

# COMPUTABILIDAD

- ❖ Rama de la lógica matemática y ciencia de la computación
- ❖ También conocido como Teoría de la recursión
- ❖ Intenta responder las preguntas:
  - ¿Que se puede calcular? (computar)
  - ¿Que no se puede calcular?
  - ¿Cuál es el límite entre estas dos?

## MODELO COMPUTACIONAL

- ❖ Es un objeto matemático, definido en papel, que nos permite analizar las capacidades, propiedades y límites de la computabilidad.
- ❖ Existen **diferentes modelos**: Cada uno de ellos permite resolver diferentes problemas
- ❖ Se pueden comparar entre ellos para determinar su poder computacional

### Ejemplos:

- Autómatas Finitos
- Autómatas de Pila
- Autómatas con dos pilas
- Autómatas celulares
- Máquinas de Turing
- Cálculo Lambda
- Oráculo

## ALFABETO

- ❖ Conjunto finito y no vacío de símbolos
- ❖ Ej:  $\{0,1\}$  ← binario,  $\{a,b,c,d,e,f,g,h,i,j,\dots,z\}$ ,  $\{0,1,i,j,k\}$
- ❖ **Cadenas**:
  - Secuencia finita de símbolos de ese alfabeto
  - Cada cadena tiene una longitud (cantidad de símbolos en la misma)
  - Ejs:
    - 001001 es una cadena del alfabeto  $\{0,1\}$  de longitud 6
    - Hola es una cadena del alfabeto  $\{a,b,c,d,\dots,z\}$  de longitud 4
  - La **cadena vacía** (empty string) es la cadena de longitud. Se la describe con la letra  $\epsilon$ .

## LENGUAJES

- ❖ Conjunto de cadenas (strings) en un alfabeto
- ❖ Ejemplos:
  - Cadenas en alfabeto binarios terminados en 0
  - Cadenas en alfabeto  $\{a,b,c,d,e,f,g,h,i,j,\dots,z\}$  de longitud 5
  - Cadenas en alfabeto decimal capicúas
  - Lenguaje de programación
  - Castellano

# AUTÓMATAS FINITOS

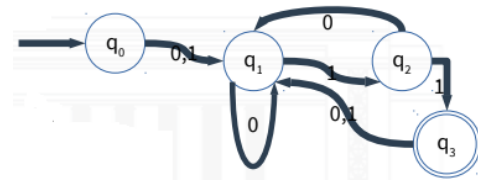
- ❖ Corresponde al modelo de computación **más simple**
- ❖ Características:
  - **No tiene memoria**
  - Reconoce un **número finito de “mensajes”**
  - Tiene un **estado** en el que se encuentra
- ❖ Ej: Una puerta automática
  - Puede estar abierta o cerrada (2 estados)
  - Tiene 2 sensores (AFUERA, ADENTRO)
  - Cada sensor reconoce la presencia (o ausencia) de una persona
  - Se abre si algún sensor reconoce una presencia
  - Sino, se cierra o mantiene cerrada

## Definición Formal

- ❖ Un autómata finito “M” es una 5-Tupla  $(Q, \Sigma, \delta, q_0, F)$  donde:
  - $Q$ : set finito llamado “estados” (ejemplo:  $q_0, q_1, q_2, q_3$ )
  - $\Sigma$ : set finito llamado “alfabeto” (ejemplo: 0,1)
  - $\delta: Q \times \Sigma \rightarrow Q$  es la función de transición
  - $q_0 \in Q$  estado inicial (ejemplo  $q_0$ )
  - $F \subseteq Q$  set de estados de aceptación (ejemplo  $q_3$ )

## Función de Transición: Representación gráfica

- ★ Cada estado es un vértice
- ★ Los estados finales tienen un borde doble
- ★ Las transiciones son ejes dirigidos desde un vértice
- ★ Las aristas están rotuladas por el o las símbolos del alfabeto que dispara el
- ★ pasaje de estado
- ★ El estado de inicio tiene un eje entrante que no tiene origen en otro vértice

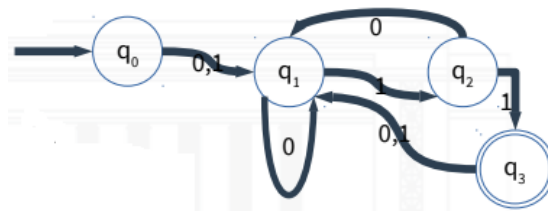


## Función de Transición: Tabla de transición

	0	1
$q_0$	$q_1$	$q_1$
$q_1$	$q_1$	$q_2$
$q_2$	$q_1$	$q_3$
$q_3$	$q_1$	$q_1$

## Cómputo:

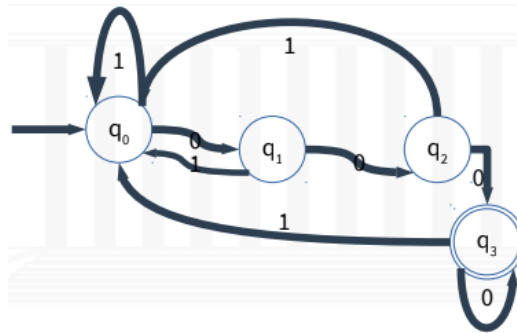
- El autómata recibe un String de entrada (escrito en el alfabeto  $\Sigma$ )
- Procesa el mismo, partiendo del estado inicial y utilizando la función de transición
- Retorna una salida de “Aceptación”: Si al terminar de procesar el string el estado final corresponde a uno de aceptación o “Rechazo” si no es de aceptación.
- **Ejemplo:**
  - Dado el string: 011110
  - La progresión de estados es:  $q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_1 \rightarrow q_2 \rightarrow q_1$
  - Retorna salida de “Rechazo”



- Dado el string: 001011
- La progresión de estados es:  $q_0 \rightarrow q_1 \rightarrow q_1 \rightarrow q_2 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$
- Retorna salida de "Aceptación"

#### ❖ Lenguaje de Máquina

- Sea A el set de todos los String que la máquina M acepta ("Aceptar" es el proceso de finalizar la ejecución en el estado de aceptación)
- Llamaremos a A al lenguaje de la máquina  $M \rightarrow L(M) = A$
- Diremos que "M reconoce A" (o que M acepta A)
- EJ:
  - La máquina reconoce el lenguaje:  $L(M) = \{ w / w \text{ binarias terminadas 3 "0" o más} \}$



### 1. Cómputo: Definición formal

Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un autómata finito y  $w = w_1 w_2 \dots w_n$  una cadena donde cada  $w_i$  es parte del alfabeto  $\Sigma$ . Entonces M acepta w si existe una secuencia de estados  $r_0, r_1, \dots, r_n$  en Q con las condiciones:

- i.  $r_0 = q_0$
- ii.  $\delta(r_i, w_{i+1}) = r_{i+1}$  para todo  $i=0, \dots, n-1$
- iii.  $r_n \in F$

## AUTÓMATAS FINITO NO DETERMINÍSTICOS

- ➔ En un proceso de **cómputo**, cada cambio de estado está determinado por el símbolo leído.
- ➔ En un proceso **determinístico**, en base al estado actual y a un símbolo leído, sabremos cuál será el próximo estado
- ➔ En un proceso **no determinista**, varias opciones pueden existir por el siguiente estado en cualquier punto

Un autómata finito no determinista:

- ➔ Es aquel para el que **cualquier transición** posible **permite tomar diferentes estados**.
- ➔ Al computar una cadena, se van generando en paralelo **diferentes ramificaciones de ejecución**

Los autómatas finitos deterministas son un caso especial de los AFND donde se limita a un único estado por transición.

