



Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Tema 3

Modelos de Computación y

Funciones Computables

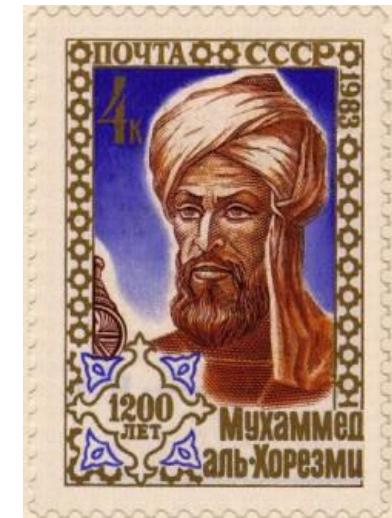
Departamento de Informática
Ciencia de la Computación e Inteligencia Artificial
Universidad de Oviedo

Objetivos

- Entender el concepto de Algoritmo y sus características
- Comprender el concepto de modelo de computación.
- Construir Programas While y Máquinas de Turing para computar funciones sencillas
- Utilizar correctamente la Tesis de Church

La Palabra Algoritmo

- Abu Abdallah Muḥammad ibn Mūsā al-Jwārizmī (Abu Yāffar) conocido generalmente como ***al-Juarismi***
- Nació alrededor del 780 DC en Jorezm, al sur del mar de Aral (hoy Jiva, Uzbekistán) y falleció en Bagdad hacia el 850 DC.
- Astronomía, Geografía y Álgebra
- Escribió en 825: “On Calculation with Arabic Numerals”, traducido más tarde como “Algoritmi de numero Indorum”.
- Kitab al-jabr wa'l-muqabala: “El libro de restaurar e igualar” o “El arte de resolver ecuaciones”. (**Álgebra**)



Concepto de Algoritmo

- Una persona de nuestro entorno no sabe utilizar un aparato para grabar un programa de televisión y nos pide ayuda. Con el objetivo de que no se le olvide, le ponemos por escrito un conjunto finito de instrucciones del siguiente tipo
 - 1. Comprueba los botones del aparato hasta que veas uno que pone “on”
 - 2. pulsa el botón visto anteriormente
 - 3. selecciona el canal donde se emite el programa deseado
 - 4. Si ya comenzó el programa deseado, entonces pulsa el botón “rec”. En otro caso...

.....

Concepto de Algoritmo

- Analizando las instrucciones anteriores:
 - a) La instrucción 1 requiere realizar una acción repetidamente hasta que se dé una condición. Si no hubiese un botón “on” no saldríamos de ella (**bucle**)
 - b) La instrucción 3 tiene otro nivel de detalle, como si se requiriesen varias instrucciones más simples para ejecutarla (**macro**)
 - c) La instrucción 4 plantea dos alternativas con diferentes acciones a realizar. (**if-then-else**)

Concepto de Algoritmo

- Con el conjunto de instrucciones anterior programaríamos la función:

Grabar (programa_deseado)

- Podríamos sentenciar que hemos construido un algoritmo, utilizando un entorno de programación (lenguaje natural) y cuya función asociada es la de grabar un programa deseado.

Concepto de Algoritmo

- Características:
 1. **Finitud:** El conjunto de instrucciones es finito
 2. Possible existencia de **Bucle:** Es lo único que puede hacer que la ejecución de las instrucciones nunca finalice. Así pues la ejecución puede ser infinita.
 3. **Función:** Hay una función asociada al algoritmo.

Modelo de Computación

- Un **modelo de computación** es un modelo matemático que permite caracterizar formalmente la resolvibilidad algorítmica de problemas complejos.
- Define un conjunto de operaciones permisibles y sus respectivos costes.
- Permite analizar los recursos computacionales requeridos (p.e. tiempo de ejecución o espacio de memoria) para resolver un problema y discutir las limitaciones de los algoritmos.

Modelo de Computación

- Contexto para hacer computación. Características principales:
 - Operaciones básicas
 - Reglas de combinación
- Aunque existen muchos modelos de computación diferentes, nosotros veremos los siguientes:
 - **Programas While**
 - **Máquinas de Turing**

Modelo de Computación

- Dos ideas de combinación:
 1. La ejecución repetida de una acción algorítmica es asimismo algorítmico siempre que también lo sea el test de parada (Recuérdese el bucle)
 2. La ejecución secuencial de un número finito de acciones algorítmicas es algorítmico

Contenidos

- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

Modelo de los Programas While (PW)

- **Dominio:** Conjunto de los **números naturales**
- **Nombres de Variables:** Cadenas finitas de letras y números que comienzan con mayúscula
 - *Ejemplos:* $X_1, X_2, TEMP, LIST1$
- **Símbolos de operaciones:**
 - Denotan operaciones básicas
 - **succ** (función sucesor), **pred** (función predecesor), **0** (función constante igual a 0)
- **Símbolo de relación:** “**#**” (“distinto de” para comparar los valores de dos variables)
- **Símbolos de programación:** **:=** (asignación), **;** (punto y coma), **begin**, **end**, **while**, **do**, **(,)**

PW: Sentencias

■ Sentencias básicas (asignación)

- $X := 0$
 - $X := \text{succ}(Y)$
 - $X := \text{pred}(Y)$
- ← **OJO:** $\text{pred}(0)=0$

■ Sentencias while:

- $\text{while } X \neq Y \text{ do } \delta$
- siendo δ una sentencia cualquiera
- $X \neq Y$ es el test, y δ es el cuerpo

■ Sentencia compuesta:

- $\text{begin } \delta_1; \delta_2; \dots; \delta_n \text{ end}$
- siendo δ_i sentencias arbitrarias y $n \geq 0$

PW: Sentencias

Un **programa while** es una sentencia compuesta que elegimos identificar como un programa, pero que puede utilizarse como sentencia compuesta en otro programa mas largo

PW: Ejemplo

```
begin
  Z := 0;
  U := 0;
  while U ≠ Y do
    begin
      V := 0;
      while V ≠ X do
        begin
          V := succ(V);
          Z := succ(Z)
        end
      U := succ(U)
    end
  end
end
```

Sentencia básica

Sentencia compuesta

Sentencia while

PW: Macros

- **Macro sentencia:** Etiqueta dada a un programa while para que éste pueda ser utilizado como parte de otros programas while
- **Ejemplos:**

$$Z := X + Y$$

$$Z := X$$

$$Z := X \div Y$$

$$Z := X^*Y$$

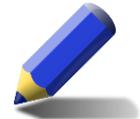
$$Z := X \text{ div } Y$$

$$Z := X \text{ mod } Y$$

$$Z := X^{**}Y$$

$$Z := n; \quad n > 0$$

PW: Macros (Ejemplo)



- ¿Cómo escribir un programa-while P que sume **X** a **Y** y deje el valor en **Z**?

1. Asignar el valor de X a Z

```
begin  
  Z := succ(X);  
  Z := pred(Z)  
end
```

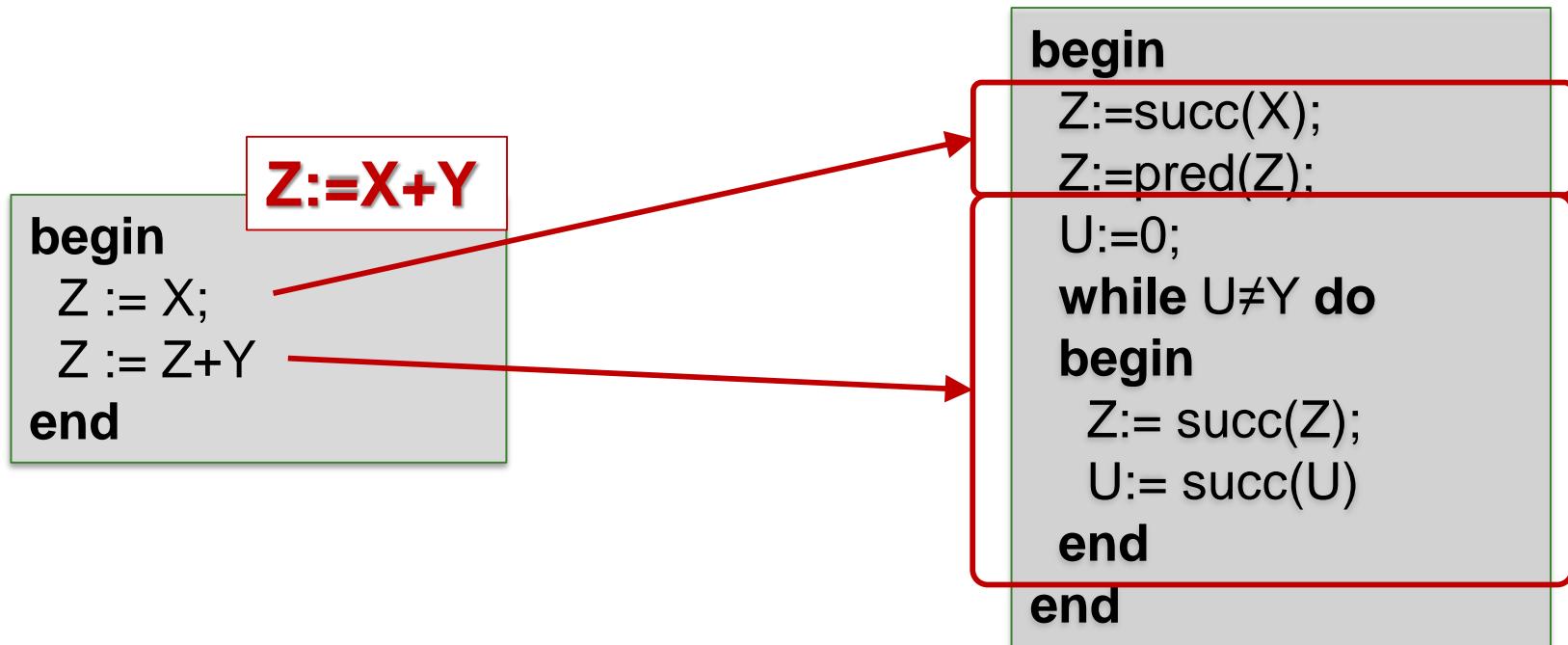
2. Añadir el valor de Y a Z

```
begin  
  U := 0;  
  while U ≠ Y do  
    begin  
      Z := succ(Z);  
      U := succ(U)  
    end  
  end
```

