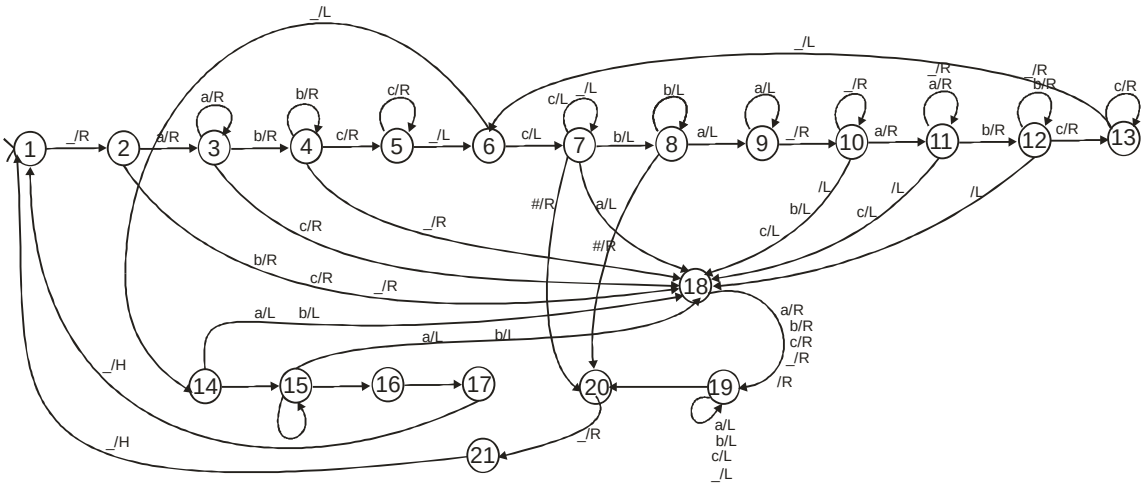


Figura 34. Diagrama de estado de la MT reconocedora del lenguaje $\{a^n b^n\}$

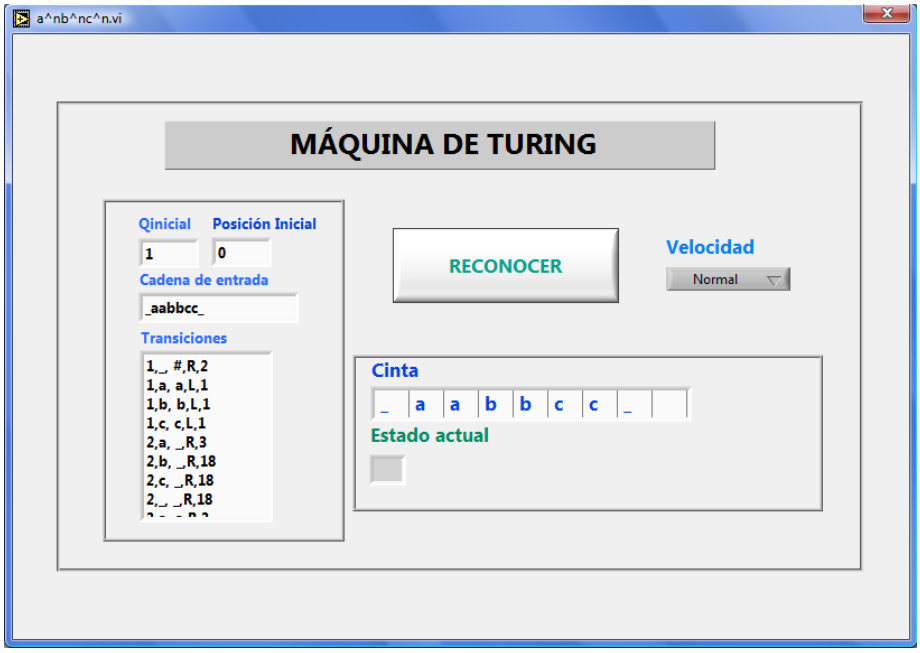


Figura 35. Ubicación de la palabra de entrada *aabbcc* en la cinta.

La palabra de entrada, *aabbcc*, está escrita inicialmente en la cinta. Como la cinta es infinita, inicialmente toda la parte de la cinta a la derecha de la palabra de entrada está llena del caracter blanco, (_) ,Figura 35.

Por definición, al iniciar la operación de la MT, la cabeza lectora está posicionada en el caracter blanco, (_) , a la izquierda de la palabra de entrada, el cual es el cuadro más a la izquierda de la cinta, Figura 35.

La operación de la MT consta de los siguientes pasos: leer un caracter en la cinta, efectuar una transición de estado y realizar una acción en la cinta.

La estrategia para el funcionamiento de dicha MT consiste en recorrer la palabra, descontando en cada una de ellas una a, una b y una c; para descontar esos caracteres simplemente se reemplazan por un caracter $\_$. En la MT la cabeza lectora es de lectura y escritura, por lo que la cinta es modificada en curso de ejecución. Además, en la MT la cabeza se mueve bidireccionalmente, izquierda y derecha, por lo que pasa repetidas veces sobre un mismo segmento de la cinta.