Ejemplo: El siguiente es un posible AFND



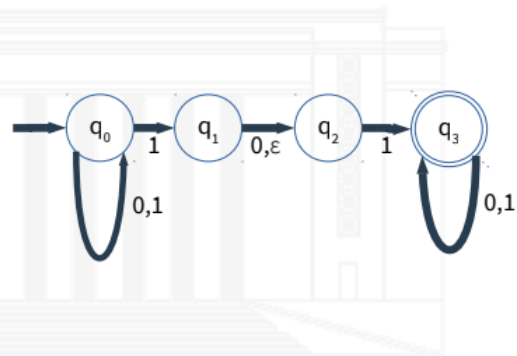
- ★ Desde un mismo estado y un mismo mensaje se puede transicionar a más de un
- ★ estado en forma simultánea ( $q_0, 1 \rightarrow q_0 \text{ y } q_1$ )
- ★ La utilización del símbolo  $\epsilon$  entre dos estados “x”, “y” indica que al llegar al estado “x” también se llega a “y” sin leer ningún símbolo (si llego a  $q_1$  también llego a  $q_2$ )
- ★ Si desde un estado x, se lee un símbolo del que no hay un estado siguiente establecido, entonces finaliza la rama de ejecución en curso ( $q_2, 0 \rightarrow \emptyset$ )

### Definición formal

- ❖ Un autómata finito no determinista “M” es una 5-Tupla  $(Q, \Sigma, \delta, q_0, F)$  donde:
  - Q: set finito llamado “estados”
  - $\Sigma$ : set finito llamado “alfabeto” (ejemplo: 0,1)
  - $\delta: Q \times \Sigma \cup \epsilon \rightarrow P(Q)$  es la función de transición
  - $q_0 \in Q$  estado inicial (ejemplo  $q_0$ )
  - $F \subseteq Q$  set de estados de aceptación (ejemplo  $q_3$ )
- ❖ Donde:
  - $\Sigma \cup \epsilon = \Sigma \cup \{\epsilon\}$ ,
  - $P(Q)$ : Es el conjunto de potencia de Q (conjunto formado por todos los subconjuntos de Q)

### Función de Transición: Tabla de transición

	0	1	$\epsilon$
$q_0$	$\{q_0\}$	$\{q_0, q_1\}$	$\emptyset$
$q_1$	$\{q_2\}$	$\emptyset$	$\{q_2\}$
$q_2$	$\emptyset$	$\{q_3\}$	$\emptyset$
$q_3$	$\{q_3\}$	$\{q_3\}$	$\emptyset$



### Cómputo

El autómata recibe un String de entrada (escrito en el alfabeto  $\Sigma$ )

Realiza una iteración:

Si existe una flecha de  $\epsilon$ , sin leer el carácter  
se divide el proceso en diferentes copias de sí mismo  
Una se mantiene en el estado en el que se encuentra  
El resto en los estados según la función de transición del  $\epsilon$   
Cada copia se seguirá procesando en paralelo.

Lee el próximo carácter

Según la función de transición se divide el proceso en una copias de sí mismo por cada

estado destino

Cada copia estará en un estado según la función de transición

Cada copia se seguirá procesando en paralelo.

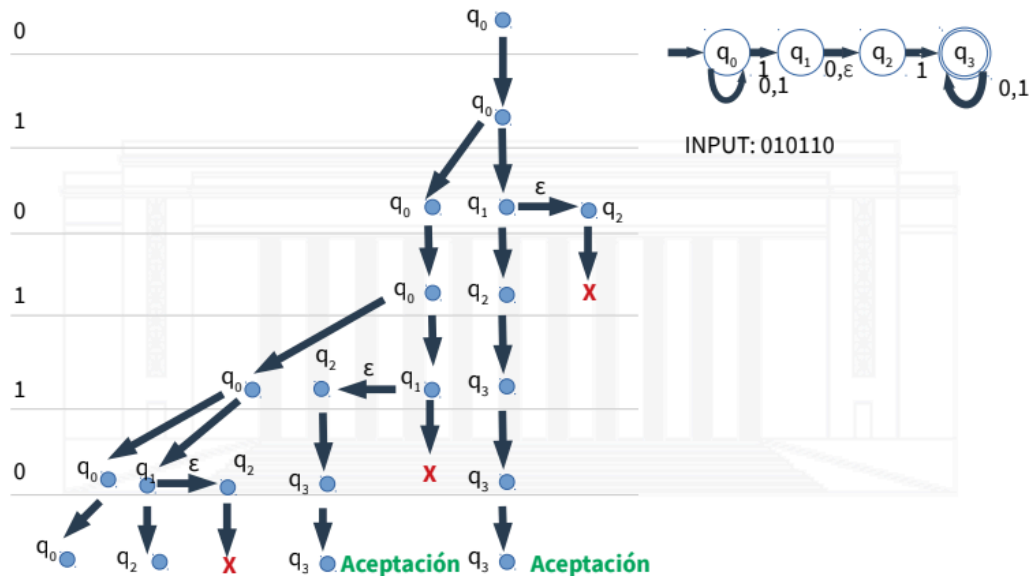
**Al finalizar el procesamiento del string,**

Si al menos una rama termina en estado de "Aceptación", retorna "Aceptación"

De lo contrario retorna "Rechazo".

**A medida que se procesen las ramas se pueden abandonar ramas que lleguen a estados siguientes  $\emptyset$**

**Ejemplo:**



## EQUIVALENCIA ENTRE AFND y un AFD

- ❖ Dos máquinas son equivalentes si reconocen el mismo lenguaje:
  - Para todo string de entrada el resultado del cómputo es el mismo.
  - Ambas aceptan o rechazan las mismas cadenas
- ❖ Los autómatas finitos deterministas:
  - Son un caso **especial de los AFND**
  - Donde se limita a un **único estado por transición**

### Poder de computación

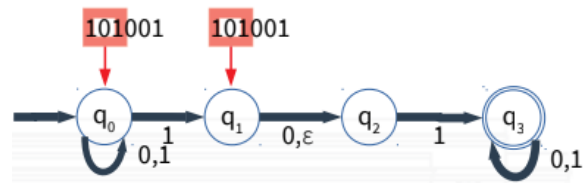
¿Cada máquina no determinística, tiene un equivalente determinístico? ¿O existen lenguajes que solo reconoce uno de ellos?

¿Podemos demostrar que cualquier AFND se puede construir con un AFD? → En ese caso ambos modelos de computación son equivalentes

¿Podemos demostrar que al menos existe un AFND que no se puede construir con un AFD? → En ese caso AFND tienen más poder de computación que las AFD

### Ramificaciones de un AFND

- ❖ Sea una AFND de  $n$  estados
- ❖ A medida que se computa un String se van ramificando las ejecuciones según la función de transición.
- ❖ Cada ramificación es como un puntero a un estado dentro de la máquina.
- ❖ Un AF **no tiene memoria**, se ejecuta según el símbolo leído y el estado actual



- ❖ Si dos ramificaciones luego de una transición coinciden en un mismo estado, se fusionan en solo 1. La historia de cómo llegaron no impacta en cómo se comportan a partir de ese momento

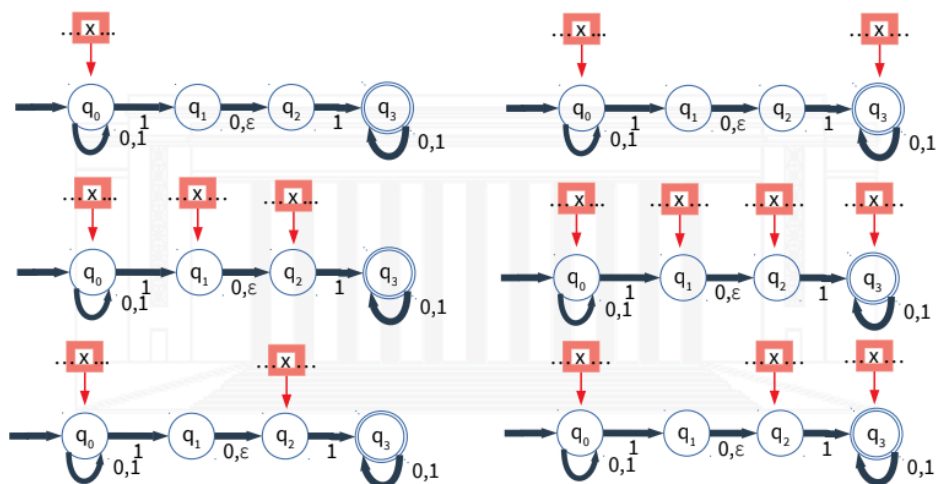


### → ¿Existe un máximo de ramificaciones posibles?

Si!, no pueden existir más variantes que todas las combinaciones de estados

- Una AFND de  $n$  estados tiene un máximo de  $2n$  combinaciones máximas de posibles “punteros a estados” simultáneos (Algunas de estas pueden no existir para ningún proceso de string particular de esa AFND)

Ejemplo: Existen  $2^4$  combinaciones de ramificaciones como máximo, pero solo las 6 siguientes son posibles.