PW: Macros (Ejemplo)



- ¿Cómo escribir un programa-while P que sume **X** a **Y** y deje el valor en **Z**?
- Siempre que usemos la macro sentencia en un programa-while, tenemos que entender que es una etiqueta que encierra el código



PW: Macros (Ejemplo)



- Diseña un programa-while P para la diferencia acotada
Z:=X-Y

$$X - Y = \begin{cases} X - Y & \text{si } X \geq Y \\ 0 & \text{si no} \end{cases}$$

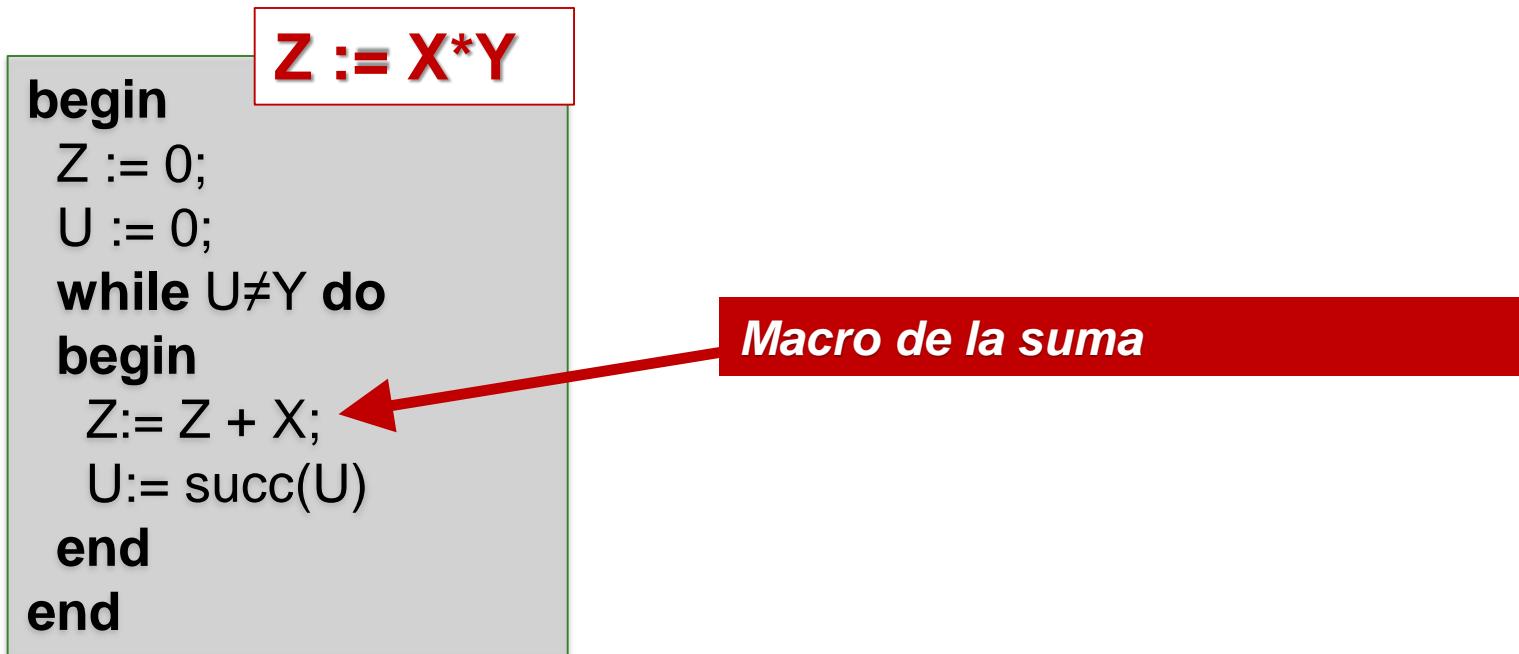
Z := X-Y

```
begin
Z:=succ(X);
Z:=pred(Z);
U:=0;
while U≠Y do
begin
    Z:= pred(Z);
    U:= succ(U)
end
end
```

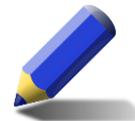
PW: Macros (Ejemplo)



- Construye un programa while para la macro **Z := X * Y**. Está permitido utilizar las macros de la suma y de la asignación



PW: Macros (Ejemplo)



- Construye un programa while para la macro $Z := X * Y$.
No se permite el uso de macros

Z := X*Y

```
begin
  Z := 0;
  U := 0;
  while U≠Y do
    begin
      Z:= Z + X;
      U:= succ(U)
    end
  end
```

Reemplazamos la macro por sus sentencias básicas

Z := X*Y

```
begin
  Z := 0;
  U := 0;
  while U≠Y do
    begin
      V:=0;
      while V≠X do
        begin
          V:=succ(V);
          Z:=succ(Z)
        end
      U:= succ(U)
    end
  end
```

PW: Macros (Ejemplo)



- Construye un programa while que calcule $Z := X \text{ div } Y$. La única macro que se permite utilizar es la de la diferencia acotada

```
begin
  W := succ(X);
  Z := 0;
  U := 0;
  while W ≠ U do
    begin
      Z := succ(Z);
      W := W - Y
    end
    Z:=pred(Z)
  end
```

Z := X div Y

Macro de la diferencia acotada

PW: Macros (Ejemplo)



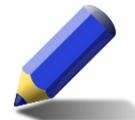
- Construye un programa while que calcule $Z := X \text{ div } Y$. No se permite utilizar macros.

```
begin
  W := succ(X);
  Z := 0;
  U := 0;
  while W ≠ U do
    begin
      Z := succ(Z);
      V:=0;
      while V≠Y do
        begin
          W:=pred(W);
          V:=succ(V)
        end
      end
      Z:=pred(Z)
    end
end
```

$Z := X \text{ div } Y$

*Para ello,
sustituimos la macro
de la diferencia
acotada por
sentencias básicas*

PW: Macros (Ejemplo)



- Construye un programa while que calcule $Z := X \bmod Y$. Se permiten las macros de asignación y diferencia acotada

```
begin
  W := succ(X);
  Z := 0;
  U := 0;
  while W ≠ U do
    begin
      Z := W;
      W := W - Y
    end
    Z:=pred(Z)
  end
```

$Z := X \bmod Y$

Macro de la asignación

Macro de la diferencia acotada

PW: Macro-test

Macro-test: Sentencia de la forma:

while T do δ ,

con δ sentencia arbitraria y T un test diferente de $X \neq Y$.

- Suponemos que en T no hay números naturales
 - (si los hay \rightarrow variables)
- Existe una expresión aritmética E_T en términos de las variables de T y de los operadores $*$, $+$ y \div tal que:

$$E_T = \begin{cases} > 0 & \text{si } T \text{ es verdad} \\ 0 & \text{en otro caso} \end{cases}$$

```
begin
  U := ET;
  V := 0;
  while U ≠ V do
    begin
      δ ;
      U := ET
    end
  end
```

PW: Macro-test

- **¿Cómo construir un T cualquiera?**
- Usando la definición inductiva de Macro-test

Caso base:

$$T = (X < Y) \quad E_T = (Y - X)$$

Caso inductivo:

Sean T_1, T_2
macro-tests

$$T = T_1 \wedge T_2 \quad E_T = E_{T_1} * E_{T_2}$$

$$T = T_1 \vee T_2 \quad E_T = E_{T_1} + E_{T_2}$$

$$T = \neg T_1 \quad E_T = 1 - E_{T_1}$$

PW: Macro-test (Ejemplos)



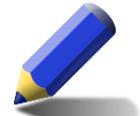
- **Construir E_T para $T = (Z \geq Y) \vee (Z > X)$**

$$\begin{aligned} E_{(Z \geq Y) \vee (Z > X)} &= E_{Z \geq Y} + E_{Z > X} = \\ &= (1 - E_{Z < Y}) + E_{X < Z} = \\ &= (1 - (Y - Z)) + (Z - X) \end{aligned}$$