Cuando ya no encuentra ninguna a, b o c en algún recorrido, si queda alguna de las otras dos letras la palabra no es aceptada; en caso contrario se llega a *halt,* detiene la operación de la MT, y se acepta la cadena de entrada como se muestra en la Figura 36 para la palabra $aabbcc$, donde se observa que el único criterio para que la palabra de entrada sea aceptada es que se llegue a *halt* en algún momento, independientemente del estado en que se encuentre y el contenido final de la cinta, el cual es visto como basura.

Figura 36. Aceptación de la cadena de entrada $aabbcc$.

6.2 CRITERIOS DE VALIDEZ Y CONFIABILIDAD

Para verificar la validez y confiablidad de los resultados se implementó, en el entorno de simulación desarrollado, ejemplos de aplicación de otros simuladores existentes, que permitieron determinar de forma clara la aceptación o no de un lenguaje según la programación para dicho ejemplo. También se compararon los

resultados entregados por el simulador desarrollado con ejemplos encontrados en algunas referencias.

Dichos ejemplos se tomaron para cada uno de los tipos de autómatas determinísticos de interés: Autómatas Finitos, Autómatas Finitos, Máquina de Pila y Máquina de Turing

6.2.1 Autómatas finitos

La referencia [26] es una utilidad escrita en Java, desarrollada para un ámbito docente. Soporta tanto autómatas finitos deterministas como no deterministas, y su finalidad es la de trabajar como una máquina reconocedora de las palabras del lenguaje que se define. Decide si una palabra es reconocida o no, o si es imposible que pertenezca al lenguaje generado por la gramática del autómata.

El ejemplo de aplicación existente en dicha referencia y ejecutado en el entorno de simulación desarrollado es la aceptación de palabras que contienen la cadena 11010101 . Como se muestra en la Figura 37 la palabra de entrada 001101010111 es aceptada por contener la cadena 11010101, y la palabra 01100101001 no es aceptada pues no contiene la cadena requerida, Figura 38.

Figura 37. Aceptación de la cadena de entrada 001101010111.

Figura 38. Rechazo de la cadena de entrada 01100101001

Comparando los resultados entregados por el simulador cuando ejecuta el programa que acepta cadenas que contienen $a's$ y/o $b's$ terminadas en a con la referencia [20], que desarrolla este autómata, se verifica la validez del autómata de estados finitos; como se muestra en la Figura 39 el autómata acepta la cadena de entrada *aaababbba* que cumple las condiciones de dicho programa. Este ejemplo también se simulo en el autómata de la referencia [26] obteniendo el mismo resultado.

Figura 39. Aceptación de la cadena de entrada *aaababbba*

6.2.2 Autómatas de pila

La referencia [27] es una herramienta reconocedora de lenguajes escrita en Java donde se define una tabla de transiciones y una palabra de entrada, y se muestran los estados por los que transcurre la simulación y los resultados en un campo de texto, es decir, si la palabra es aceptada o no.

El ejemplo de aplicación existente en dicha herramienta de simulación y en el texto escrito por James A. Anderson, referencia [1], ejecutado en el entorno de simulación desarrollado, es el autómata reconocedor de palabras que contienen la misma cantidad de $a's$ y $b's$, muestra el correcto funcionamiento de la máquina de pila. Como se observa en la Figura 40 el autómata acepta la palabra de entrada $aaababbb$, pues contiene el mismo número de $a's$ y $b's$, y para la palabra $aabbabb$ la respuesta del autómata es de no aceptación, Figura 41.

Figura 40. Aceptación de la cadena de entrada $aaababbb$

Figura 41. Rechazo de la cadena de entrada *aabbabb*

6.2.3 Máquina de Turing

La referencia [28] es una interfaz en Java que sólo, permite simular máquinas de Turing definiendo un programa, conjunto de transiciones, o cargando una de las que ya hay definidas en el servidor. Se pueden hacer simulaciones paso a paso, o una ejecución en tiempo real, con distintas velocidades. La cinta muestra los cambios que van sucediendo y al final muestra un mensaje de solución de aceptación o no de la palabra de entrada.

Cargando el conjunto de transiciones del programa que acepta palabras palíndromas, existente en esta referencia, se muestra el correcto funcionamiento de la Máquina de Turing desarrollada, en la Figura 42 la palabra *baabbaab* es aceptada por ser una palabra palíndroma, caso contrario el mostrado en la Figura 43 *baabbaa* donde la palabra no es aceptada.