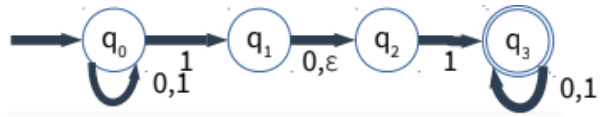


- ❖ Sea una AFND de “ $n$ ” estados, construimos un AFD con:
  - $Q$ : todas las combinaciones posibles de ramificaciones:  $2^k$
  - $\Sigma$ : alfabeto de la AFND

- $\delta: Q \times \Sigma \rightarrow Q$  la transición de un estado de una ramificación a otra (única)
- $q_0 : Q$  “ramificación” inicial de la AFND
- $F$ : cada ramificación posible que tenga al menos 1 estado de finalización.
- ❖ Este nuevo AFD es equivalente a la AFND  $\rightarrow$  cumple las características de la AFD.
- ❖ El poder de “expresión” entre AFD y AFND es **equivalente!**

#### Ejemplo:

Para la AFND de la figura el alfabeto será el mismo:  $\{0,1\}$



Creamos los “meta” estados:  $\{q_0\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_0, q_3\}, \{q_0, q_1, q_2\}, \{q_0, q_2, q_3\}, \{q_0, q_1, q_3\}, \{q_0, q_1, q_2, q_3\}, \{q_1\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}, \{q_2\}, \{q_2, q_3\}, \{q_3\}$ ,

Será el estado inicial  $\{q_0\}$  y serán los estados finales aquellos que tengan a  $q_3$ :  $\{q_0, q_3\}, \{q_0, q_2, q_3\}, \{q_0, q_1, q_3\}, \{q_0, q_1, q_2, q_3\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}, \{q_2, q_3\}, \{q_3\}$

#### La función de transición

	0	1
$\{q_0\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$\{q_0, q_3\}$	$\{q_0, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_3\}$
$\{q_0, q_2, q_3\}$	$\{q_0, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$

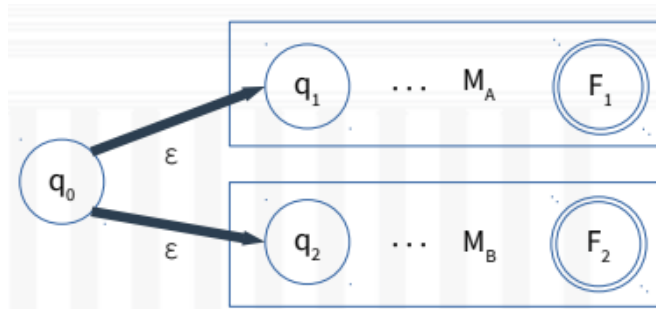
	0	1
$\{q_1\}$	$\{q_2\}$	$\emptyset$
$\{q_1, q_2\}$	$\{q_2\}$	$\{q_3\}$
$\{q_1, q_3\}$	$\{q_2, q_3\}$	$\{q_3\}$
$\{q_1, q_2, q_3\}$	$\{q_2, q_3\}$	$\{q_3\}$
$\{q_2\}$	$\emptyset$	$\{q_3\}$
$\{q_2, q_3\}$	$\{q_3\}$	$\{q_3\}$
$\{q_3\}$	$\{q_3\}$	$\{q_3\}$
$\emptyset$	$\emptyset$	$\emptyset$

## LENGUAJES REGULARES

- ❖ Un lenguaje es **regular** si **existe algún autómata finito que lo reconozca**.
- ❖ Existen algunas operaciones que se pueden aplicar a los lenguajes regulares llamadas operaciones regulares cuyo resultado es también un lenguaje regular.
- ❖ Sean A, B dos lenguajes regulares:
  - Unión:  $A \cup B = \{x / x \in A \text{ o } x \in B\}$
  - Concatenación:  $A \circ B = \{xy / x \in A, y \in B\}$
  - Estrella (star):  $A^* = \{x_1 x_2 \dots x_k / k \geq 0 \text{ y cada } x_i \in A\}$
- ❖ Ejemplo:
  - $A = \{\text{auto, avion}\}$  y  $B = \{\text{rojo, verde, azul}\}$
  - $A \cup B = \{\text{auto, avion, rojo, verde, azul}\}$
  - $A \circ B = \{\text{autorrojo, autoverde, autoazul, avionrojo, avionverde, avionazul}\}$
  - $A^* = \{\epsilon, \text{auto, avion, autoauto, avionavion, autoavion, avionauto, ...}\}$

### UNIÓN

- ❖ Sean A,B autómatas finitos, queremos ejecutar simultáneamente la máquina A y B y ver si alguna de ellas termina en estado de aceptación.



- ❖ Creamos un nuevo autómata que incluya A y B, con un estado inicial y una transición sin lectura al estado inicial de A y B.

### CONCATENACIÓN

- ❖ Sean A, B autómatas finitos, Queremos que una se ejecute inmediatamente luego de la otra para ver si el resultado termina en estado de aceptación



- ❖ Creamos un nuevo autómata que incluya A y B Se agrega transición con símbolo  $\epsilon$  entre los estados de aceptación de A y el estado de inicio de B.
- ❖ Los estados de aceptación de A dejan de serlo.

### ESTRELLA

- ❖ Sea A un autómata finito, queremos que se ejecute nuevamente inmediatamente al llegar al estado de aceptación.



- ❖ Creamos un nuevo autómata:
  - Con un nuevo estado inicial y al mismo tiempo de aceptación
  - Se agrega transición con símbolo  $\epsilon$  entre el nuevo estado y el estado de aceptación de A
  - Se agrega transición con símbolo  $\epsilon$  entre los estados de finalización de A y el nuevo estado
  - Los estados de aceptación de A dejan de serlo

### EXPRESIONES REGULARES

- ❖ Se conoce como “expresiones regulares” al uso de operaciones regulares para construir expresiones describiendo lenguajes
- ❖ Cualquier expresión regular se puede resolver con un AFD (o AFND). Por lo tanto son equivalentes en su poder de expresión (y reconocimiento).
- ❖ R es una expresión regular si es:
  - a para algún a en el alfabeto  $\Sigma$
  - $\epsilon$  (string vacío)
  - $\emptyset$  (ausencia de string)
  - $(R1 \cup R1)$
  - $(R1 \circ R2)$
  - $(R1)^*$



(Con R1 y R2 expresiones regulares)

## LENGUAJES NO REGULARES

- ❖ Lenguajes que un AF no puede reconocer. Ej:  $B = \{0^n 1^n / n \geq 0\}$
- ❖ El AF carece de memoria → Operaciones que requieren “recordar” algo (salvo particularidades) no son posibles.
- ❖ Un lenguaje tiene que cumplir una propiedad para ser regular
  - Utilizaremos un teorema conocido “**pumping lemma**” para verificarlo
- ❖ **Pumping lemma (Lema del “bombeo”)**
  - Si un lenguaje no tiene la propiedad → NO ES REGULAR
  - Todos los strings de un lenguaje regular → pueden “bombarse” si son al menos tan largos que un cierto valor especial, llamado longitud de bombeo.
  - Cada string contiene una sección que puede repetirse cualquier cantidad de veces con el string resultante aun perteneciendo al lenguaje
  - Si A es un lenguaje regular, existe un número p (longitud de bombeo), donde si s es cualquier string A de longitud al menos p, entonces s puede ser dividido en 3 partes  $s=xyz$  que satisfacen las condiciones:
    - Para cada  $i \geq 0$ ,  $xy^i z \in A$
    - $|y| > 0$ ,
    - $|xy| \leq p$