- **Construir E_T para $T = (X = Y)$**

$$\begin{aligned} E_{X=Y} &= E_{(X \leq Y) \wedge (X \geq Y)} = E_{X \leq Y} * E_{X \geq Y} = \\ &= (1 - E_{Y < X}) * (1 - E_{X < Y}) = \\ &= (1 - (X - Y)) * (1 - (Y - X)) \end{aligned}$$

PW: Macro-test (Ejemplos)



- Deshacer los macro-test:
 - Se permiten las macros de resta, suma y asignación.

```
begin T
  while (X<Y) ∨ (Y>=Z) do
    begin
      X:=succ(X);
      Y:=pred(Y); } δ;
    end
  end
```

1º. Calculamos la E_T

$$\begin{aligned} E_{(X < Y) \vee (Y \geq Z)} &= E_{X < Y} + E_{\neg(Y < Z)} = \\ &= (Y - X) + (1 - E_{Y < Z}) = \\ &= (Y - X) + (1 - (Z - Y)) \end{aligned}$$

```
begin
  U:=Y - X; //  $E_{X < Y}$ 
  V:=1;
  W:=Z - Y; } //  $E_{\neg(Y < Z)}$ 
  V:=V - W;

  U:=U+V; //  $E_{(X < Y) \vee (Y \geq Z)}$ 
  V:=0;
  while U ≠ V do
    begin
      X:=succ(X);
      Y:=pred(Y); } δ;
      U:=Y - X;
      V:=1;
      W:=Z - Y; } Se recalcula  $E_{(X < Y) \vee (Y \geq Z)}$ 
      V:=V - W; } Pues X e Y cambian e influyen en T
    end
  end

  U:=U+V;
  V:=0; → Se compara  $E_{(X < Y) \vee (Y \geq Z)}$  con 0, por eso V:=0
  end
end
```

PW: Sentencias estructuradas

- **Proposición:** Una **sentencia estructurada** de la forma:

if T then
δ;

if T then
δ₁;
else
δ₂;

repeat
δ;
until T

donde **T** es un test y **δ**, **δ₁** y **δ₂** son sentencias, es una
macro sentencia

PW: Sentencias estructuradas

If T then δ

```
begin  
  U:=ET;  
  V:=0;  
  while U≠V do  
    begin  
      δ ;  
      U:=0;  
    end  
  end
```

δ_1 sólo se ejecuta cuando $E_T > 0$ (T cierto)

If T then δ_1 , else δ_2

```
begin  
  U:=ET;  
  V:=1 ÷ ET;  
  W:=0;  
  while U≠W do  
    begin  
      δ1 ;  
      U:=0;  
    end  
  while V≠W do  
    begin  
      δ2 ;  
      V:=0;  
    end  
  end
```

δ_1 sólo se ejecuta cuando $E_T > 0$ (T cierto).
 δ_2 sólo se ejecuta cuando E_T igual a 0 (T falso)

Repeat δ until T

```
begin  
  δ;  
  U:=1 ÷ ET;  
  V:=0;  
  while U≠V do  
    begin  
      δ ;  
      U:=1 ÷ ET;  
    end  
  end
```

δ se ejecuta al menos 1 vez y hasta que T es cierto.

Contenidos

- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

PW: Definiciones

- **Sentencia (Programa) k-variables:** Una **sentencia** δ (en particular un programa while) se dice **k-variables**, si utiliza un subconjunto de las variables $\{X_1, X_2, \dots, X_k\}$
- **Vector estado:** Un **vector estado de la computación** de un programa while k-variables es un vector

$$\hat{a} = (a_1, \dots, a_k) \in \mathbb{N}^k$$

en el que a_i es el contenido de la variable X_i

- Si el programa **tiene k variables**, su **vector estado** de la computación tendrá siempre **k componentes**

PW: Secuencia de computación

- Dado un programa while k-variables P, una **secuencia de computación** de P es una sucesión (que puede ser infinita) de la forma

$$\hat{a}_0 A_1 \hat{a}_1 A_2 \hat{a}_2 \dots$$

donde \hat{a}_i son vectores estado y A_j son instrucciones de P

- Las instrucciones pueden ser sentencias básicas o tests.
- \hat{a}_0 es el **vector estado inicial**
- En caso de ser finita, la secuencia de computación es de la forma

$$\hat{a}_0 A_1 \hat{a}_1 A_2 \hat{a}_2 \dots \hat{a}_{(n-1)} A_n \hat{a}_n$$

siendo n su **longitud** y \hat{a}_n el **vector estado final**.

PW: Ejemplo

- Secuencia de computación de P con vector estado inicial $\hat{a}_0 = (4,2)$

Programa While P

```
begin
  X1 := 0;
  while X1 ≠ X2 do
    X1 := succ(X1)
  end
```

$\hat{a}_0 = (4,2)$
 $A_1 = X1 := 0;$
 $\hat{a}_1 = (0,2)$
 $A_2 = X1 \neq X2$
 $\hat{a}_2 = (0,2)$
 $A_3 = X1 := \text{succ}(X1)$
 $\hat{a}_3 = (1,2)$
 $A_4 = X1 \neq X2$
 $\hat{a}_4 = (1,2)$
 $A_5 = X1 := \text{succ}(X1)$
 $\hat{a}_5 = (2,2)$
 $A_6 = X1 \neq X2$
 $\hat{a}_6 = (2,2)$

PW: Ejemplo

- Secuencia de computación de P con vector estado inicial $\hat{a}_0 = (3,4,0,2)$

Programa While P

```
begin
  X2 := X1;
  while X2 ≠ X4 do
    begin
      X3 := succ(X3);
      X2 := pred(X2);
      while X3≠X2 do
        X3 := succ(X3)
    end
  end
```

$\hat{a}_0 = (3,4,0,2)$
 $A_1 = X2 := X1;$
 $\hat{a}_1 = (3,3,0,2)$
 $A_2 = X2 \neq X4$
 $\hat{a}_2 = (3,3,0,2)$
 $A_3 = X3 := \text{succ}(X3)$
 $\hat{a}_3 = (3,3,1,2)$
 $A_4 = X2 := \text{pred}(X2)$
 $\hat{a}_4 = (3,2,1,2)$
 $A_5 = X3 \neq X2$
 $\hat{a}_5 = (3,2,1,2)$
 $A_6 = X3 := \text{succ}(X3)$
 $\hat{a}_6 = (3,2,2,2)$
 $A_7 = X3 \neq X2$
 $\hat{a}_7 = (3,2,2,2)$
 $A_8 = X2 \neq X4$
 $\hat{a}_8 = (3,2,2,2)$

PW: Función Semántica

- Una secuencia de computación es una traza del programa.
 - En el vector estado inicial estará almacenado el input e inicializadas el resto de las variables del programa.
 - Si la secuencia es finita, una vez concluida tendremos en el vector estado final el contenido de todas las variables, incluido el output.
- Podemos definir una función que asocie cada input a su correspondiente output, una vez ejecutado el programa.

PW: Función Semántica

- La **función semántica j-aria**, $\varphi_P^j: \mathbb{N}^j \rightarrow \mathbb{N}$ ($j > 0$) de un programa k variables P se define como sigue:
- Dado un vector de entrada $\hat{a} = (a_1, \dots, a_j)$, la función $\varphi_P^j(a_1, \dots, a_j)$ se evalúa según las reglas siguientes:
 1. **$k \geq j$** : $\varphi_P^j(a_1, \dots, a_j)$ se obtiene aplicando P al vector estado inicial $\hat{a}_0 = (a_1, \dots, a_j, 0, 0, \dots^{(k-j)}, \dots, 0)$
 2. **$k < j$** : $\varphi_P^j(a_1, \dots, a_j)$ se obtiene aplicando P al vector estado inicial $\hat{a}_0 = (a_1, \dots, a_k)$
 - Si P para con ese vector, el contenido final de **X1** es el valor de $\varphi_P^j(a_1, \dots, a_j)$
 - En otro caso, $\varphi_P^j(a_1, \dots, a_j) = \perp$ (indefinido)

PW: Función Semántica

- En la definición anterior, j es el tamaño del input, mientras que k es el número de variables del programa.
- Si el número de variables es mayor que el input, el resto de variables se inicializan a cero. En caso de que el número de variables sea menor que el input, éste no se puede introducir completamente en el vector estado inicial.
- Un mismo **programa** tiene una **función semántica** para cada tamaño de input (**aridad**)

PW: Función Semántica. Ejemplo

- El siguiente PW computa $\varphi^{(2)}(x, y) = x * y$, permitiendo las macros de la suma y la asignación.

```
begin
  while X3 ≠ X2 do
    begin
      X3 := succ(X3);
      X4 := X4+X1
    end
    X1 := X4
  end
```

Programa While **4-variables**: (X1, X2, X3, X4)