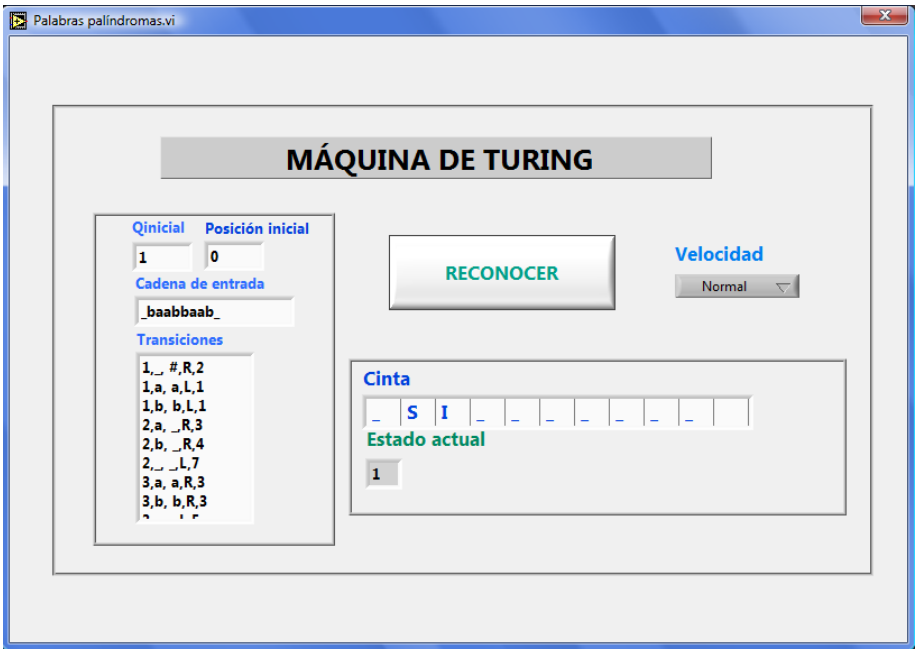


Figura 42. Aceptación de la cadena de entrada $baabbaab$.

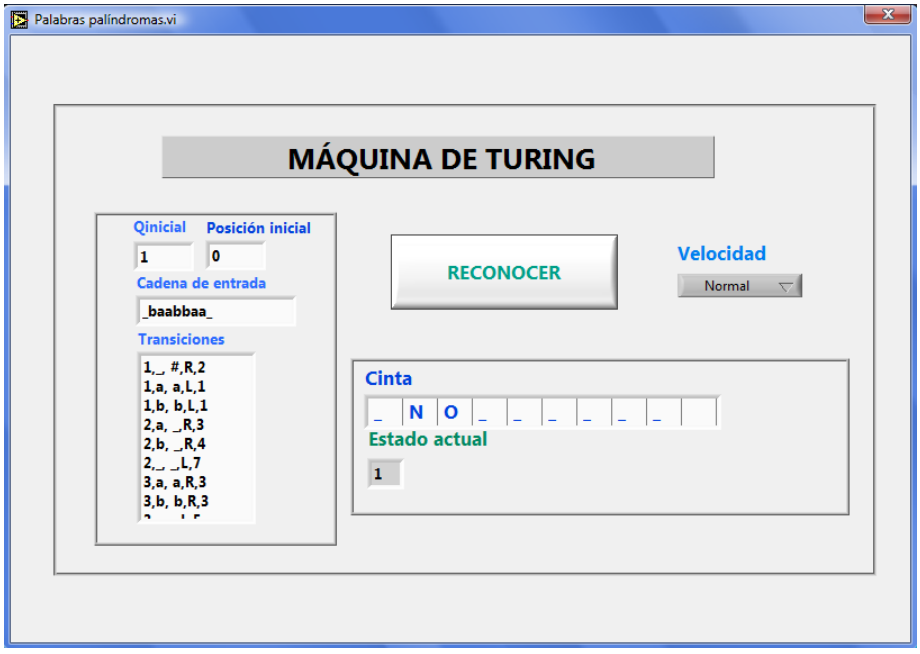


Figura 43. Rechazo de la cadena de entrada $baabbaa$.

Verificando los resultados entregados por el simulador desarrollado con el ejemplo copia $a^n b^n c^n$ encontrado en la referencia [8], se muestra la confiabilidad de la Máquina de Turing como se observa en la Figura 44 donde la cadena de entrada $aaabbbccc$ es aceptada, y la cadena $aabbbcc$ no es aceptada como se muestra en la Figura 45.

Figura 44. Aceptación de la cadena de entrada *aaabbbccc* .