## AUTÓMATAS DE PILA

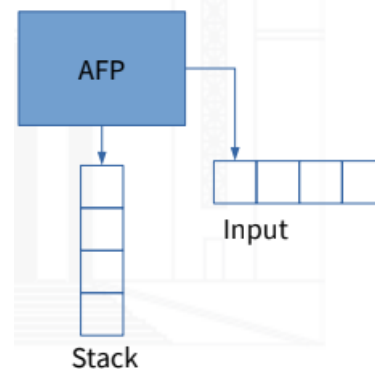
- ❖ Falta de memoria → Limitación fundamental de AF
- ❖ El agregado de una memoria → Posibilita reconocer algunos lenguajes no regulares
- ❖ Existen diferentes formas de memoria. Agregaremos una de las más simples: una **PILA**

El Agregado de una memoria de tipo pila a un AFND:

- Permite leer y guardar un símbolo del alfabeto de pila. (el alfabeto de pila y de input puede ser diferente)

En cada iteración:

- Se lee un símbolo de la entrada (o  $\epsilon$ )
- Se lee un símbolo de la pila (o  $\epsilon$ )
- Se modifica el estado (o se queda en el mismo)
- Se graba un símbolo de la pila (o  $\epsilon$ )
- Si se realiza una ramificación, se duplica el stack con su contenido



- ❖ Un autómata de pila “M” es una 6-Tupla  $(Q, \Sigma, \Gamma, \delta, q_0, F)$  donde:
  - $Q$ : set finito de “estados”
  - $\Sigma$ : alfabeto de entrada
  - $\Gamma$ : alfabeto de la pila
  - $\delta: Q \times \Sigma \times \Gamma \rightarrow P(Q \times \Gamma)$  es la función de transición
  - $q_0 \in Q$  estado inicial (ejemplo  $q_0$ )
  - $F \subseteq Q$  set de estados de aceptación (ejemplo  $q_3$ )

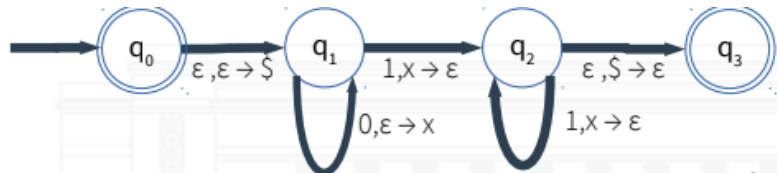
- ❖ Ejemplo: Queremos crear un autómata de pila que reconozca el siguiente lenguaje:  
 $A = \{0^n 1^n \mid n \geq 0\}$

Definimos:

→ Alfabeto de entrada  $\Sigma = \{0, 1\}$

→ Alfabeto de pila  $\Gamma = \{x, \$ \}$

Proponemos el siguiente diagrama de estados:



Con su correspondiente tabla de transición:

$\Sigma$	0	0	0	1	1	1	$\epsilon$	$\epsilon$	$\epsilon$
$\Gamma$	x	\$	$\epsilon$	x	\$	$\epsilon$	x	\$	$\epsilon$
$q_0$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_1, \$)\}$
$q_1$	$\emptyset$	$\emptyset$	$\{(q_1, x)\}$	$\{(q_2, \epsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$q_2$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_2, \epsilon)\}$	$\emptyset$	$\emptyset$	$\emptyset$	$\{(q_3, \epsilon)\}$	$\emptyset$
$q_3$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Con los siguientes strings:

- 0011 → es reconocido
- 011 → es rechazado
- 010 → es rechazado

## GRAMÁTICAS LIBRES DE CONTEXTO

- ❖ Al igual que los Lenguajes regulares, son reconocidos por los autómatas finitos
- ❖ Existen lenguajes que pueden ser reconocidos por autómatas de pila
- ❖ Una gramática consiste en una colección de reglas de sustitución
  - variable → variables y/o terminales
- ❖ Los terminales son el alfabeto del lenguaje:
  - Una cadena del lenguaje está conformada únicamente por terminales
  - Todas los string del lenguaje constituyen la **gramática del lenguaje**
- ❖ Existe una variable de inicio:
  - Desde la misma se derivan todas las cadenas válidas de ese lenguaje.
  - La secuencia de sustituciones para obtener una cadenas se conoce como **derivación**
- ❖ Ejemplo:  
 Sean las reglas:  $A \rightarrow 0A1$ ,  $A \rightarrow B$ ,  $B \rightarrow \#$   
 Con A variable de inicio, se puede derivar:  
 $A \rightarrow 0A1 \rightarrow 00A11 \rightarrow 000A111 \rightarrow 000B111 \rightarrow 000\#111$   
 $A \rightarrow B \rightarrow \#$

→  $L = \{0^n \# 1^n \mid n \geq 0\}.$

- ❖ Todos los **strings** que se pueden **generar mediante derivación**
  - **Constituyen el lenguaje de la gramática**
- ❖ Si llamamos  $G_1$  a la una gramática determinada
  - Indicaremos  $L(G_1)$  al lenguaje de la gramática  $G_1$
- ❖ **Cualquier lenguaje** que pueda **generarse** por alguna **gramática libre de contexto**
  - Es conocido como **lenguaje libre de contexto**
- ❖ Una gramática libre de contexto es un 4-tupla  $(V, \Sigma, R, S)$ , donde:
  - $V$  es un set finito de variables
  - $\Sigma$  es un set finito disjunto a  $V$  de terminales
  - $R$  es un set finito de reglas, cada una parte de una variable y termina en un string compuesto de variables y/o terminales
  - $S \in V$  es la variable de inicio

### GRAMATICAS AMBIGUAS

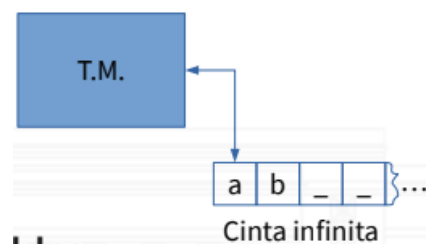
- ❖ Si un String se puede derivar de dos o más maneras diremos que es un **String ambiguo**.
- ❖ Una gramática con al menos un String ambiguo es una **gramática ambigua**.
- ❖ Ejemplo:  
Reglas:  $A \rightarrow 0B \mid 0C1 \mid 1$   $B \rightarrow 0101$   $C \rightarrow 0A0$   
 $A \rightarrow 0B \rightarrow 00101$   
 $A \rightarrow 0C1 \rightarrow 00A01 \rightarrow 00101$

### LENGUAJES NO LIBRES DE CONTEXTO

- ❖ Un lenguaje es libre de contexto **si y sólo si** existe un **autómata finito de pila** que lo reconoce
- ❖ Un lenguaje regular pertenece a los lenguajes libres de contexto. Cualquier AFND se puede construir con un Autómata de pila sin leer o escribir en la pila.
- ❖ Existen lenguajes que son no libres de contexto:
  - Estos no pueden ser reconocidos por un autómata de pila
  - Ejemplo:  $B = \{a^n b^n c^n \mid n \geq 0\}$