Aridad de la función semántica **j=2**

Caso **k ≥ j**: Vector estado inicial:

$$\hat{a}_0 = (x, y, 0, 0)$$

‘x’ se guarda en X1 e ‘y’ en X2. El resto de las variables valen 0

El resultado final se deja en X1

PW: Función Semántica. Ejemplo



- Dado el siguiente programa P, calcula su función semántica de **aridad 1**.

```
begin
  X3:=0;
  while X1 ≠ X3 do
    begin
      X2 := pred(X2);
      X1 := pred(X1)
    end
    X1 := X2
  end
```

Aridad de la función semántica **j=1**

Programa While **3-variables**

Caso **k ≥ j**: Ejecutamos P con vector estado inicial

$$\hat{a}_0 = (x, 0, 0)$$

¿Qué queda en X1 al final de la ejecución?

$$\varphi^{(1)}(x) = 0$$

PW: Función Semántica. Ejemplo



- Dado el siguiente programa P, calcula su función semántica de **aridad 3**.

```
begin
  X3:=0;
  while X1 ≠ X3 do
    begin
      X2 := pred(X2);
      X1 := pred(X1)
    end
    X1 := X2
  end
```

Aridad de la función semántica **j=3**

Programa While **3-variables**

Caso **k ≥ j**: Ejecutamos P con vector estado inicial

$$\hat{a}_0 = (x, y, z)$$

¿Qué queda en X1 al final de la ejecución?

$$\varphi^{(3)}(x, y, z) = y - x$$

PW: Función Semántica. Ejemplo



- Dado el siguiente programa P, calcula su función semántica de **aridad 5**.

```
begin
  X3:=0;
  while X1 ≠ X3 do
    begin
      X2 := pred(X2);
      X1 := pred(X1)
    end
    X1 := X2
  end
```

Aridad de la función semántica **j=5**

Programa While **3-variables**

Caso **k < j**: Ejecutamos P con vector estado inicial

$$\hat{a}_0 = (x, y, z)$$

¿Qué queda en X1 al final de la ejecución?

$$\varphi^{(5)}(x, y, z, u, v) = y - x$$

PW: Función Semántica. Ejemplo



- Dado el siguiente programa P, calcula sus funciones semánticas de **aridad 1** y **aridad 3**.

```
begin
  X4:=X1+X2;
  while X4 > X5 do
    begin
      X5 := succ(X5);
      X4 := pred(X4)
    end
    X1 := X4;
    while X3 ≠ 0 do
      begin
        X1 := succ(X1);
        X3 := pred(X3)
      end
    end
```

PW: Función Semántica. Ejemplo



Vector estado inicial: $\hat{a}_0 = (x, 0, 0, 0, 0)$

Tras cada iteración i:

$$\hat{a} = (x, 0, 0, x-i, i)$$

El bucle acaba tras la primera iteración i tal que $X4 \leq X5$, o lo que es lo mismo, cuando $(x/2) \leq i$.

Si x es par:

$$(x, 0, 0, x/2, x/2)$$

$$(x/2, 0, 0, x/2, x/2)$$

No se cumple

Si x es impar:

$$(x, 0, 0, x/2, x/2+1)$$

$$(x/2, 0, 0, x/2, x/2+1)$$

No se cumple

$$\hat{a}_1 = (x, 0, 0, x, 0)$$

Vector estado
tras el bucle:

```
begin
    X4 := X1 + X2;
    while X4 > X5 do
        begin
            X5 := succ(X5);
            X4 := pred(X4)
        end
        X1 := X4;
        while X3 ≠ 0 do
            begin
                X1 := succ(X1);
                X3 := pred(X3)
            end
        end
    end
```

$$\varphi^{(1)}(x) = x/2$$

PW: Función Semántica. Ejemplo



Vector estado inicial: $\hat{a}_0 = (x, y, z, 0, 0)$

Tras cada iteración i:

$$\hat{a} = (x, y, z, x+y-i, i)$$

El bucle acaba tras la primera iteración i tal que $X4 \leq X5$, o lo que es lo mismo, cuando $(x+y)/2 \leq i$.

$$(x, y, z, (x+y)/2, (x+y)/2+1)$$

$$((x+y)/2, y, z, (x+y)/2, (x+y)/2+1)$$

$$((x+y)/2+z, y, 0, (x+y)/2, (x+y)/2+1)$$

$$\hat{a}_1 = (x, y, z, x+y, 0)$$

Vector estado
tras el bucle:

```
begin
  X4 := X1 + X2;
  while X4 > X5 do
    begin
      X5 := succ(X5);
      X4 := pred(X4)
    end
    X1 := X4;
    while X3 ≠ 0 do
      begin
        X1 := succ(X1);
        X3 := pred(X3)
      end
    end
end
```

Vector estado
tras el bucle:

$$\varphi^{(3)}(x) = \frac{x + y}{2} + z$$

Tras cada iteración i, X1 se incrementa en 1 unidad y X3 se decrementa en 1 unidad
El bucle acaba tras X3 iteraciones.

Funciones While-Computables

Una función $f: \mathbb{N}^j \rightarrow \mathbb{N}$ es **while-computable**, o simplemente computable, si existe algún programa while **P** :

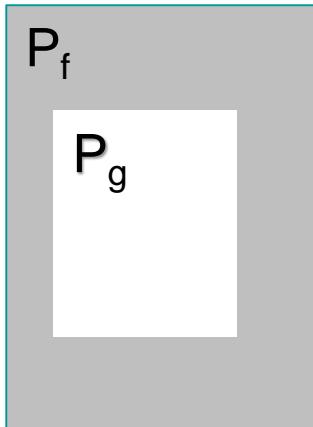
$$\varphi_P^{(j)}(a_1, \dots, a_j) = f(a_1, \dots, a_j) \quad \forall (a_1, \dots, a_j) \in \mathbb{N}^j$$

Contenidos

- **Programas While**
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- **Máquinas de Turing**
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- **Equivalencia entre PW y MT**
- **Tesis de Church**

Composición de PW

- ¿Qué ocurre si queremos integrar un programa While dentro de otro?



- P_f computa una función de aridad j y es k1-variable:
$$\hat{a}_0 = (a_1, \dots, a_j, 0, \dots^{(k1-j)} \dots, 0)$$
 - La salida está en X1
 - P_g computa una función de aridad i y es k2-variable:
$$\hat{a}_0 = (a_1, \dots, a_i, 0, \dots^{(k2-i)} \dots, 0)$$
 - Deja su salida en X1
-
- P_f , además de las de P_g y las que necesite para sus cálculos, utilizará variables adicionales para asegurar que el vector de estado es en cada momento el que debe ser:
 - Antes de ejecutar P_g , asegurar que el vector de estado sea el correcto
 - Guardar la salida de P_g , en variables que no use P_g , si procede

Composición de PW: Ejemplo

- Sea P_g un programa while con exactamente k variables. Diseña otro programa while P_f que integre el código de P_g para computar la siguiente función ternaria:

$$f(x, y, z) = g(x \div y, x + y) + g(x \div z, x + z)$$

P_f será k' -variables. $\longrightarrow f(x, y, z) = \varphi_{P_f}^{(3)}(x, y, z)$, con $\hat{a}_0 = (x, y, z, 0, \dots^{(k'-3)}, \dots, 0)$

P_g es k -variables. $\longrightarrow g(x, y) = \varphi_{P_g}^{(2)}(x, y)$, con $\hat{a}_0 = (x, y, 0, \dots^{(k-2)}, \dots, 0)$

Para computar la salida de P_f necesitamos sumar las salidas de utilizar dos veces P_g :

- Una vez con vector estado inicial: $\hat{a}_0 = (x \div y, x + y, 0, \dots^{(k-2)}, \dots, 0)$
- Una vez con vector estado inicial: $\hat{a}_0 = (x \div z, x + z, 0, \dots^{(k-2)}, \dots, 0)$

Composición de PW: Ejemplo

$$f(x, y, z) = g(x \dot{-} y, x + y) + g(x \dot{-} z, x + z)$$

- Debemos preparar el vector estado inicial para cada llamada a P_g

$$\hat{a}_0 = (x, y, z, 0, \dots^{(k'-3)}, \dots, 0)$$

```
begin
  X3 := X1 - X2;
  X2 := X1 + X2;
  X1 := X3;
  X3 := 0;
  ...
  Xk := 0;
  Pg;
  ...
end
```

Antes de utilizar P_g :

$$\hat{a} = (x \dot{-} y, x + y, 0, \dots^{(k-2)}, \dots, 0, 0, \dots^{(k'-k)}, \dots, 0)$$

Después de utilizar P_g :

$$\hat{a} = (g(x \dot{-} y, x + y), ?, \dots^{(k-1)}, \dots, ?, 0, \dots^{(k'-k)}, \dots, 0)$$

Hemos perdido x y z. No podemos utilizar P_g por segunda vez para calcular $g(x \dot{-} z, x + z)$