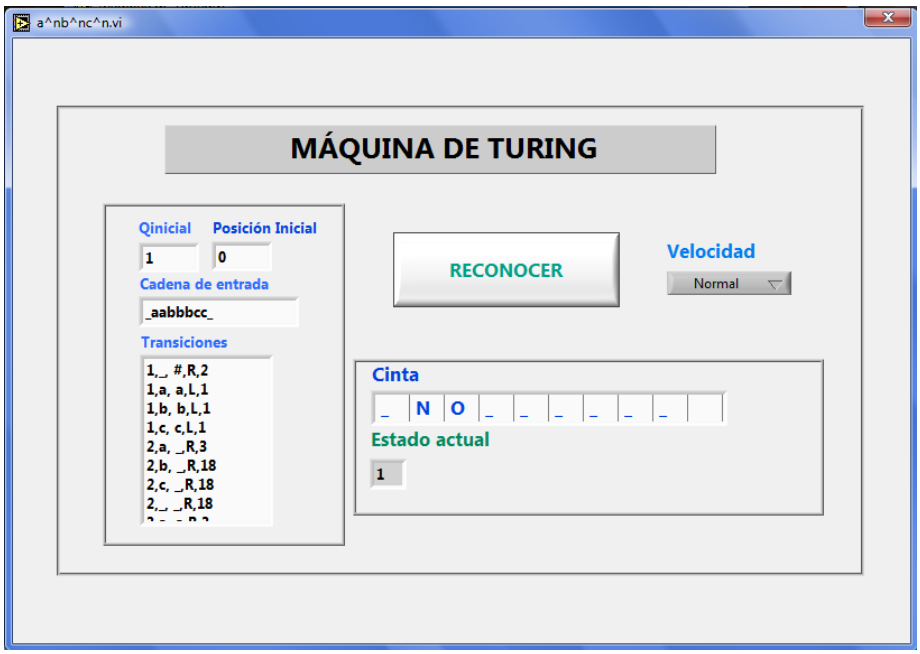


Figura 45. Rechazo de la cadena de entrada *aabbbcc* .

7. CONCLUSIONES

- Los autómatas han ido materializándose paulatinamente hasta convertirse en herramientas cada vez más participativas; aportando, con validez en la realización de cálculos. En esta evolución de la materialización de los procedimientos se ha llegado a la construcción de máquinas con capacidad para recibir información y ajustar su comportamiento de acuerdo con la información recibida, máquinas en las que cada vez intervienen menos, de forma directa, las personas; siendo de gran importancia dar a conocer de forma pedagógica dichas maquinas: Autómatas Finitos, Máquina de Pila y Máquina de Turing; con sus respectivos principios, utilidades, limitaciones y jerarquías.

- Todos los sistemas modernos de cómputo, pueden ser modelados utilizando una abstracción matemática, la Máquina de Turing, la cual permite describir, mediante un conjunto simple de operaciones, todas las funciones computables.

- En el curso de relevación se estudian los autómatas de estados finitos con salida dejando de lado máquinas superiores a estas, como son los autómatas de pila y la máquina de Turing; es importante introducir al estudiante dichas máquinas pues los nuevos desafíos relacionados con los autómatas tratan cada vez con sistemas más difíciles de simular, implementar y validar, por lo que se hace necesario emplear técnicas de mayor generalidad y poder, que permitan una posterior implementación en los sistemas tradicionales o actuales.

- Los procesos computacionales están limitados por el lenguaje en el cual puedan expresarse La Teoría de la Computabilidad pretende abstraer los detalles de los sistemas computacionales y buscar un algoritmo para efectuar un cálculo sin preocuparse por los detalles de implantación. Puede verificarse si una función es computable utilizando una máquina de Turing como un modelo de máquina isomorfa a cualquier otro sistema computacional.

- El interés de Turing acerca de la capacidad de las máquinas para pensar lo llevó a desempeñar un papel importante en el desarrollo de las computadoras, no solo teóricas sino reales. Reforzando la tesis de Church-Turing que dice: si una máquina de Turing no puede resolver un problema, entonces ningún computador puede hacerlo, ya que no existe algoritmo para obtener una solución. Por tanto, las limitaciones corresponden a procesos computacionales y no a la tecnología.

- El entorno de simulación de autómatas deterministas desarrollado permite representar el funcionamiento de un reconocedor de lenguajes, determinando si una palabra, pertenece o no a un leguaje dado, a partir de la definición de alfabeto, estados y función de transición; como una herramienta pedagógica que enmarca el poderío matemático y generalidad de la Máquina de Turing entre las

máquinas abstractas equivalentes a la jerarquía de lenguajes formales que desarrolló Noam Chomsky.

- Después del trabajo realizado sobre las Máquinas de Turing se puede concluir que el tema en cuestión es bastante amplio por lo cual queda por analizar y estudiar más a fondo temas como las extensiones a la Máquina de Turing tales como las no deterministas, las multi-cintas entre otras y el hecho de que puede probarse matemáticamente que para cualquier programa de computador es posible crear una máquina de Turing equivalente.

8. RESULTADOS

- Entorno de simulación para autómatas deterministas: autómata de estados finitos, máquina de pila y máquina de Turing

- Manual del usuario para el manejo del software entorno de simulación para autómatas deterministas

- Se comprobó la confiabilidad del software ejecutando en él ejemplos existentes en otros simuladores, obteniendo resultados positivos

- Se comparó con ejercicios encontrados en las diferentes referencias citadas, obteniendo una validez del ciento por ciento en el entorno de simulación desarrollado

9. BIBLIOGRAFÍA

[1] ANDERSON, James, A. Automata Theory with Modern Applications, Cambridge University Press, United States, 2006.