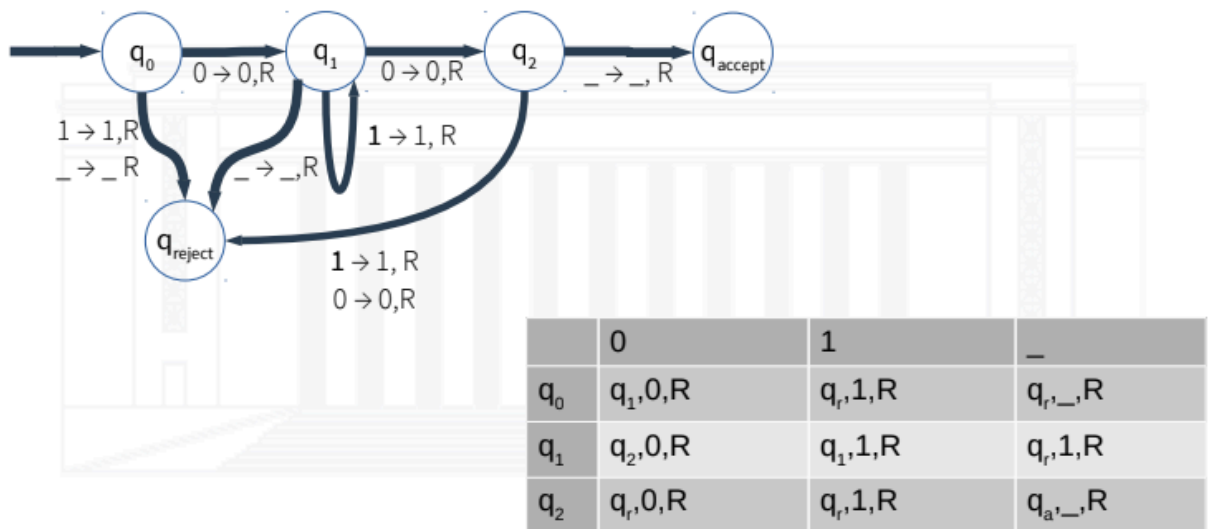
### MÁQUINA DE TURING

- ❖ Similar a un autómata finito, Pero con **memoria infinita**
- ❖ Utiliza una cinta infinita dividida en celdas donde tiene el input inicial a computar (y el resto en blanco)
- ❖ Puede leer o grabar en cada secuencia: 1 símbolo en la celda donde está ubicado y moverse luego 1 celda a la derecha o izquierda.
- ❖ Tiene un conjunto finito de estados. Desde cada estado se puede transicionar a otros según lo leído en la cinta y una función de transición
- ❖ Existen estados especiales (Halting) que al llegar a ellos finaliza el cómputo:
  - "aceptación"
  - "rechazo"
- ❖ El cómputo podría no llegar nunca a finalizar → Si no se llega a un estado de finalización



- ❖ Una máquina de turing M es una 7-tupla  $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$  donde:
  - $Q$ : set finito de “estados”
  - $\Sigma$ : alfabeto de entrada (sin contener el símbolo “blanco”  $\_$ )
  - $\Gamma$ : alfabeto de la cinta, donde  $\_ \in \Gamma$  y  $\Sigma \cap \Gamma = \emptyset$
  - $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$  es la función de transición (L:mover cabezal a la izquierda, R:moverlo a la derecha)
  - $q_0 \in Q$  estado inicial
  - $q_{\text{accept}} \in Q$  estado de aceptación
  - $q_{\text{reject}} \in Q$  estado de rechazo ( $q_{\text{reject}} \neq q_{\text{accept}}$ )
- ❖ Ejemplo: Queremos construir un TM que acepte el siguiente lenguaje  $L(M) = \{ w \in \{0,1\}^* / w=01^*0 \}$   
 El lenguaje del input  $\rightarrow \Sigma = \{0,1\}$   
 El lenguaje de la cinta  $\rightarrow \Gamma = \{0,1,\_ \}$

### Diagrama de estados

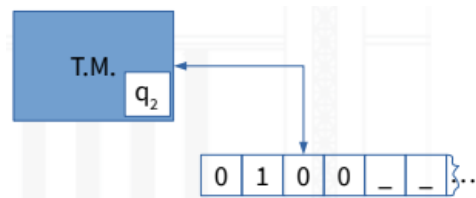


### CONFIGURACIÓN

- ❖ Mientras una TM computa se modifican:
  - El contenido de la cinta
  - La posición del “cabezal”
  - El estado actual
- ❖ Configuración  $\rightarrow$  Estado de los 3 elementos anteriores de una TM en un determinado momento del cómputo

Podemos representar la configuración:

- Como 2 strings u y v que separan el contenido de la cinta donde se encuentra el cabezal
- Entre u y v se coloca el estado actual
- En el ejemplo, la configuración actual es 01q200



- ❖ La configuración inicial  $\rightarrow$  estado inicial más el input de la TM:  $q_0 w_0 w_1 w_2 \dots$
- ❖ Configuración de aceptación  $\rightarrow$  Aquella donde el estado es  $q_{\text{accept}}$
- ❖ Configuración de rechazo  $\rightarrow$  Aquella donde el estado es  $q_{\text{reject}}$

### Sucesión de configuraciones

- ❖ Una configuración  $c_x$  cambia a  $c_y$  en un paso → **Funcion de Transicion**
- ❖ Podemos, entonces, ver la función de transición como:
  - Una acción sobre una configuración que determina una nueva configuración
  - Modifica (o no) la celda y moviendo el cabezal a la derecha o a la izquierda
- ❖ Para una TM determinada diremos
  - Que la configuración  $C_i$  produce la configuración  $C_j$ , si dada la función de transición, la MT puede pasar de  $C_i$  a  $C_j$  en **un solo paso**.

### Cómputo

- ❖ Una máquina de Turing M, Acepta un input w si Una secuencia de configuraciones  $C_1, C_2, \dots, C_k$  existe donde:
  - $C_1$  es la configuración inicial de M con el input w
  - Cada configuración  $C_i$  determina a  $C_{i+1}$
  - $C_k$  es una configuración de aceptación
  -

### Lenguajes turing reconocibles

- ❖ La colección de string que M acepta, es el lenguaje reconocido de M. Lo denotaremos como  $L(M)$
- ❖ Un lenguaje es Turing reconocible → Si existe algún TM que lo reconoce

### Loop en una Turing Machine

- ❖ Cuando se computa un input en una TM, pueden pasar 3 cosas:
  - Acepta
  - Rechaza
  - Loopea (nunca termina)
- ❖ A veces determinar si una TM loopea o está tardando en aceptar/rechazar es difícil

### Lenguajes turing decidibles

- ❖ Llamamos decidores a aquellos TM que no loopea ante ningún input posible
  - **siempre** acepta o rechaza
- ❖ Un lenguaje es Turing decidable
  - Si existe algún TM que lo decide
- ❖ Un lenguaje Turing decidable es **también** un **lenguaje Turing reconocible**

## VARIANTES DE LAS MÁQUINAS DE TURING

- ❖ Existen múltiples variantes de las máquinas de Turing:
  - Con posibilidad de no avanzar además de retroceder o avanzar el cabezal
  - Con múltiple cinta
  - No determinística
  - Con impresora ( cinta output)
- ❖ Todos estos → Se han demostrado que tienen el mismo poder de cómputo → (pueden reconocer los mismos lenguajes)

### TURING MACHINE MULTICINTA

- ❖ Variante de una TM con múltiples cintas
- ❖ Cada cinta tiene un cabezal para leer y escribir
- ❖ Inicialmente:
  - La primer cinta contiene el input
  - El resto de las cintas se encuentra en blanco
- ❖ La función de transición permite leer, escribir y moverse en alguna o todas las cintas

- ❖ Formalmente, con  $k$  cintas
  - $\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k \rightarrow (S \text{ implica que el cabezal no se mueve})$
- ❖ Por ejemplo  $\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$ :
  - Se lee “si la máquina está en el estado  $q_i$  y leo en la cinta  $x$  el símbolo  $a_x$ . Entonces se realiza una transición al estado  $q_j$ , se graba en la cinta  $y$  el símbolo  $b_y$  y en cada cinta muevo al cabezal según lo que se indica”

### Poder de cómputo de la TM multicinta

- ❖ Si una TM multicinta puede **reconocer más lenguajes** que una TM:
  - Tiene **más poder de cómputo** o es “más robusta”
- ❖ Si demostramos que **dado una** máquina se puede **construir otra** equivalente
  - **Reconocen los mismos lenguajes**
- ❖ Mediante la construcción teórica de cualquier TM multicinta en su equivalente TM probaremos que estos modelos son equivalentes

### Equivalencia entre TM y Multicinta

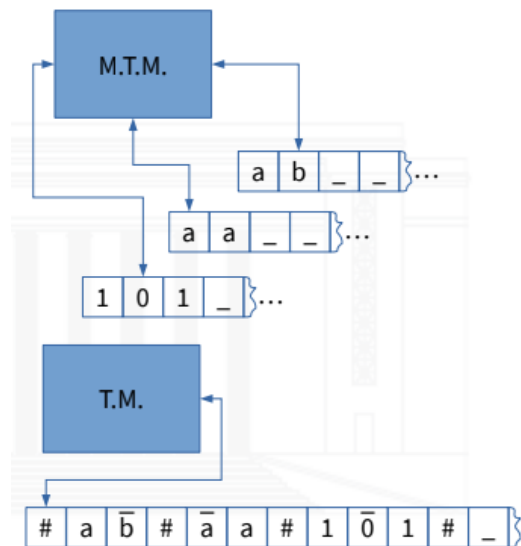
Sea una TM de  $K$  cintas (MTM) generamos una TM equivalente

Pasos:

- Creamos Un nuevo carácter especial “#” de separación de cintas
- Concatenamos cada input de la cinta de la MTM a la la cinta de la TM separados por el “#”
- Por cada símbolo en la MTM creamos uno equivalente marcado que usaremos para identificar la posición virtual de un cabezal

Para simular una iteración la TM explora la cinta desde el primer # al  $(k+1)$ -esimo # para determinar los símbolos marcados y con eso la posición de las cabezas virtuales

Luego realiza una segunda pasada para actualizar las cintas de acuerdo a lo que dicta la función de transición de la M.T.M.



