Máquinas de Turing
(*Turing Machines*)

Alan Reyes-Figueroa

Teoría de la Computación **(Aula 21) 14.octubre.2024**

Motivación

Definición

Ejemplos

Funciones computables

Motivación

- Hasta ahora hemos visto clases de lenguajes relativamente simples.
- Lo que vamos a ver ahora es preguntarnos ¿qué lenguajes pueden definirse por cualquier equipo computacional?
- Vamos a ver qué pueden hacer las computadoras y los problemas que no pueden resolver, a los que llamaremos indecidibles.

Motivación

- Trataremos de determinar a las *funciones efectivamente calculables*.
  Esto es, aquellas funciones matemáticas para las cuales existe un procedimiento efectivo o mecánico que calcula sus imágenes.
- Función calculable = existe un algoritmo para calcular f.

Máquinas de Turing

- Si existe un algoritmo para realizar una tarea argumentamos que se puede construir una máquina hipotética y determinística que realice dicha tarea.
  - Estas son llamadas **Máquinas de Turing**.

Publicadas en
“On Computable Numbers, with an applicaton to the Entscheidungsproblem” (1936).

Ejemplo

- Podemos pensar en un programa de computadora que imprima “hola!” cuando encuentre un entero positivo $n > 2$, que cumpla: $x^n + y^n = z^n$, para x, y, z enteros positivos.
- La solución entera de la ecuación de arriba se conoce como el último teorema de Fermat, (llevó 300 años resolver).
- El poder analizar cualquier programa de computadora y decidir si va a imprimir un letrero como “hola” es en general indecidible.

Historia

- El propósito de la teoría de indecibilidad no es sólo establecer cuáles problemas son indecidibles, sino también dar una guía sobre qué es lo que se puede hacer o no con programación.
- También tiene que ver con cuáles problemas, que aunque sean decidibles, son intratables.
- A finales del siglo XIX y principios del XX, David Hilbert lanzó la pregunta abierta, si era posible encontrar un algoritmo que determinara el valor de verdad de una formula en lógica de primer orden aplicada a los enteros (*Entscheidungsproblem*).

Historia

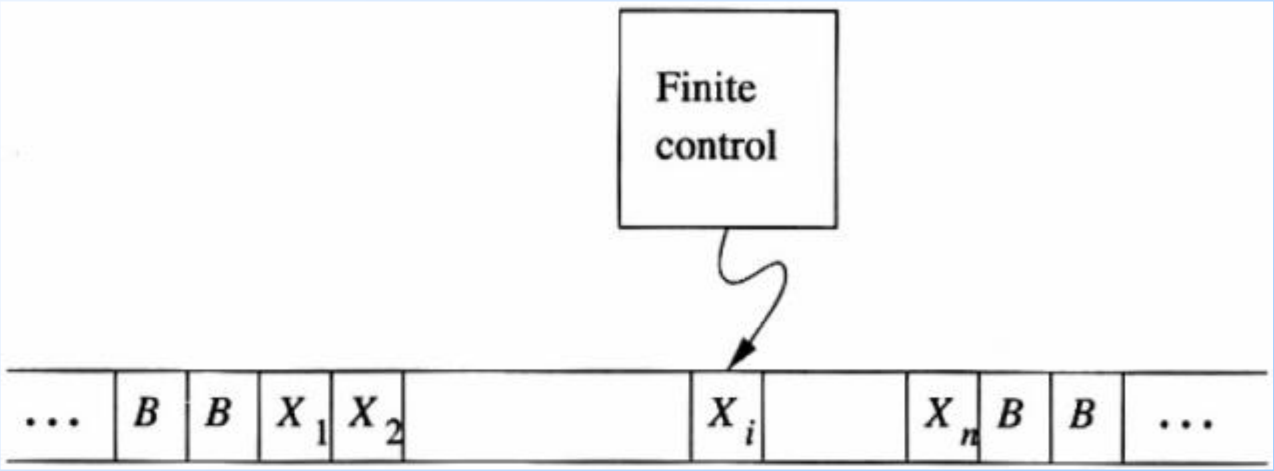
- En 1931, Kurt Gödel probó su Teorema de incompletitud para probar que no se puede construir dicho algoritmo.
- En 1936, Alan Turing publicó su máquina de Turing como un modelo para cualquier tipo de computación (aunque todavía no existían las computadoras).
- La Hipótesis de Church o la tesis de Church-Turing dice que lo que las máquinas de Turing (y en general, las computadoras modernas) pueden calcular son las funciones **recursivamente enumerables**.