Composición de PW: Ejemplo

$$f(x, y, z) = g(x-y, x+y) + g(x-z, x+z)$$

- Debemos guardar las variables que necesitamos (x, z)
- La primera variable que P_g no sobreescribe es X_{k+1}

```
begin
  Xk+1 := X1;
  Xk+2 := X3;
  X1 := X1 - X2;
  X2 := Xk+1 + X2;
  X3 := 0;
  ...
  Xk := 0;
  Pg;
  ...
end
```

$$\hat{a}_0 = (x, y, z, 0, \dots^{(k'-3)}, \dots, 0)$$

Antes de utilizar P_g :

$$\hat{a} = (x-y, x+y, 0, \dots^{(k-2)}, \dots, 0, \textcolor{red}{x}, \textcolor{red}{z}, 0, \dots^{(k'-k-2)}, \dots, 0)$$

Después de utilizar P_g :

$$\hat{a} = (g(x-y, x+y), ?, \dots^{(k-1)}, \dots, ?, \textcolor{red}{x}, \textcolor{red}{z}, 0, \dots^{(k'-k-2)}, \dots, 0)$$

Composición de PW: Ejemplo

$$f(x, y, z) = g(x - y, x + y) + g(x - z, x + z)$$

- Nos preparamos para usar P_g por segunda vez

```
begin
  Xk+1 := X1;
  Xk+2 := X3;
  X1 := X1 - X2;
  X2 := Xk+1 + X2;
  X3 := 0; ...; Xk := 0
  Pg;
  Xk+3 := X1;
  X1 := Xk+1 - Xk+2;
  X2 := Xk+1 + Xk+2;
  X3 := 0; ...; Xk := 0
  Pg;
  X1 := X1 + Xk+3
end
```

El resultado de utilizar P_g está en la variable $X1$. Lo guardamos para sumarlo al final

Preparamos el vector de estado para calcular $g(x - z, x + z)$

Sumamos $g(x - y, x + y)$ (guardado en $Xk+3$) y $g(x - z, x + z)$ (guardado en $X1$)

Composición de PW

- En resumen: Cuando queramos integrar el código de un PW k-variables P_g , en otro programa debemos:
 1. Guardar los valores que no queramos perder.
 - Se guardan a partir de la variable X_{k+1}
 2. Preparar el vector estado inicial para P_g .
 - Si queremos calcular $\varphi_g^{(j)}(x_1, \dots, x_j)$, se colocan los valores correspondientes en las variables X_1, X_2, \dots, X_j .
 - El resto de variables (X_{j+1}, \dots, X_k) se ponen a 0
 3. Ejecutar el código de P_g .
 4. Recoger el resultado de ejecutar P_g .
 - El resultado siempre queda en la variable X_1

Composición de PW: Ejercicio I



- Sea P_g un PW con exactamente k variables, diseña otro PW P cuya función semántica ternaria sea:

```
begin
  Xk+1 := X1;
  Xk+2 := X2;
  Xk+3 := X3;
  while (Xk+3 ≠ Xk+4) do
    begin
      X1 := Xk+1 + Xk+3 ;
      X2 := Xk+2 * Xk+3 ;
      X3:=0; ...; Xk:=0;
      Pg;
      Xk+5 := Xk+5 + X1;
      Xk+3 := pred(Xk+3);
    end
    X1 = Xk+5;
  end
```

$$\varphi_P^{(3)}(x, y, z) = \begin{cases} \sum_{i=1}^z g(x + i, y * i) & \text{si } z \neq 0 \\ 0 & \text{si } z = 0 \end{cases}$$

Guardamos x, y, z

Preparamos el vector estado inicial para P_g

Ejecutamos el código de P_g

Acumulamos y guardamos el resultado en una variable “segura”

Composición de PW: Ejercicio II



- Diseña un PW para la división entera, suponiendo que se dispone de dos programas P1 y P2 de k1 y k2 variables respectivamente, que calculan el producto y la expresión algebraica asociada a la comparación \leq , que vale >0 si es cierta y 0 si es falsa:

$$f(x, y) = \text{div}(x, y) = \max\{z \geq 0 \mid z * y \leq x\}$$

```
begin
  Xk+1 := X1;
  Xk+2 := X2;
  X1 := 0;
  X2 := Xk+2;
  P1;
  X2 := Xk+1;
  X3 := 0; ...; Xk2 := 0;
  P2
  ...
```

Guardamos x, y. La primera variable segura es X_{k+1} , con $k = \max\{k_1, k_2\}$

Preparamos el vector estado inicial para calcular
 $\varphi_{P1}^{(2)}(0, y) = 0 * y$

Preparamos el vector estado inicial para calcular
 $\varphi_{P2}^{(2)}(0 * y, x) = \begin{cases} > 0 & \text{si } 0 * y \leq x \\ 0 & \text{si } 0 * y > x \end{cases}$

Composición de PW: Ejercicio II



$$f(x, y) = \text{div}(x, y) = \max\{z \geq 0 \mid z * y \leq x\}$$

```
begin
  Xk+1 := X1;
  Xk+2 := X2;
  X1 := 0;
  X2 := Xk+2;
P1;
  X2 := Xk+1;
  X3 := 0; ...; Xk2:=0;
P2;
  ...
  ...
```

```
...
while (X1 ≠ Xk+4) do
begin
  Xk+3 := succ(Xk+3);
  X1 := Xk+3;
  X2 := Xk+2;
  X3 := 0; ...; Xk1 := 0;
P1;
  X2 := Xk+1;
  X3 := 0; ...; Xk2 := 0;
P2;
end;
  X1 := pred(Xk+3);
end
```

$$\varphi_{P1}^{(2)}(z, y) = z * y$$

$$\varphi_{P2}^{(2)}(z * y, x) = \begin{cases} > 0 & \text{si } z * y \leq x \\ 0 & \text{si } z * y > x \end{cases}$$

Contenidos

- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

Modelo de las Máquinas de Turing



Alan Turing
1912 - 1954

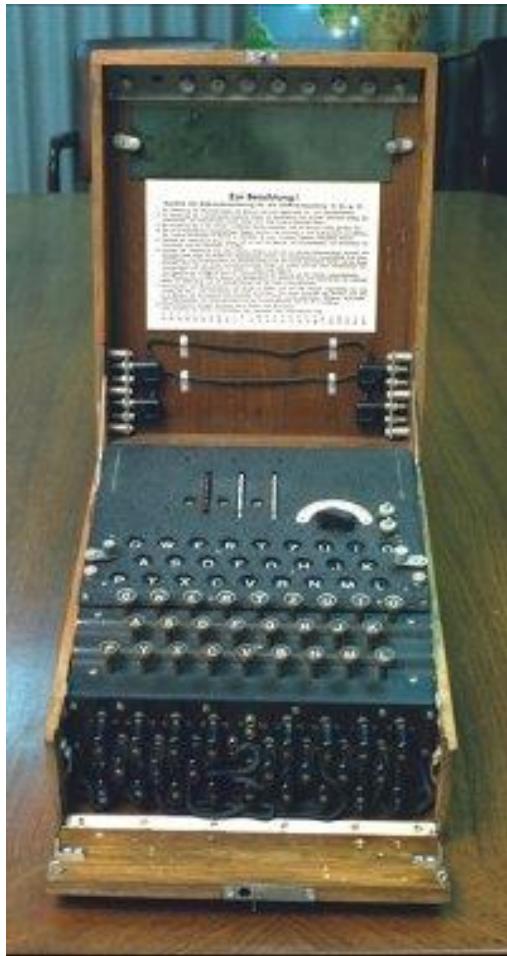
Alan Turing fue uno de los fundadores de la Informática.

- Su modelo de computador fue una premonición del computador electrónico que llegaría dos décadas después.
- Fue uno de los principales artífices de los trabajos del Bletchley Park para descifrar los códigos secretos nazis en WWII.
- Inventó el “Turing Test” usado en AI.

Algo de Historia: II Guerra Mundial

- Papel decisivo en descifrar los mensajes secretos nazis (código de la máquina Enigma).
- 1918 Arthur Scherbius construyó la Enigma.
 - Máquina de rotores que permitía cifrar y descifrar mensajes
 - En 1923 se vendió para uso comercial
 - En 1926 Alemania la adoptó para uso militar.
 - Ventaja: fácil manejo y cifrado difícil (supuestamente inviolable).
 - Polonia tenía buenos expertos descifrando códigos. Francia llegó a comprar las claves, pero no pudieron hacer nada con ellas.
 - <http://www.enigmaco.de/>

Algo de Historia: II Guerra Mundial



- En 1939 solicitaron ayuda a Turing para descifrar Enigma.
 - (Análisis Criptográfico de Enigma)
- Sus estudios ayudaron a construir Colussus (“bomba”), el primer ordenador programable (Max Newman).
- A partir de 1943 se pudieron descifrar los mensajes en Enigma.
- Winston Churchill ordenó destruir todos los equipos

Algo de Historia: II Guerra Mundial

- Construcción primeros ordenadores programables: **Colossus** y **Mark I**.
 - Basado en:
 - La velocidad y la fiabilidad de la tecnología electrónica.
 - La ineficiencia del diseño de máquinas diferentes para diferentes procesos lógicos.
- Su participación en **Colossus**, le hizo pensar que se podría construir un ordenador que trabajase como la mente humana.
- *“Turing estaba convencido de que si un computador podía hacer todas las operaciones matemáticas, podría hacer todo lo que una persona pudiera hacer”* (**Tesis de Church-Turing**)