Si en alguna iteración desde el cabezal virtual llega a la marca “#” que limita el contenido de esa cinta se debe mover todos los símbolos siguientes 1 posición creando una celda nueva en blanco “\_”. Luego continúa la simulación

### Turing machine no determinística

- ❖ Permite en un punto del cómputo transicionar en simultáneo a varias posibilidades
- ❖ Podemos describir su función de transición como:
  - $\delta: Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{L, R\})$
- ❖ El cómputo de una TM no determinística
  - Corresponde a un árbol de cuyas ramas corresponden a diferentes posibilidades de la máquina
  - Si al menos 1 rama termina en el estado de aceptación, entonces el resultado es de aceptación

## Equivalencia entre TM y Multicinta

Podemos simular una TM no determinística:

- Para eso realizaremos una nueva TM que recorra cada nodo del árbol de la computación no determinística
- Si encuentra un estado de aceptación termina, sino continua (pudiendo loopear por siempre)

El recorrido del árbol debe realizarse mediante Breadth first search (usar DFS podría ocasionar seleccionar una rama que loopea para siempre)

Llamaremos  $b$  a la cantidad máxima de ramificaciones simultáneas según la función de transición

Con  $b$  generamos el alfabeto

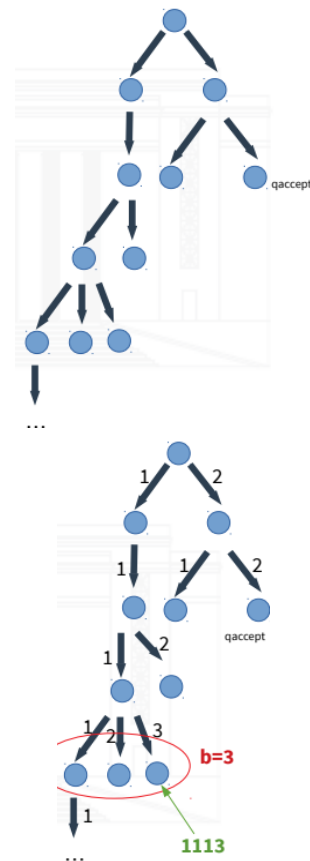
$$\Gamma_b = \{1, 2, \dots, b\}$$

Cada eje que sale de un nodo:

- le asignaremos un número del alfabeto  $\Gamma_b$  de 1 a  $x$ , con  $x \leq b$

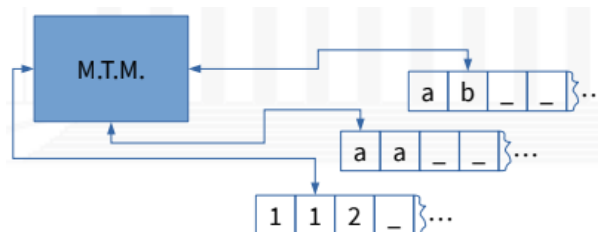
A cada nodo del árbol:

- Le asignaremos una etiqueta utilizando un string en el alfabeto  $\Gamma_b$
- Corresponde al camino de ejes realizado para llegar al mismo
- A la raíz le corresponde el string vacío  $\epsilon$



Utilizaremos un MTM con 3 cintas:

- ❖ La cinta 1 contendrá el input del NDTM
- ❖ La cinta 2 se utilizará para simular el cómputo del input hasta una rama del árbol de computación no determinístico
- ❖ La cinta 3 realiza un seguimiento de la ubicación en el cálculo del árbol de computo



1. inicialmente la cinta 1 contiene el input y la cinta 3 el string vacío (raíz)
2. Repetir hasta aceptación (cada testeo es una rama del árbol)  
copiar el contenido de la cinta 1 a la cinta 2  
usar la cinta 2 para simular la NDTM con el input

En cada iteración

Aplicar la función de transición usando la opción de transición que indica el cabezal en la cinta 3 entre las posibles.

Si la rama a elegir no existe, o se llega a un estado de rechazo seguir con próxima rama

Si se llega a estado de aceptación terminar todo el proceso "aceptando".

Sino, mover a la derecha el cabezal de cinta 3 y repetir proceso de iteración del cabezal

Asignar a la cinta 3 la etiqueta de la próxima rama a simular

En el ejemplo, la secuencia de ramas a probar son  
E, 1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, 112 ...

(en azul aquellas “direcciones” inválidas)

El proceso terminará al simular la rama “22”

## TESIS CHURCH - TURING

### Entscheidungsproblem

- ❖ Problema de decisión lanzado por David Hilbert y Wilhelm Ackermann en 1928
- ❖ Implica encontrar un algoritmo general que decidiese si una fórmula del cálculo de primer orden es un teorema
- ❖ Turing propone su máquina automática (Máquina de Turing) como herramienta de cálculo.
- ❖ Paralelamente (y aventajando por unos meses), Alonzo Church desarrolla su cálculo lambda ( $\lambda$ -calculus) para el mismo motivo.
- ❖ Turing demuestra que su Máquina y el  $\lambda$ -calculus son equivalentes.

### Tesis de Church - Turing

- ❖ Con el trabajo de Church y Turing se dió un marco para definir que es un algoritmo
- ❖ Coloquialmente:
  - Todo algoritmo **existe si** es equivalente a una máquina de Turing
  - Todos los modelos matemáticos de cómputo posibles tienen igual o menor poder computacional que las máquinas de Turing. (con posibles queremos decir construibles en el mundo real, por ejemplo: la imposibilidad de realizar una cantidad de operaciones infinita en un tiempo finito)
- ❖ **Importante!** Es una tesis, por lo tanto **no está probado**, pero es **aceptado casi universalmente**

### 10mo problema: Un problema más sencillo

- ❖ Encontrar un algoritmo para determinar si una ecuación polinómica diofántica (con 2 o más incógnitas) con coeficientes enteros, tiene una solución entera.
- ❖ Solución entera: Si la evaluación del polinomio con una asignación de sus variables con valores enteros da como resultado cero (raíz entera)
- ❖ Si restringimos el 10mo problema a determinar si un polinomio de 1 variable tiene raíces enteras:
  - Expresamos el polinomio como:  $f(x) = c_1x^k + c_2x^{k-1} + \dots + c_{k+1}$
  - Podemos ir probando valores de  $x$ : 0, 1, -1, 2, -2, 3, -3, ...
- ❖ Resolución:
  - Podemos construir una TM para computar el problema
  - Definimos el lenguaje para nuestra TM  $P = \{p / p \text{ polinomio con variable } x \text{ con una raíz entera}\}$
  - Nuestra TM  $M$  será  $\rightarrow$  “Ante el input ( $p$ ): donde  $p$  polinomio con variable  $x$  evaluar  $p$  estableciendo  $x$  según la secuencia 0, 1, -1, 2, -2, 3, -3 ... Si en algún momento  $p$  se evalúa en 0, aceptar”
- ❖ Análisis de la TM:
  - El algoritmo encontrará eventualmente si el polinomio tiene una raíz entera. Si no tiene, loopeará eternamente
  - Por lo tanto, la TM anterior reconoce el lenguaje  $P$