Máquinas de Turing

- Una máquina de Turing consiste de un control finito que puede estar en cualquier estado de un conjunto finito de estados.
- Se tiene una cinta dividida en celdas, cada celda con un símbolo.
- Inicialmente, la entrada (cadena finita de símbolos del alfabeto) se coloca en la cinta, el resto de las celdas tienen el símbolo especial vacío (*blank*).
- La cabeza de la cinta esta siempre sobre una celda y al principio está sobre la celda más a la izquierda con el primer símbolo de la cadena de entrada.

Máquinas de Turing

- Un movimiento o transición puede cambiar de estado (o quedarse en el estado actual), escribir un símbolo (reemplazando el símbolo que existía o dejando el mismo) y mover la cabeza a la izquierda o derecha.



Enteros, cadenas y más …

- Los tipos de datos se han vuelto muy importantes como herramienta de programación.
- A otro nivel, sólo hay un tipo de datos, que puede considerar como números enteros o cadenas.
- Punto clave: las cadenas que son programas son sólo otra forma de pensar sobre el mismo tipo de datos.

Ejemplo: Texto

- Las cadenas de caracteres ASCII o Unicode pueden considerarse cadenas binarias, con 8 o 16 bits/carácter.
- Las cadenas binarias se pueden considerar como números enteros.
  - "abc" → "97 98 99"
    → "01100001 01100010 01100011"
    → N
- Podemos hablar de "la i-ésima cadena".

Cadenas binarias a enteros

- Hay un pequeño *glitch*:
  - Si se piensa simplemente en números enteros binarios, entonces las cadenas como 101, 0101, 00101... todas parecen ser "la quinta cadena".
- Arreglamos estos añadiendo un prefijo “1” a la cadena antes de convertirla a un entero.
  - Así, 1101, 10101, y 100101 son las cadenas 13^{th}, 21^{st}, y 37^{th}, respectivamente.

Ejemplo: Imágenes

- Representar una imagen (digamos) en formato JPG.
- El archivo JPG es una cadena ASCII.

  1 01000110 01101000 01110110 … 0100001
- Convertimos a una cadena binaria.
- Convertimos la cadena binaria a un entero k.
- Podemos hablar ahora de "la k-ésima imagen."

Ejemplo: Demostraciones

- Una prueba formal es una secuencia lógica de expresiones, cada una de las cuales se sigue de las anteriores.
- Codificamos las expresiones matemáticas y del lenguaje usando caracteres Unicode (ASCII, UTF-8, …).
- Convertimos las expresiones a una cadena binaria, y luego a un entero.

Demostraciones

- Pero una prueba es una secuencia de expresiones, por lo que necesitamos una forma de separarlas.
  - Decir dónde termina una, y comienza la siguiente.
- Además, necesitamos indicar qué expresiones se dan.

Demostraciones

- Forma rápida de introducir nuevos símbolos en cadenas binarias:
  1. Dada una cadena binaria, anteceder cada bit por 0.
     - Ejemplo: 101 se vuelve 010001.
  2. Usar cadenas de dos o más 1's como símbolos especiales (separadores).
     - Ejemplo: 111 = "inicio: siguiente expresión es"; 11 = "fin de la expresión."

Ejemplo: Codificando Pruebas

$\boxed{111}\boxed{010001}\boxed{11}\boxed{111}\boxed{000000101}\boxed{11}0101\ldots$

Sigue una expresión dada

Fin de expresión

Expresión

Fin

Sigue una expresión dada

Una expresión

Note que este 1 no es parte de un "fin"

Ejemplo: Programas

- Programas son sólo otro tipo de datos:
- Representar un programa en ASCII.
- Convertir a cadena binaria, y luego a un entero.
- Así, hace sentido hablar de "el i-ésimo programa."
- No hay muchos programas (cantidad enumerable).

Conjuntos Finitos

- Intuitivamente, un *conjunto finito* es un conjunto para el cual hay un entero particular que "cuenta a sus elementos".
- Ejemplo: {a, b, c} es un conjunto finito; su *cardinalidad* es 3.
- Es imposible encontrar un mapeo 1-1 entre un conjunto finito y cualquiera de sus subconjuntos propios.