[2] AVENDAÑO, Luis Enrique. Reconocimiento de Patrones. Universidad Tecnológica de Pereira, 1990.

[3] BACKUS J.W. *The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference*, Proc. of the Intl Conf on Information Processing, IFIP'59, pp. 125–131, 1959

[4] CASSANDRAS, Christos G. y LAFORTUNE ,Stéphane. Introduction to Discrete Event Systems, Springer, 2008

[5] CASES Muñoz, Rafael. Lenguajes, gramáticas y autómatas. Universidad Politécnica de Cataluña, Barcelona, 2002.

[6] CHOMSKY N. *Aspects of The Theory Of Syntax*, Cambridge, Mit Press, 1965.

[7] CHOMSKY N. *On certain formal properties of grammars. Information and Control*, 2(2):137–167, 1959.

[8] HOPCROFT, John E. *Introduction to Automata Theory, Languages, and Computation*, Pearson Education S.A., New York, 2000.

[9] HROMKOVIC, Juraj. *Theoretical Computer Science, Springer*, New York, 2004.

[10] HUFFMAN. D.A. *The synthesis of sequential switching circuits. Journal of the Franklin Institute*, 257:161–190, 1954.

[11] LEWIS, Ch. H. PAPADIMITRIOU. *Elements of the Theory of Computation*, Prentice Hall, 1981.

[12] LINZ, P. An Introduction to Formal Languages and Automata, D. C. Heath and Company, 1990.

[13] MEALY, G. *A method for synthesizing sequential circuits*, BSTJ n.34, 1955, pp1045-1079.

[14] MINSKY M. Computation: Finite and Infinite Machines, Prentice Hall, 1967.

[15] MOORE, E. Gedanken. *Experiments on Sequential Machines, en "Automata Studies,* (C. Shannon, J. McCarthy eds.), Princeton Univ. Press, 1956.

[16] ROCHA, Jairo. Verificación de Autómatas y Gramáticas, Departamento de Matemáticas, Universidad de las Islas Baleares, 2003.

[17] ROCHA, Jairo. Autómatas de Pila y Máquinas de Turing Estructurados, Departamento Ciencias Matemáticas e Informática, Universidad de las Islas Baleares, 2005.

[18] SCOTT y RABIN, Michael O, *Finite Automata and Their Decisión Problem*., 1959

[19] SIPSER, M. *Introduction to the Theory of Computation*, PWS Pub. Co, 1997.

[20] STRAUBING, Howard, Finite Automata, Formal Logic, and Circuit Complexity, Birkhäuser, Berlin, 1994.

[21] TAKAHASHI Silvia, Notas de curso Teoría de Lenguajes, Universidad de los Andes, Colombia, 2006.

[22] TURING, A. *On computable numbers with an application to the Entscheidungs-problem*, Proc. London Math.

[23] WEYUKER, Elaine, SIGAL, Ron y DAVIS, Martin. Computability, Complexity, and Languages, Morgan Kaufman, 1994.

[24] Maquinas de Turing, REVISTA UNIVERSIDAD EAFIT, No 103, 1996.

[25] Maquina Universal de Turing: Algunas Indicaciones para su Construcción, REVISTA UNIVERSIDAD EAFIT, No 108, 1997.

[26] Simulador autómata de estados finitos
http://www.ucse.edu.ar/fma/sepa/chalchalero.htm

[27] Simulador Maquina de Pila
http://www.cs.iitm.ernet.in/tell/automata/Automata/pda/app.html

[28] Simulador Máquina de Turing http://ironphoenix.org/tril/tm/

10. ANEXO: GUÍA PARA EL MANEJO DEL ENTORNO DE SIMULACIÓN DE AUTÓMATAS DETERMINISTAS

Durante la elaboración de este trabajo se desarrolló un entorno para la simulación de Autómatas Finitos, Autómatas de Pila y Máquinas de Turing, proporcionando útiles herramientas que permiten la aplicación de conceptos y mejorarán la experiencia en el campo de los autómatas y lenguajes regulares.

Las funciones de edición visual del entorno de simulación permiten crear autómatas rápidamente y utilizar sus algoritmos pudiendo ver todos los pasos seguidos.