- ¿Se puede evitar el posible loop? → Si! podemos acotar el rango donde se puede encontrar la raíz entera.
- ❖ Rango de la raíz entera:
  - Sea  $f(x_0) = c_1x_0^k + c_2x_0^{k-1} + \dots + c_{k+1} = 0$
  - Podemos operar algebraicamente:
 
$$c_1x_0^k + c_2x_0^{k-1} + \dots + c_{k+1} = 0 \Rightarrow c_1x_0^k = -(c_2x_0^{k-1} + \dots + c_{k+1}) \Rightarrow$$

$$|c_1x_0^k| = |-(c_2x_0^{k-1} + \dots + c_{k+1})| \Rightarrow |c_1||x_0^k| = |c_2x_0^{k-1} + \dots + c_{k+1}| \Rightarrow$$

$$|c_1||x_0^k| \leq |c_2x_0^{k-1}| + \dots + |c_{k+1}|$$
  - Sea  $C_{\max}$  el coeficiente con mayor número absoluto:
 
$$|c_1||x_0^k| \leq |c_{\max}x_0^{k-1}| + \dots + |c_{\max}| \Rightarrow |c_1||x_0^k| \leq |c_{\max}| * (|x_0^{k-1}| + \dots + 1) \Rightarrow$$

$$|c_1||x_0^k| \leq |c_{\max}| * (k|x_0^{k-1}|) \Rightarrow |x_0| \leq \frac{|c_{\max}| * k}{|c_1|}$$
- Entonces puedo restringir la búsqueda de la raíz entera entre valores  $\pm \frac{|c_{\max}| * k}{c_1}$  y mi TM se convierte en un decidor
- Como corolario:  $P = \{p/ p \text{ polinomio con variable } x \text{ con una raíz entera}\}$  es **TURING DECIDIBLE**
- ❖ Regresando a la ecuación polinómica diofántica:
  - Podemos construir una TM para computar el problema
  - Definimos el lenguaje para nuestra TM:
 
$$\rightarrow D = \{p/ p \text{ polinomio con 2 o más variables con una raíz entera}\}$$
  - Podemos ir probando los valores:
 
$$(0,0,\dots,0), (1,0,\dots,0), (1,1,\dots,0), \dots, (1,1,\dots,1), \dots, (0,0,\dots,-1), \dots$$
 todas las combinaciones posibles
  - Por lo tanto:  $D = \{p/ p \text{ polinomio con 2 o más variables con una raíz entera}\}$  es **TURING RECONOCIBLE**
- ❖ Se puede crear una TM para el problema que sea decidor?
  - En 1971, Yuri Matijasevich demostró que no es posible (utilizó el andamiaje creado por Church y Turing)
  - Por lo tanto  $D = \{p/ p \text{ polinomio con 2 o más variables con una raíz entera}\}$  **NO ES TURING DECIDIBLE**

## TURING UNDECIDIBLE: El problema de la parada

- ❖ Una **Turing Machine universal (UTM)** corresponde a una TM que **simula cualquier otra TM con un input arbitrario**.
- ❖ Propuesto por Turing entre 1936 y 1937
- ❖ Tanto el input como la descripción de la TM a simular se incluyen en el input de la UTM.
- ❖ Se lo considera el origen de la idea de un programa de computador almacenado utilizado por Von Neumann

### ¿Es decidible si una TM acepta un input?

- ❖ Construimos el lenguaje  $A_{TM} = \{(M,w) / M \text{ es una TM y } M \text{ acepta } w\}$
- ❖ Construimos la siguiente turing machine universal:
  - $U = \text{"con input } (M,w) \text{ donde } M \text{ es una TM y } w \text{ un input}$
  - Simulamos  $M$  con input  $w$

- Si M entra en algún momento a su estado de aceptación, acepta
- Si M entra en algún momento a su estado de rechazo, rechazamos
- ❖ Queremos saber **¿ $A_{TM}$  es Turing reconocible?**
  - Podemos ver que si:
    - M reconoce el lenguaje ATM entonces U lo reconoce
    - M rechaza el lenguaje ATM entonces U lo rechaza
    - Pero si M loopea, U también lo hace ...

### Problema de la parada (Halting Problem)

¿Existe una TM que tome por parámetro cualquier TM y diga si la misma es decidible? (recordemos que una TM es equivalente a un lenguaje)

- ❖ Sea  $A_{TM} = \{ \langle M, w \rangle \mid M \text{ es una TM y } M \text{ acepta } w \}$
- ❖ Asumimos que  $A_{TM}$  es decidible y esperamos obtener una contradicción.

### Halting Problem: proof

- ❖ Suponemos H un TM decididor de  $A_{TM}$
- ❖  $H(\langle M, w \rangle) = \{ \text{acepta si } M \text{ acepta } w \text{ y rechaza si } M \text{ no acepta } w \}$
- ❖ Construimos D una TM que usa H como subrutina
  - D = con input (M), con M es una TM
  - Ejecutar H con la entrada  $\langle M, \langle M \rangle \rangle$
  - Responder lo opuesto que indique H
- ❖ En resumen,  $D(\langle M \rangle) = \{ \text{acepta si } M \text{ no acepta } \langle M \rangle \text{ y rechaza si } M \text{ acepta } \langle M \rangle \}$
- ❖ ¿Qué pasa si D se ejecuta con su propia descripción  $\langle D \rangle$  como entrada?
  - $D(\langle D \rangle) = \{ \text{acepta si } D \text{ no acepta } \langle D \rangle \text{ y rechaza si } D \text{ acepta } \langle D \rangle \}$
  - No importa que haga D, está forzado a hacer lo contrario. LO QUE ES UNA CONTRADICCIÓN
  - Por lo tanto → No puede existir ni D ni H

### Post Correspondence Problem (PSP)

- ❖ Se cuenta con un conjunto de piezas tipo dominó, cada una de ellas contiene 2 strings, uno por lado.
- ❖ Encontrar una secuencia de dominos (con repetición permitida) tal que el string leído de un extremo superior sea igual al leído en el extremo inferior.
- ❖ Llamaremos match a esta lista que cumple con el requisito
- ❖ Ejemplo:

Sea el siguiente set de dominos

b	a	ca	abc
ca	ab	a	c

El siguiente corresponde a un match

a	b	ca	a	abc	→	abcaaaabc
ab	ca	a	ab	c	→	abcaaaabc

- ❖ Existen
  - Conjuntos de dominos en el que no importa la combinación o longitud no es posible conformar un match
  - Conjuntos de dominos en el que para lograr el match debemos conformar un secuencia de gran cantidad de dominos

- ❖ Estos casos impiden conocer si el proceso está en un loop, probando cada vez secuencias más largas con un set donde no es posible o simplemente aún no hayo el match
- ❖ **PSP corresponde a un lenguaje no turing decidable**

## LENGUAJES TURING NO RECONOCIBLES

- ❖ Existen
  - Infinitos lenguajes a reconocer
  - Infinitas maquinas de Turing que se pueden generar
- ❖ Sabemos que ciertos lenguajes no son decidibles... pero es posible que algún lenguaje sea no reconocible?
- ❖ **Tamaños de los infinitos - Georg Cantor:**
  - Cantor fue el matemático responsable de fundar la teoría de conjuntos
  - Un conjunto de elementos tiene una cardinalidad: una cantidad de elementos
  - Los conjuntos finitos se pueden “contar” algunos conjuntos infinitos también se pueden contar.

### Correspondencia (biyectiva)

- ❖ Sean A y B dos conjuntos,  $F: A \rightarrow B$  función que mapea un elemento de A a uno de B.
- ❖ Diremos que es una correspondencia (biyectiva) si todos los elementos del conjunto de A tienen una imagen distinta en el conjunto B, y a cada elemento del conjunto de B le corresponde un elemento del conjunto de A.