Conjuntos Infinitos

- Formalmente, un *conjunto infinito* es un conjunto para el cual existe un mapa 1-1 entre él y un subconjunto propio de él mismo.
- Ejemplo: Enteros positivos $Z^+ = \{1, 2, 3, \ldots\}$ es un conjunto infinito.
  - Hay una correspondencia 1-1 entre Z^+ y su subconjunto propiode los números pares $2Z^+$:
  - $1 \rightarrow 2,\ 2 \rightarrow 4,\ 3 \rightarrow 6, \ldots \qquad n \rightarrow 2n$

Conjuntos Enumerables

- Un *conjunto enumerable* es un conjunto con una correspondencia 1-1 con los enteros positivos Z^+.
  - Todos los conjuntos enumerables son infinitos.
- Ejemplo: Los enteros Z.
  - $0 \rightarrow 1$; $-n \rightarrow 2n$; $+n \rightarrow 2n+1$.
  - Este orden sería: 0, -1, 1, -2, 2, -3, 3, …
- Ejemplo: el conjunto de enteros binarios
- Ejemplo : el conjunto de programas Java.

Ejemplo: Pares de Enteros

- Ordenar los pares de enteros positivos, primero por suma, luego por 1a componente:
- [1,1], [2,1], [1,2], [3,1], [2,2], [1,3], [4,1], [3,2],..., [1,4], [5,1],...
- Ejercicio!: Determinar una función f(i,j) tal que el par [i,j] corresponde al entero f(i,j) en este orden.

Enumeraciones

- Una *enumeración* de un conjunto S es una correspondencia 1-1 entre S y el conjunto de enteros positivos Z^+.
- Entonces…
- Hemos visto enumeraciones para las cadenas, programas, pruebas, y pares de enteros.

¿Cuántos Lenguajes?

- Son los lenguajes sobre {0,1}* enumerables?
- No; aquí hay una prueba.
- Suponga que podemos enumerar todos los lenguajes sobre {0,1}* y hablar de "el i-ésimo lenguaje."
- Considere el siguiente lenguaje
  L = {w: w es la i-ésima cadena binaria y w no está en el i-ésimo lenguaje}.

Prueba –

- L es un lenguaje sobre {0,1}*.
- Entonces, L debe de ser el j-ésimo lenguaje, para algún j particular.
- Sea x la j-ésima cadena.
- ¿Está x en L?
  - Caso afirmativo, x no están en L, (def de L).
  - Caso negativo, x está en L (def de L).

L_j

x

Obs!: L = {w: w es la i-th cadena binaria y w no está en el i-th lenguaje}.

j-th

Argumento de Diagonalización

Cadenas

Lenguajes

	1	2	3	4	5	...
1	1	0	1	1	0	...
2		1				
3			0			
4				0		
5					1	
...						...

Argumento de Diagonalización

Voltear cada entrada diagonal

Cadenas

Lenguajes

	1	2	3	4	5	…
1	0	0	1	1	0	…
2		0				
3			1			
4				1		
5					0	
…						…

Esta diagonal no puede ser una fila – ya que es diferente en la entrada i con la fila i, Para todo i.

Prueba – Conclusión

- Tenemos una contradicción: x está en L o no está en L, de modo que el supuesto inicial (de que hay una enumeración de todos los lenguajes sobre {0,1}*) es falsa.
- Obs!: Muy malo!!: hay más lenguajes que programas.
- E.g., hay languajes sin un algoritmo de pertenencia.