Alan Turing

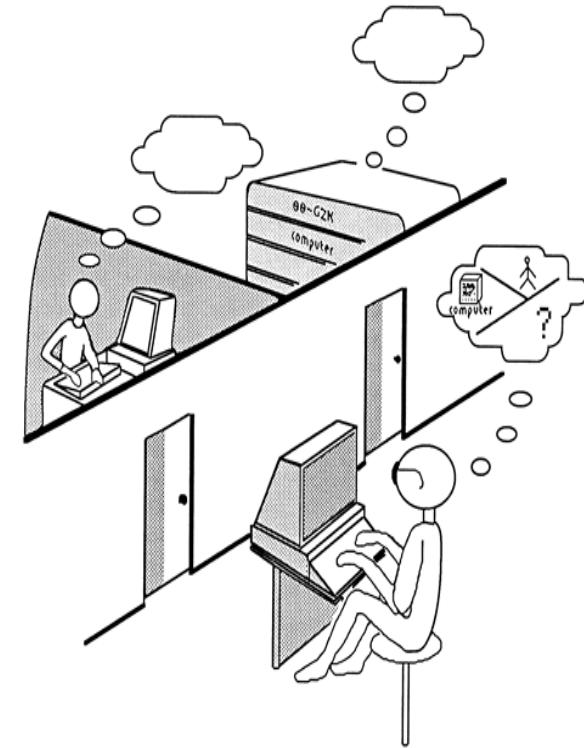
- Formalización del concepto de algoritmo:
 - Extiende el trabajo de Gödel (sobre los límites de la demostrabilidad y la computación) sustituyendo el lenguaje formal universal por las:

Máquinas de Turing

- Tesis de Church-Turing

El Test de Turing

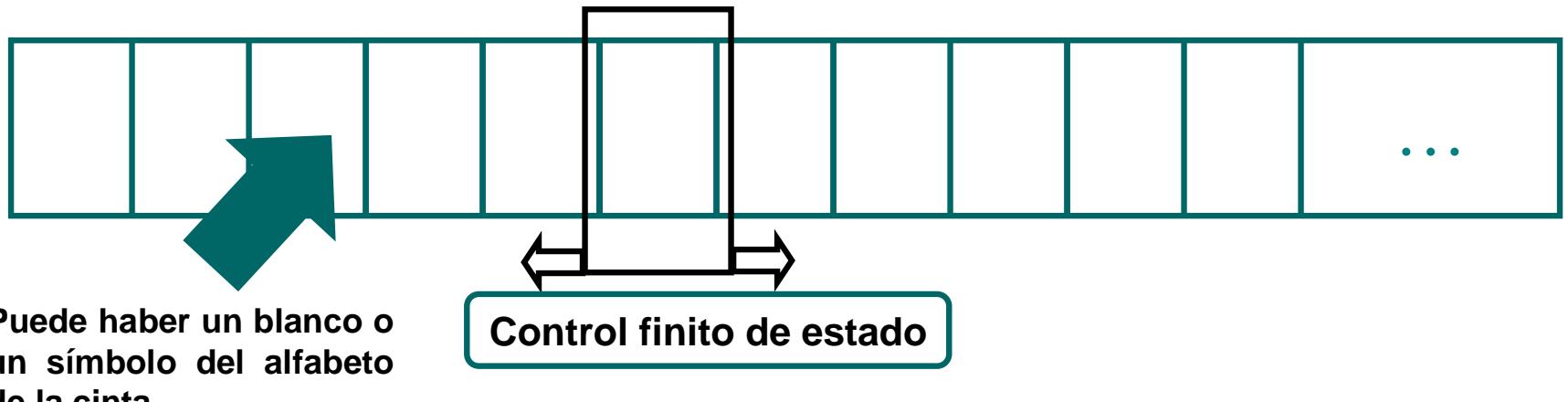
- A. Turing, “*Computing machinery and intelligence*”, Mind (1950)
- Definición operativa de inteligencia
- Premio Loebner al programa más inteligente
- **CAPTCHA**: Prueba de Turing pública y automática para diferenciar máquinas y humanos



Contenidos

- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

Concepto de Máquina de Turing (MT)



- Cinta extensible “ad infinitum” en ambas direcciones.
- Cabeza lectora/escritora.
- Control finito (estados):
 - programa que controla la posición de la cabeza lectora, el símbolo que se está leyendo y el estado actual.

Funcionamiento Máquina de Turing

- Inicialmente:
 - En la cinta se coloca el input codificado.
 - La cabeza lectora/escritora se sitúa sobre el primer símbolo de la cadena de entrada
 - El estado de la máquina es el denominado estado inicial.
- En cada movimiento, dependiendo del **estado** y del **símbolo** sobre el que se encuentra la cabeza, la MT puede **cambiar de estado**, **imprimir** un símbolo en la cinta y **mover la cabeza** una celda a la derecha, a izquierda o bien no moverse.
- Si la computación termina, dependiendo del estado en que se encuentre, la cinta tendrá codificado el output.

Funcionamiento Máquina de Turing

- Primera instrucción: Estado inicial. Cabeza sobre el primer símbolo no blanco de la cinta.
- Instrucción siguiente:
 - Dependiendo del par (estado, símbolo):
 - Cambiar de estado.
 - Escribir un símbolo en la cinta.
 - Realizar una de las siguientes acciones:
 - Mover cabeza a la izquierda (I)
 - Mover cabeza a la derecha (D)
 - No moverse (N)
 - Parar la computación (H)

Funcionamiento Máquina de Turing

- Test de parada: La Máquina para cuando no hay instrucción siguiente a realizar o cuando se le indica
- Resultado:
 - Si **para** y se encuentra en un **estado final**, el resultado está **codificado en la cinta**.
 - Si **para** y se encuentra en un **estado no final**, el resultado se considerará **indefinido**
 - Si **no para**, el resultado se considera **indefinido**

Definición Formal de una MT

- Una MT es una quíntupla (X, Q, T, i, F) donde:
 - X es el alfabeto de símbolos que pueden aparecer en la cinta.
Posee un símbolo especial o símbolo blanco (B).
 - Q es un conjunto finito y no vacío de estados.
 - $i \in Q$ es el estado inicial.
 - $F \subseteq Q$ es el conjunto de estados finales.
 - $T: Q \times X \rightarrow X \times \text{Acciones} \times Q$
 $\text{Acciones} = \{I, D, N, H\}$

Secuencia de computación en MT

■ Patrón de una **descripción instantánea**:

$y_1 \dots y_n \textcolor{red}{q} \textcolor{blue}{x}_1 \dots x_m$

- $\textcolor{red}{q}$: Estado actual de la máquina
- $\textcolor{blue}{x}_1$: Símbolo sobre el que se encuentra la cabeza
- $x_2 \dots x_m$: Símbolos desde la cabeza hasta el último símbolo no blanco.
- $y_1 \dots y_m$: Símbolos desde el primer símbolo no blanco hasta la cabeza
 - y_1 es el primer símbolo no blanco.

■ Descripción inicial: $i \textcolor{blue}{x}_1 \dots x_m$

Secuencia de computación en MT

- Una **transición** nos da la descripción siguiente a una descripción instantánea:

$y_1 \dots y_n \textcolor{red}{q} \textcolor{blue}{x}_1 \dots x_m$

- Si $T(\textcolor{red}{q}, \textcolor{blue}{x}_1) = (\textcolor{teal}{x}'_1, \textcolor{red}{q}', \textcolor{violet}{D})$:

$$y_1 \dots y_n \textcolor{red}{q} \textcolor{blue}{x}_1 \dots x_m \rightarrow y_1 \dots y_n \textcolor{teal}{x}'_1 \textcolor{red}{q}' \textcolor{blue}{x}_2 \dots x_m$$

- Si $T(\textcolor{red}{q}, \textcolor{blue}{x}_1) = (\textcolor{teal}{x}'_1, \textcolor{red}{q}', \textcolor{violet}{I})$:

$$y_1 \dots y_n \textcolor{red}{q} \textcolor{blue}{x}_1 \dots x_m \rightarrow y_1 \dots y_{n-1} \textcolor{red}{q}' y_n \textcolor{teal}{x}'_1 x_2 \dots x_m$$

- Si $T(\textcolor{red}{q}, \textcolor{blue}{x}_1) = (\textcolor{teal}{x}'_1, \textcolor{red}{q}', \textcolor{violet}{N})$

$$y_1 \dots y_n \textcolor{red}{q} \textcolor{blue}{x}_1 \dots x_m \rightarrow y_1 \dots y_n \textcolor{red}{q}' \textcolor{teal}{x}'_1 x_2 \dots x_m$$

- Si $T(\textcolor{red}{q}, \textcolor{blue}{x}_1) = (\textcolor{teal}{x}'_1, \textcolor{red}{q}', \textcolor{violet}{H})$

$$y_1 \dots y_n \textcolor{red}{q} \textcolor{blue}{x}_1 \dots x_m \rightarrow y_1 \dots y_n \textcolor{red}{q}' \textcolor{teal}{x}'_1 x_2 \dots x_m$$