10.1 VENTANA PRINCIPAL

Al abrir el simulador se podrá visualizar la siguiente ventana:

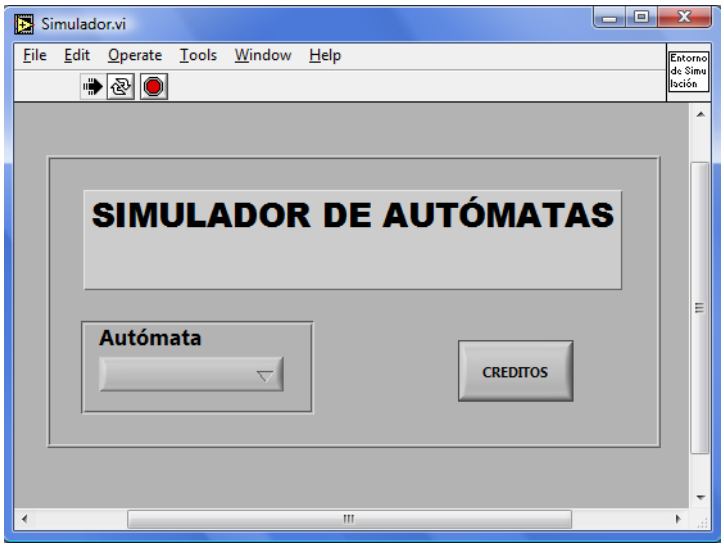


Figura 46. Ventana principal del simulador de autómatas.

En la ventana principal se puede distinguir el menú para la elección de los diferentes autómatas, Figura 47:

- Estados finitos: Abre la ventana para empezar a trabajar con un autómata de estados finitos.
- Autómata de Pila: Abre la ventana para empezar a trabajar con una Máquina de Pila
- Turing: Abre la ventana para empezar a trabajar con una máquina de Turing.

Figura 47. Menú ventana principal del simulador de autómatas.

10.2 ESTADOS FINITOS

Al seleccionar esta opción se abre la ventana de la Figura 48, la cual permite crear y guardar nuevos programas para este autómata, contando con una ayuda para cada uno de los diferentes campos, que se obtiene presionando ctrl+h, Figura 49.

Figura 48. Autómata de Estados Finitos

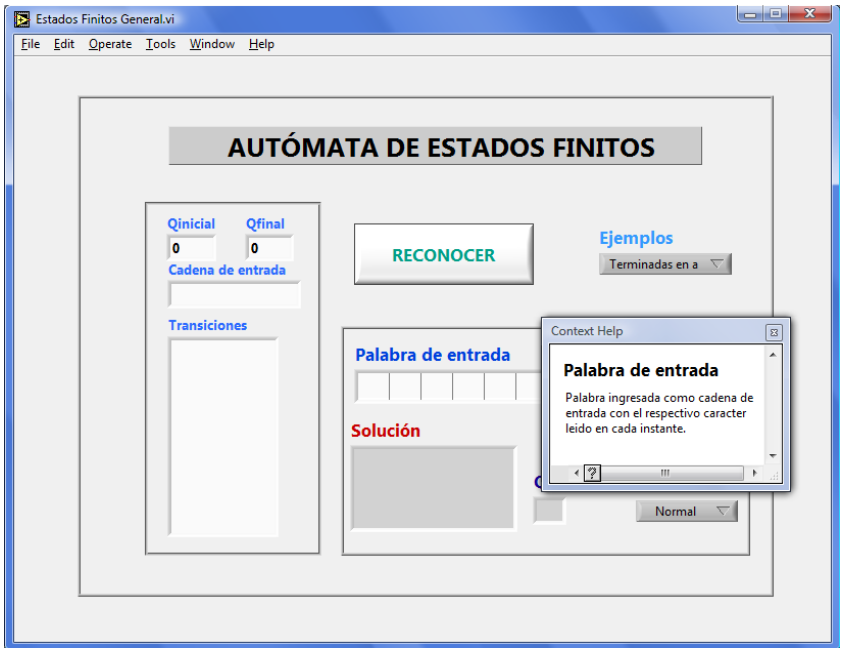


Figura 49. Ayuda autómata de Estados Finitos

Se pueden desplegar algunos ejemplos, Figura 50, que permiten observar la forma de ingresar adecuadamente los datos y el funcionamiento de este autómata.

Figura 50. Ejemplos autómata de Estados Finitos

Un ejemplo es a^2b^2, Figura 51, donde se observan los diferentes campos requeridos para el correcto funcionamiento del autómata, estos son:

- Qinicial: Estado inicial del autómata.
- Qfinal: Estado final del autómata.
- Cadena de entrada: Palabra a ser reconocida por el autómata.
- Transiciones: Programa o transiciones que ejecuta el autómata. Estas transiciones deben ser ingresadas de la siguiente forma:

  Estado actual,caracter leído de la cadena de entrada,(espacio)estado siguiente

Figura 51. Ejemplo a^2b^2

Presionando el botón RECONOCER se observa, en el campo Qactual, los diferentes estados tomados por el autómata durante el proceso de reconocimiento de la cadena de entrada, finalizado dicho proceso se muestra el mensaje de aceptación en el campo SOLUCIÓN, Figura 52.

Figura 52.Respuesta de aceptación a la cadena de entrada

10.3 AUTÓMATA DE PILA

Al seleccionar esta opción se abre la ventana de la Figura 53, la cual permite crear y guardar nuevos programas para este autómata, contando con una ayuda para cada uno de los diferentes campos, que se obtiene presionando ctrl+h. También se pueden desplegar algunos ejemplos, que permiten observar la forma de ingresar adecuadamente los datos y el funcionamiento de este autómata.