Argumentos Húngaros

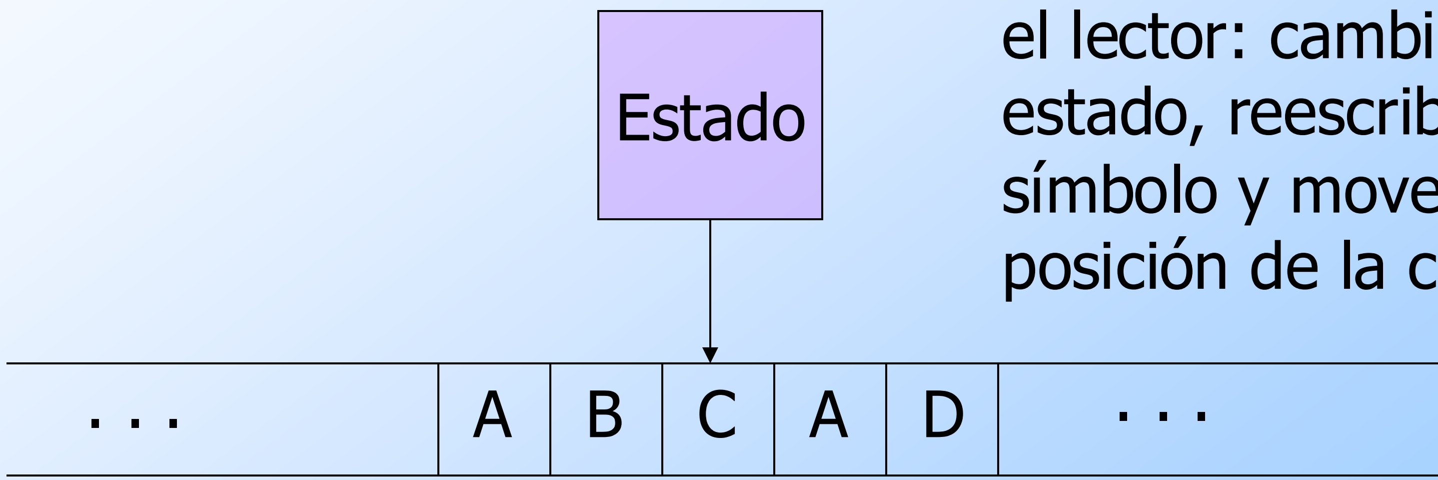
- Hemos demostrado la existencia de un lenguaje sin algoritmo para probar la membresía, pero no tenemos forma de exhibir un lenguaje particular con esa propiedad.
- Una prueba que consiste en contar las cosas que fallan y ver que son menos que todas las cosas se llama *argumento húngaro* (argumento de conteo).

Máquinas de Turing

- El propósito de la teoría de las máquinas de Turing es probar que ciertos lenguajes específicos no tienen algoritmo.
- Comenzar con un lenguaje sobre las propias máquinas de Turing.
- Las reducciones se utilizan para demostrar que las preguntas más comunes son indecidibles.

Máquinas de Turing

Acción: basada en el estado y el símbolo de la cinta bajo el lector: cambiar de estado, reescribir el símbolo y moverse una posición de la cinta.



Una cinta infinita con espacios (cuadrados) y símbolos elegidos de un alfabeto finito.

¿Por qué Máquinas de Turing?

- ¿Por qué no trabajar con programas (de C o de Python) o algo parecido?
- Respuesta: Podríamos, pero es más fácil probar propiedades sobre las Máquinas de Turing, porque son simples.
  - Y sí, son igual de poderosas que cualquier computador.
    - Además, tienen memoria infinita.

¿Por qué no usar Autómatas?

- En principio, podríamos usarlos, pero no es constructivo.
- Los modelos de programación no se pueden construir bajo memoria limitada.
  - Podríamos “comprar más memoria”.
- Los autómata finitos son vitales al nivel base (verificación).
  Pero no a nivel generalización.

Máquinas de Turing

- Una Máquina de Turing se describe por:
  1. Un conjunto finito de *estados* (Q).
  2. Un *alfabeto de entrada* (Σ).
  3. Un *alfabeto de cinta* (Γ; contiene a Σ).
  4. Una *función de transición* (δ).
  5. Un *estado inicial* ($q_0 \in Q$).
  6. Un *símbolo blanco* ($B \in \Gamma - \Sigma$).
     - Toda la cinta, excepto el input está en blanco.
  7. Un conjunto de *estados finales* ($F \subseteq Q$).

Convenciones

- a, b, c, … son símbolos input.
- …, X, Y, Z son símbolos cinta.
- …, w, x, y, z son cadenas de símbolos input.
- α, β,… son cadenas de símbolos cinta.

Función de Transición

- $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times D$ toma dos argumentos:
  1. Un estado, q en Q.
  2. Un símbolo de cinta Z en Γ.
- δ(q, Z) está indefinido, o es una tripla de la forma (p, Y, D), con:
  - $p \in Q$ un estado,
  - $Y \in \Gamma$ un símbolo de cinta,
  - $D \in \{L, R\}$ una *dirección*, (Left ó Right).

Acciones de una MT

- Si $\delta(q, Z) = (p, Y, D)$ entonces, en el estado q, leyendo el símbolo Z (en el lector de la cinta), la máquina de Turing hace lo siguiente:
  1. Cambia al estado p.
  2. Reemplaza Z por Y en la cinta.
  3. Mueve una posición el lector de la cinta, en la dirección D.
     - D = L: se mueve a la izquierda; D = R; a la derecha.

Ejemplo: Máquina de Turing

- Esta máquina de Turing lee el input hacia la derecha, buscando un 1.
- Si encuentra uno, lo modifica a 0, va al estado final f, y termina.
- Si alcanza un símbolo blanco, lo cambia a 1 y se mueve hacia la izquierda.