Secuencia de computación en MT

■ Casos “especiales”:

- $T(q, x_1) = (x'_1, q', l)$:

$$q \ x_1 \dots x_m \rightarrow q' \ B \ x'_1 \ x_2 \dots x_m$$

- $T(q, x_1) = (x'_1, q', D)$:

$$y_1 \dots y_n \ q \ x_1 \rightarrow y_1 \dots y_n \ x'_1 \ q' \ B$$

■ Una **secuencia de computación** es una sucesión finita o infinita de descripciones instantáneas que comienza con:

i $x_1 \dots x_m$

Contenidos

- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

Computación con una MT

■ Codificación de la entrada:

- Notación unaria ($X = \{0,1\}$)
- Dado $n \in \mathbb{N}$, éste se codifica en la cinta con **(n+1) 1's consecutivos.**
- Dado un vector (n_1, \dots, n_k) su codificación será:

$1 \dots^{(n_1+1)} \dots 1 \textcolor{teal}{0} 1 \dots^{(n_2+1)} \dots 1 \textcolor{teal}{0} \dots 0 1 \dots^{(n_k+1)} \dots 1$

Función Semántica de una MT

- La **función semántica j-aria** $\varphi_M^j: \mathbb{N}^j \rightarrow \mathbb{N}$ ($j > 0$) de una máquina de Turing $M = (X, Q, T, i, F)$ se define como sigue:
- Dado un input (n_1, \dots, n_j) , la función $\varphi_M^j(n_1, \dots, n_j)$ se evalúa según las reglas siguientes:
 - Se codifica el vector de entrada en la cinta.
 - Se pone la máquina en el estado inicial con la cabeza en el primer 1.
 - Si la computación de M para en un estado final, $\varphi_M^j(n_1, \dots, n_j)$ es el **número de 1's** (consecutivos o no) de la cinta
 - En otro caso, $\varphi_M^j(n_1, \dots, n_j) = \perp$ (indefinido)

MT: Función Turing-Computable

Una función $f: \mathbb{N}^j \rightarrow \mathbb{N}$, se dice que es **Turing-computable**, si existe una MT, \mathbf{M} , tal que:

$$\varphi_M^{(j)}(x_1, \dots, x_j) = f(x_1, \dots, x_j) \quad \forall (x_1, \dots, x_j) \in \mathbb{N}^j$$

MT: Ejemplo X+Y

- Construye una MT \mathbf{M} , tal que $\varphi_{\mathbf{M}}^{(2)}(x, y) = x + y$ y haz la secuencia de computación para $\varphi_{\mathbf{M}}^{(2)}(2, 1)$.

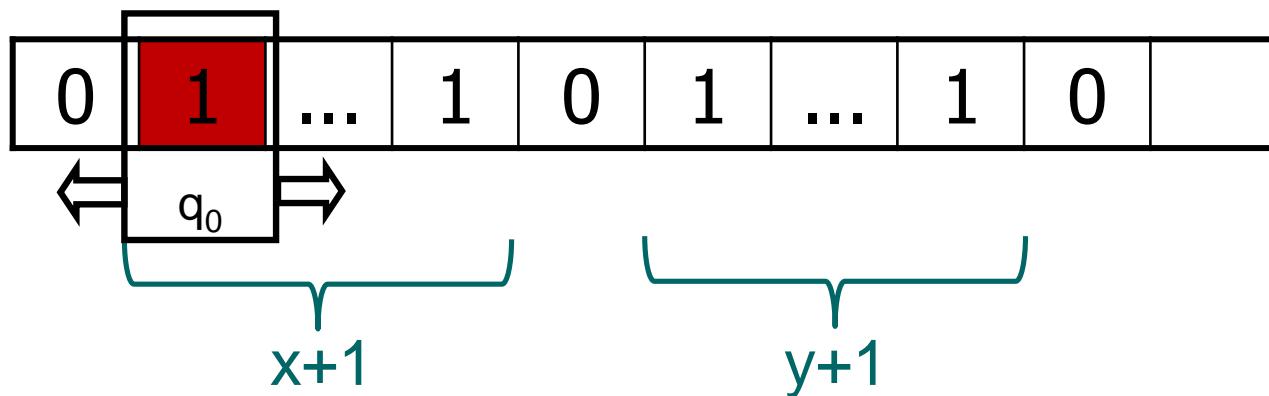
$\mathbf{M} = (X, Q, T, i, F)$:

- $X = \{0,1\}$ (Notación unaria)
- $Q = \{q_0\}$ (veremos si necesitamos más)
- T : La iremos definiendo poco a poco
- $i = q_0$
- F = Lo definiremos más adelante

MT: Ejemplo X+Y

- Construye una MT \mathbf{M} , tal que $\varphi_{\mathbf{M}}^{(2)}(x, y) = x + y$.

Codificación de entrada



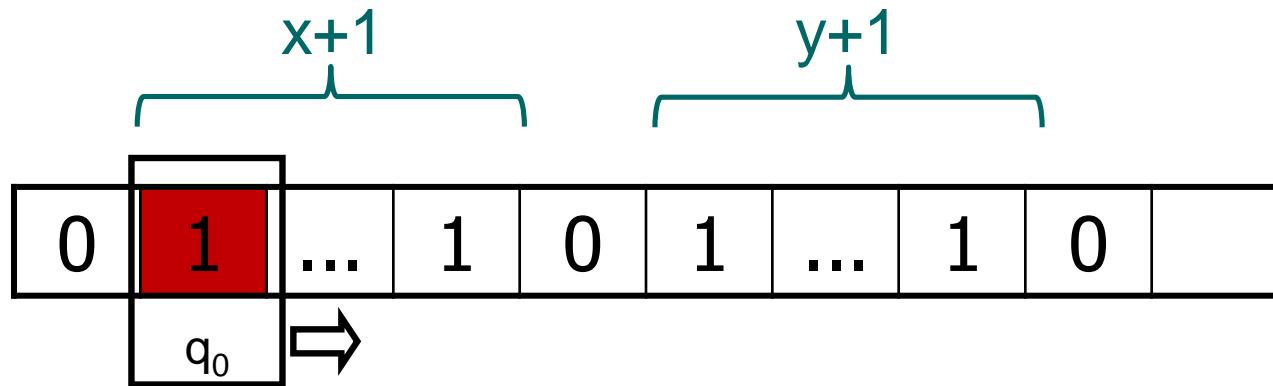
Objetivo:

- Terminar en un estado final con exactamente $x+y$ 1's en la cinta

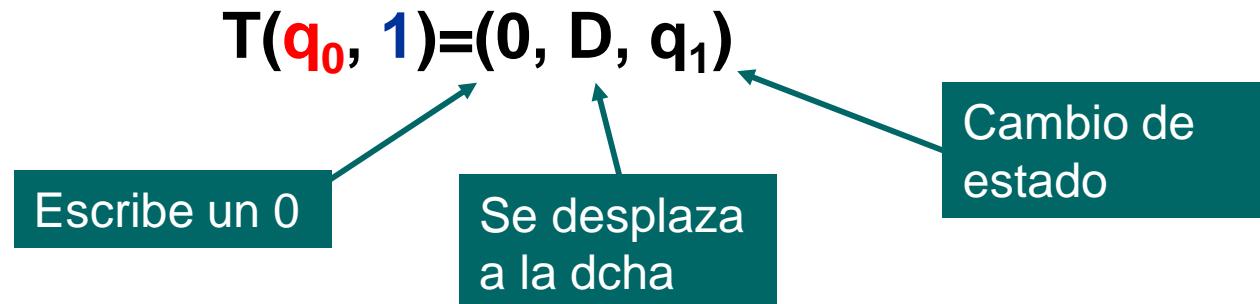
Idea:

- Tenemos $x+y+2$ 1's en la cinta. Debemos eliminar dos de ellos

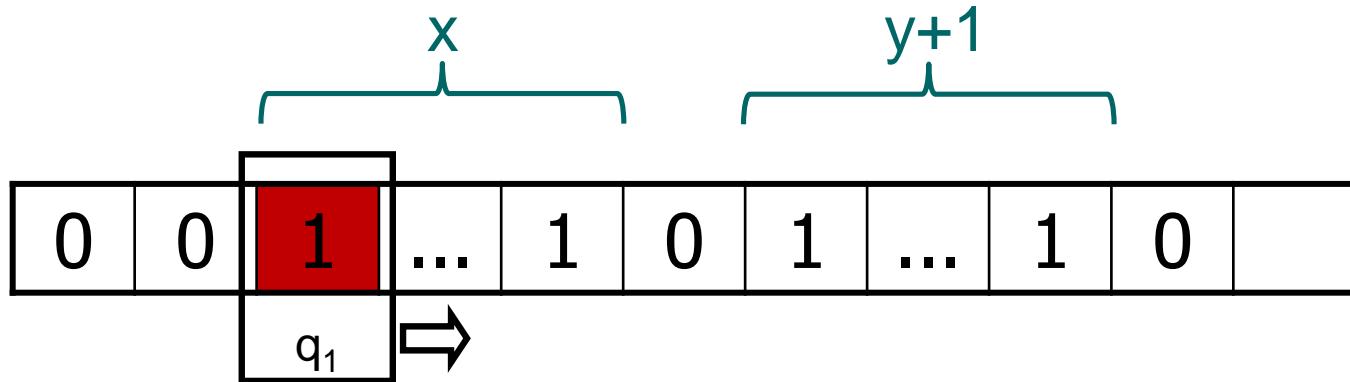
MT: Ejemplo X+Y



- Eliminamos el primer 1 de la codificación de x y nos movemos a la derecha
- Si no cambiamos de estado, borraremos todos los 1's