### Correspondencia entre dos conjuntos finitos

- ❖ 2 sets infinitos tienen el mismo tamaño si se puede establecer entre ellos una correspondencia biyectiva
- ❖ Ejemplo:

Sea N el conjuntos de los números naturales  
Sea E el conjunto de los números pares

Podemos establecer las correspondencia  $f(n)=2N$

Por lo tanto  $\text{Size}(N) = \text{Size}(E)$

N	E
1	2
2	4
3	6
4	8
...	...
n	2n

### Conjuntos contables

- ❖ Un conjunto A es contable si es finito o si su tamaño es igual al conjuntos de los números naturales
- ❖ Ejemplos: Pares, impares, primos, compuestos

### Números racionales

- ❖ ¿Son los números racionales contables?
- ❖  $Q = \{m/n \mid m, n \in \mathbb{N}\}$
- ❖ Los conjuntos infinitos antes analizados
  - Se “intuían” de menor tamaño
  - Si tomo el valor n entero, existían menos elementos en esos conjuntos menores a n que en el conjunto de los enteros
  - Ejemplo: hay 100 números enteros con  $n=100$ , y solo 50 pares
- ❖ Con los números racionales no es el caso
- ❖ ¿Cómo podemos hacer la correspondencia?... iniciemos con un caso más sencillo

## Números enteros

- ❖ Los números enteros son
  - Contiene a los números naturales, sus opuestos y al cero
  - $\mathbb{Z} = \{\dots, -3, -2, -1, 0, 1, 2, 3 \dots\}$
- ❖ Los números enteros son contables
- ❖ Podemos realizar la correspondencia de la siguiente forma:  
 $1 \rightarrow 0, 2 \rightarrow 1, 3 \rightarrow -1, 4 \rightarrow 2, 5 \rightarrow -2, 6 \rightarrow 3, 7 \rightarrow -3, \dots$

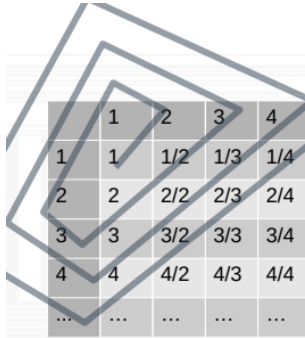
## Volvemos a los racionales

- ❖ Podemos representar a todos los racionales  $Q = \{m/n \mid m, n \in \mathbb{N}\}$  utilizando la siguiente tabla:

Utilizando un recorrido tipo espiral

Podemos listar los números y realizar la correspondencia con los naturales 1, 2,  $1/2, 3, 2/2, 1/3, 4, 3/2, 2/3, \dots$

Por lo tanto, los números racionales son contables



	1	2	3	4	...	n
1	1	$1/2$	$1/3$	$1/4$	...	$1/n$
2	2	$2/2$	$2/3$	$2/4$	...	$2/n$
3	3	$3/2$	$3/3$	$3/4$	...	$3/n$
4	4	$4/2$	$4/3$	$4/4$	...	$4/n$
...	...	...	...	...	...	...
m	m	$m/2$	$m/3$	$m/4$	...	$m/n$

## Números reales

¿Son los números reales contables?  
Es decir, ¿podemos realizar una correspondencia con los números naturales?

Supongamos por unos instantes que sí, entonces puedo hacer una tabla 1 a 1 entre los elementos de ambos conjuntos

¿Están todos los números reales en la tabla?  
Demostremos que siempre existirán números no listados (infinitos!)

1	1	0	0	0	1	0	1	4	...
2	3	1	4	1	5	9	2	6	...
3	0	4	7	1	8	8	8	8	...
4	1	6	3	4	5	0	1	3	...
5	2	7	1	8	2	8	1	8	...
6	7	5	6	7	3	3	2	1	...
7	8	4	2	1	9	8	5	2	...
8	9	5	0	0	0	0	0	0	...
9	5	1	2	5	0	0	0	5	...
...	...	...	...	...	...	...	...	...	...

## Números reales - Diagonal

Vamos a generar un número que no se encuentra en la tabla

Para eso vamos a pasar por la diagonal de la tabla eligiendo en cada "celda" un número diferente al encontrado

El nuevo número armado no se encuentra en la tabla

Si estuviese en la i-ésima posición, su i-ésimo símbolo sería diferente!

2	1	1	0	0	0	1	0	1	4	...
4	2	3	1	4	1	5	9	2	6	...
6	3	0	4	7	1	8	8	8	8	...
5	4	1	6	3	4	5	0	1	3	...
3	5	2	7	1	8	2	8	1	8	...
7	6	7	5	6	7	3	3	2	1	...
3	7	8	4	2	1	9	8	5	2	...
2	8	9	5	0	0	0	0	0	0	...
...	9	5	1	2	5	0	0	0	5	...
...	...	...	...	...	...	...	...	...	...	...

**Por lo tanto, los números reales no son contables**

## Conjunto de secuencias binarias infinitas

Una secuencia binaria infinita es una secuencia de 1 y 0 sin final:

1010111111000011110010101010...

Podemos usar el mismo criterio de la diagonal para mostrar que este conjunto es incontable

El conjunto de secuencias binarias infinitas no es contable

0	1	1	0	0	0	1	0	1	1	...
0	2	0	1	0	1	0	1	1	0	...
1	3	0	1	0	1	1	1	1	1	...
1	4	0	0	0	0	0	0	0	0	...
0	5	1	1	1	1	1	1	1	1	...
1	6	1	0	1	0	1	0	1	0	...
1	7	0	1	0	1	0	1	0	1	...
1	8	1	1	1	1	0	0	0	0	...
...	9	0	0	0	0	1	1	1	1	...
...	...	...	...	...	...	...	...	...	...	...

## Conjunto de todas las TM

¿Son el conjunto de todas la TM contables o no?

- ❖ Cada TM puede descripto por un String en un alfabeto y ser utilizado por una Universal Turing Machine como input.
- ❖ El alfabeto tiene “n” número finito de símbolos. Por ejemplo, 256 símbolos (nuestro “cotidiano” byte)
- ❖ En un string de longitud:
  - $1 \rightarrow$  tengo n posibles TM
  - $k \rightarrow n^k$  posibles TM
- ❖ Algunos de esos Strings representan TM válidas (y otras no)
- ❖ Cualquier TM que puede representarse con el alfabeto con una longitud de K estará en el string  $n^k$ .
- ❖ Puedo hacer una correspondencia con los números naturales
  - Primeros los string de longitud 1 (son n),
  - Luego los de longitud 2 ... hasta longitud  $\infty$
- ❖ Por lo tanto, el conjunto de las TM es contable

## Conjunto de todos los lenguajes

- ❖ Sean  $\Sigma$  un alfabeto y  $\Sigma^*$  el conjunto de todos los posibles strings sobre el alfabeto  $\Sigma$
- ❖ Podemos ver que
  - $\Sigma^*$  es contable
  - Primero el string vacío, luego los string de longitud 1, longitud 2, ...
- ❖ Por ejemplo si  $\Sigma=\{0,1\}$ , entonces  $\Sigma^*=\{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots\}$

## Continuo...

- ❖ Sean
  - $\Sigma$  un alfabeto
  - $\Sigma^*$  el conjunto de todos los posibles strings sobre el alfabeto  $\Sigma$
  - L el conjunto de todos los lenguajes sobre el alfabeto  $\Sigma$
  - $A \in L$  un lenguaje en el alfabeto  $\Sigma$
- ❖ Podemos representar A como una secuencia de 0 y 1 tal que el bit i esta en 1 si el i-esimo string de  $\Sigma^*$  pertenece al lenguaje A
- ❖ Ejemplo
  - $A = \{\text{todos los string que terminan en } 0\}$
  - $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \dots\}$
  - $X_A = \{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, \dots\}$
- ❖ Se puede ver que  $X_A$  es una secuencia binaria infinito

- ❖ En L, existen infinitos lenguajes
  - Podemos representar cada uno de ellos con una secuencia binaria infinita.
  - Existe una correspondencia entre el conjunto de todos los lenguajes y el conjunto de todas las secuencias binarias infinitas
- ❖ Por lo tanto el conjunto de todos los lenguajes no es contable

### **Lenguajes No reconocibles por una TM**

- ❖ Como:
  - el conjunto de todas las TM es contable
  - el conjunto de todos los posibles lenguajes es incontable
  - Y un lenguaje es reconocible si una TM puede reconocerlo
- ❖ Entonces **existen lenguajes no reconocibles por una Máquina de Turing**