Ejemplo: Máquina de Turing

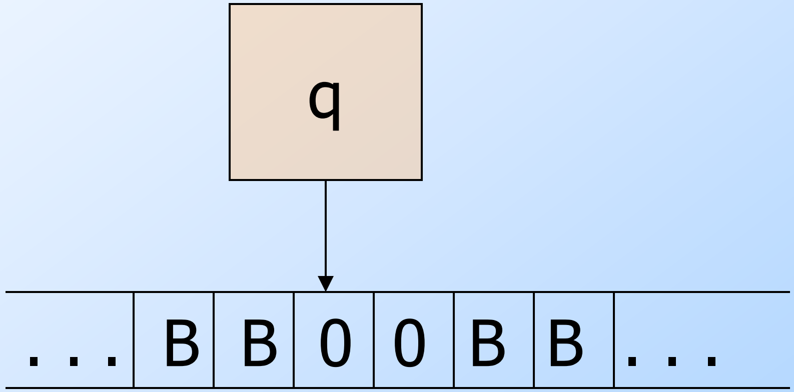
- Estados: $Q = \{q$ (inicial), f (final)$\}$.
- Símbolos input: $\Sigma = \{0, 1\}$.
- Símbolos cinta: $\Gamma = \{0, 1, B\}$.
- Función de Transición:
  - $\delta(q, 0) = (q, 0, R)$.
  - $\delta(q, 1) = (f, 0, R)$.
  - $\delta(q, B) = (q, 1, L)$.