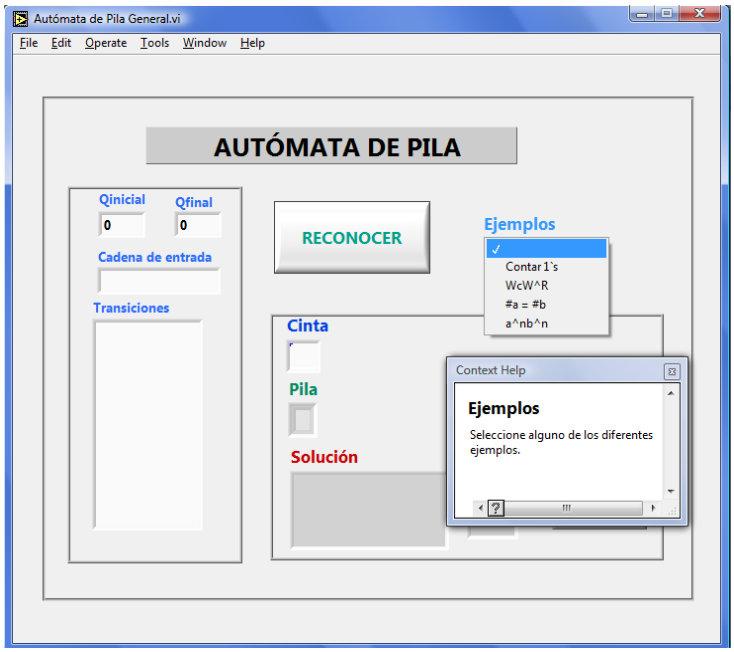


Figura 53. Ventana principal Autómata de Pila

Un ejemplo es $a^n b^n$, Figura 54 donde se observan los diferentes campos requeridos para el correcto funcionamiento del autómata, estos son:

- Qinicial: Estado inicial del autómata.
- Qfinal: Estado final del autómata.
- Cadena de entrada: Palabra a ser reconocida por el autómata.
- Transiciones: Programa o transiciones que ejecuta el autómata. Estas transiciones deben ser ingresadas de la siguiente forma:

  Estado actual,caracter leído de la cadena de entrada,caracter del tope de la pila,estado siguiente,acción a ejecutar en la pila (escribir o borrar caracteres)

Figura 54. Ejemplo $a^n b^n$

Presionando el botón RECONOCER se observan las acciones ejecutadas en la pila y los diferentes estados tomados por el autómata durante el proceso de reconocimiento de la cadena de entrada, finalizado dicho proceso se muestra el mensaje de aceptación en el campo SOLUCIÓN, Figura 55.

Figura 55. Respuesta de aceptación a la cadena de entrada

10.4 MÁQUINA DE TURING

Al seleccionar esta opción se abre la ventana de la Figura 56, la cual permite crear y guardar nuevos programas para este autómata, contando con una ayuda para cada uno de los diferentes campos, que se obtiene presionando ctrl+h. También se pueden desplegar algunos ejemplos, que permiten observar la forma de ingresar adecuadamente los datos y el funcionamiento de este autómata.

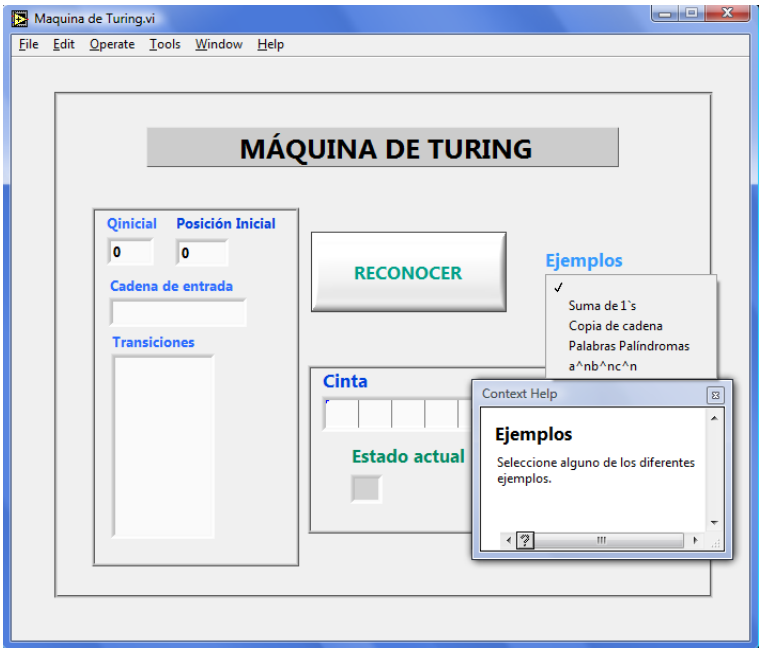


Figura 56. Ventana principal Máquina de Turing

Un ejemplo es $a^n b^n c^n$, Figura 57 donde se observan los diferentes campos requeridos para el correcto funcionamiento del autómata, estos son:

- Qinicial: Estado inicial del autómata.
- Posición inicial: Posición inicial de la cabeza lectura/escritura en la cinta.
- Cadena de entrada: Palabra a ser reconocida por el autómata.
- Transiciones: Programa o transiciones que ejecuta el autómata. Estas transiciones deben ser ingresadas de la siguiente forma:

> *Estado actual,caracter leído,(espacio)caracter a escribir en la posición actual (posición del caracter leído),dirección a mover la cabeza de lectura/escritura (R ó L ó H),estado siguiente*

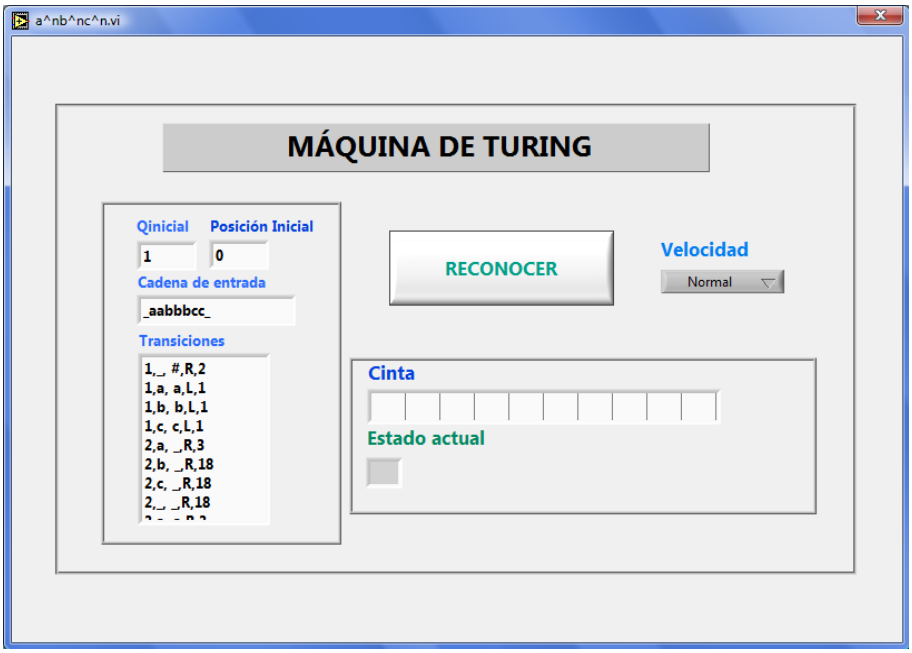


Figura 57. Ejemplo $a^n b^n c^n$

Presionando el botón RECONOCER se observan las acciones ejecutadas en la cinta durante el proceso de reconocimiento de la cadena de entrada, finalizado dicho proceso se muestra el mensaje de aceptación en el campo SOLUCIÓN, Figura 58.

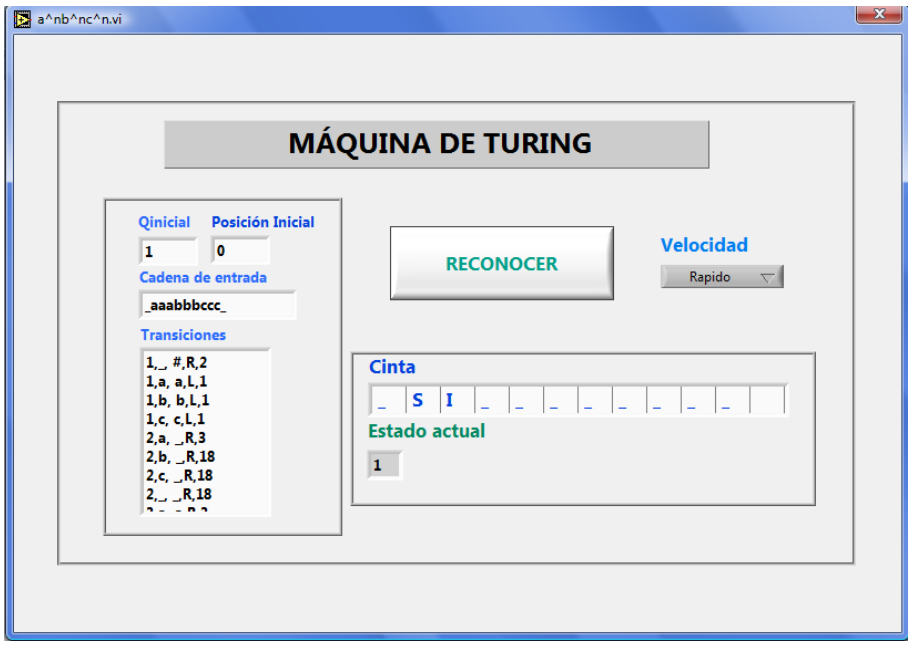


Figura 58. Respuesta de aceptación a la cadena de entrada