Simulación

$$\delta(q, 0) = (q, 0, R)$$

$$\delta(q, 1) = (f, 0, R)$$

$$\delta(q, B) = (q, 1, L)$$

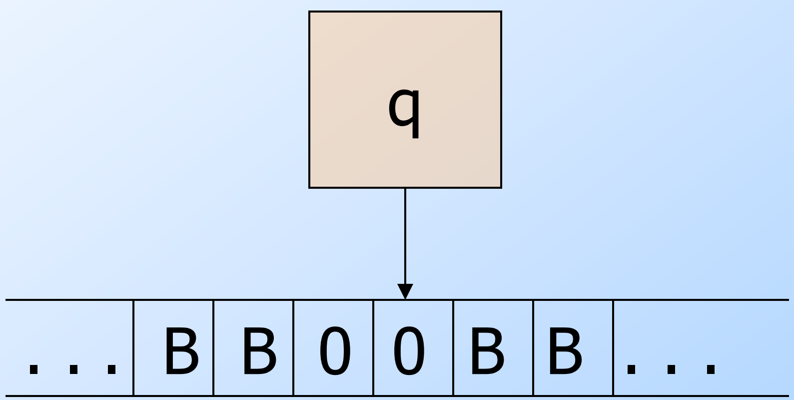


Simulación

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

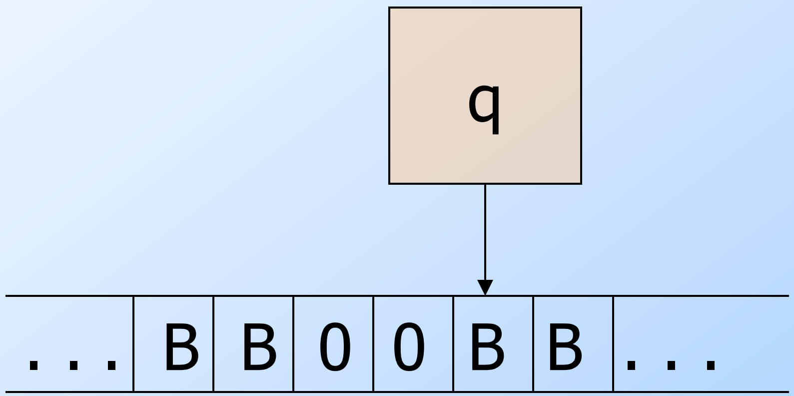


Simulación

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

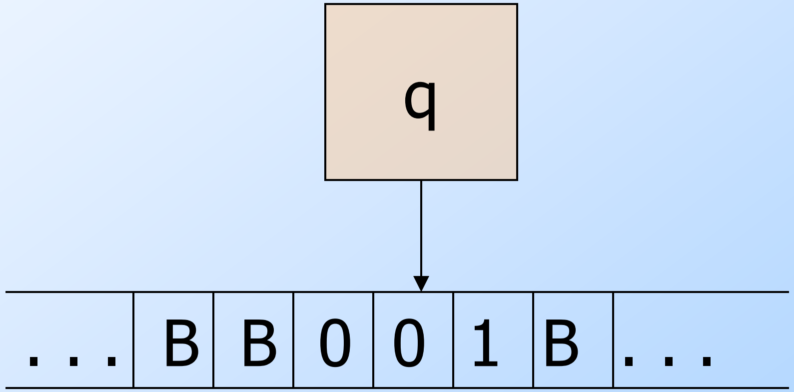


Simulación

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

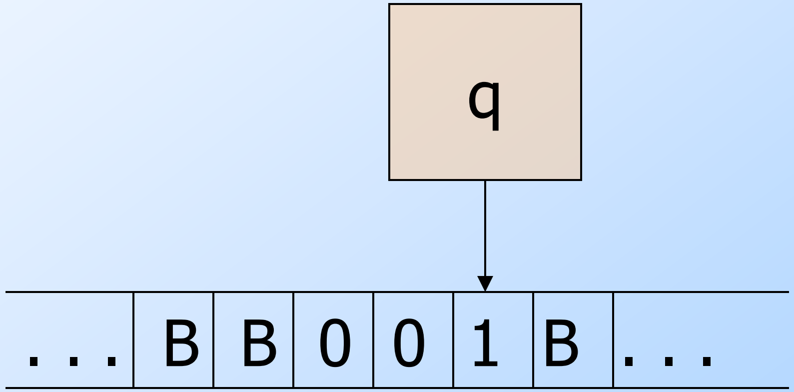


Simulación

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

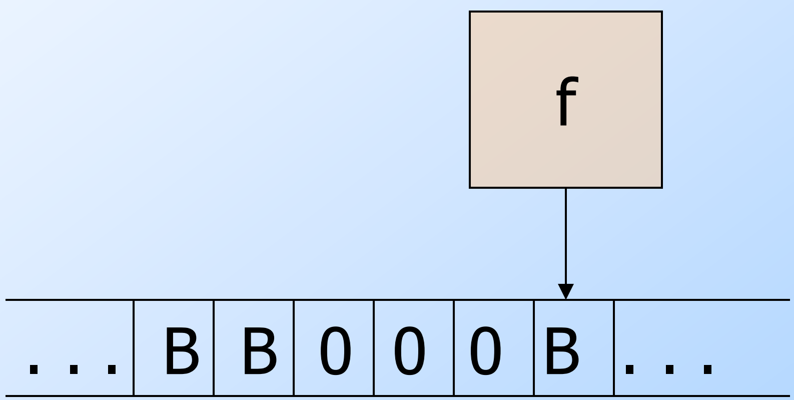


Simulación

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$



No ha más movidas posibles. La máquina para y acepta.

Descripciones Instantáneas

- Como la cinta es infinita, se representan solo los símbolos entre los B's (a veces se pueden incluir algunos B's) y
- se incluye un símbolo especial para indicar la posición del lector cinta.
- Por ejemplo:

  $$\alpha \, q \, \beta \;=\; X_1 X_2 \ldots X_{i-1} \, q \, X_i X_{i+1} \ldots X_n$$

  representa una descripción instantánea donde:
  - q es el estado de la maquina de Turing
  - la cabeza de la cinta está viendo al i-ésimo símbolo a la izquierda
  - $X_1 X_2 \ldots X_n$ es el pedazo de cinta entre los símbolos más a la izquierda y más a la derecha que no son vacíos.

Descripciones Instantáneas

- Usamos la misma notacion de descripción instantánea que en los autómatas de pila: $\vdash$ y $\vdash^*$.
  - $\vdash$ "se convierte en un movimiento",
  - $\vdash^*$ "se convierte en cero o más movim."
- Ejemplo: Los movimientos de la MT anterior son
  $q00 \vdash 0q0 \vdash 00q \vdash 0q01 \vdash 00q1 \vdash 000q$

Definición Formal de $\vdash$

1. Si $\delta(q, Z) = (p, Y, R)$, entonces escribimos:

$$\alpha qZ\beta \vdash \alpha Yp\beta$$

   Si Z es el símbolo blanco B, entonces

$$\alpha q \vdash \alpha Yp$$

1. Si $\delta(q, Z) = (p, Y, L)$, entonces escribimos:
   - Para cualquier X, $\alpha XqZ\beta \vdash \alpha pXY\beta$
   - Además, $qZ\beta \vdash pBY\beta$