MT: Ejemplo X+Y



- Si $x > 0$, nos encontramos otro 1
- Si $x = 0$, nos encontramos con un 0
- Lo más seguro es eliminar el 1 al comienzo de y

Si $x > 0$:

Pasamos sobre
los 1's de x

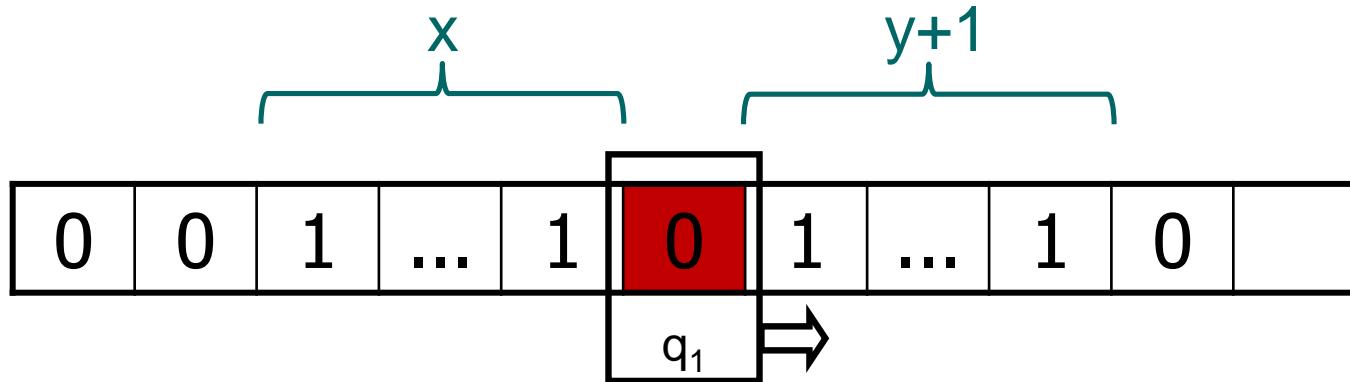
$$T(q_1, 1) = (1, D, q_1)$$

Deja el 1

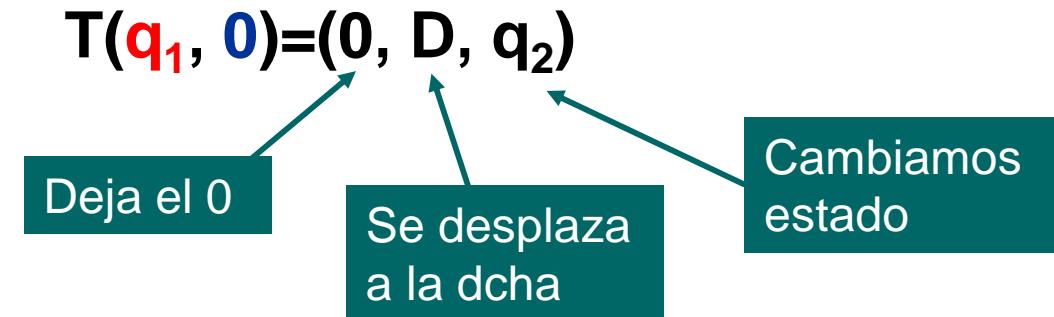
Se desplaza
a la dcha

Mantenemos
estado

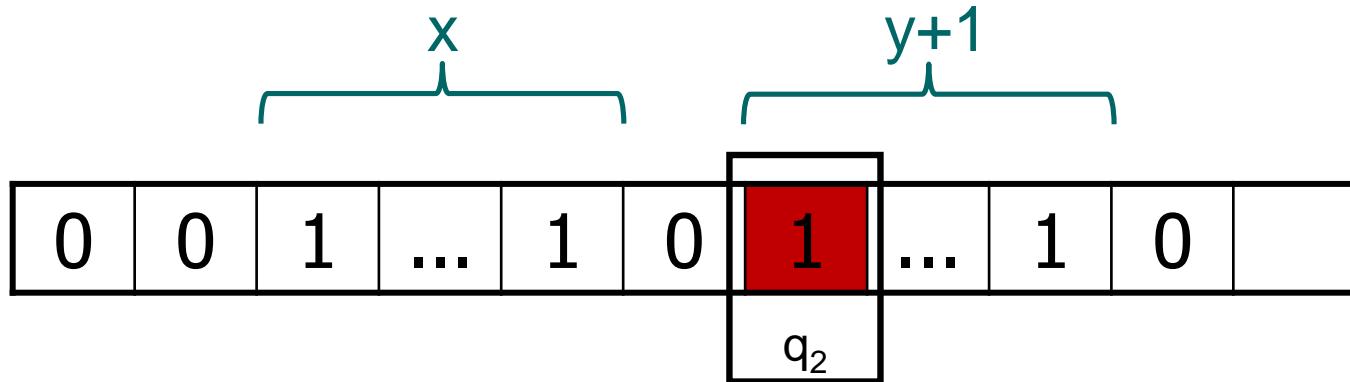
MT: Ejemplo X+Y



- Cuando nos encontramos con el 0 que separa x e y .
- Pasamos sobre él y cambiamos de estado



MT: Ejemplo X+Y



- Eliminamos el primer 1 de y
- Hemos terminado

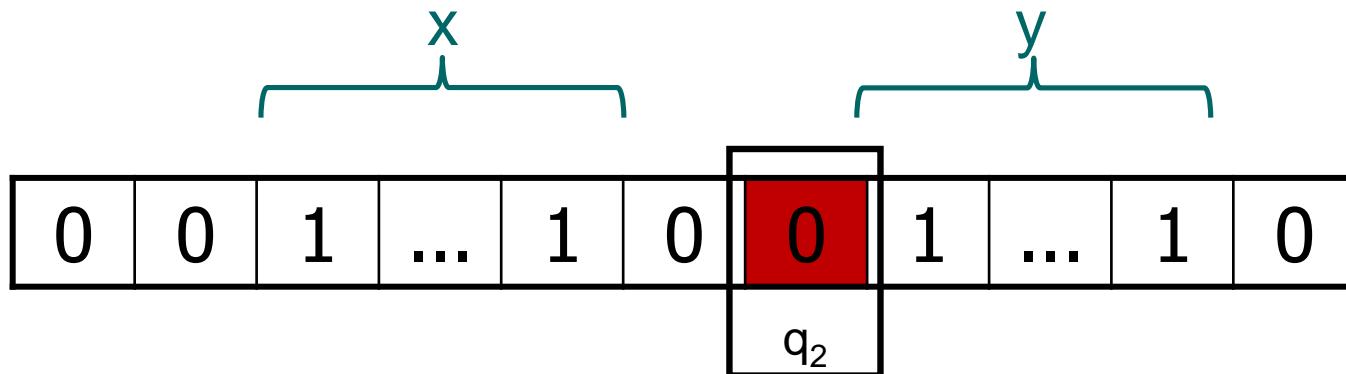
$$T(q_2, 1) = (0, N, q_2)$$

Escribe un 0 sobre el 1

No se desplaza

Mantenemos estado

MT: Ejemplo X+Y



- Quedan exactamente $x+y$ 1's
- Dejamos la transición $T(q_2, 0)$ sin definir.
- La máquina se para y marcamos q_2 como estado final

MT: Ejemplo X+Y

- Construye una MT \mathbf{M} , tal que $\varphi_{\mathbf{M}}^{(2)}(x, y) = x + y$ y haz la secuencia de computación para $\varphi_{\mathbf{M}}^{(2)}(2, 1)$.

$\mathbf{M} = (X, Q, T, i, F)$:

- $X = \{0, 1\}$
- $Q = \{q_0, q_1, q_2\}$
- $i = q_0$
- $F = \{q_2\}$

Formato alterativo:

$T(q_0, 1) = (0, D, q_1)$

$(q_0, 1, 0, D, q_1)$

$T(q_1, 1) = (1, D, q_1)$

$(q_1, 1, 1, D, q_1)$

$T(q_1, 0) = (0, D, q_2)$

$(q_1, 0, 0, D, q_2)$

$T(q_2, 1) = (0, N/H, q_2)$

$(q_2, 1, 0, N/H, q_2)$

MT: Ejemplo X+Y

- Construye una MT \mathbf{M} , tal que $\varphi_M^{(2)}(x, y) = x + y$ y haz la secuencia de computación para $\varphi_M^{(2)}(2, 1)$.

$$\begin{array}{ll} M = (\{0,1\}, \{q_0, q_1, q_2\}, T, q_0, \{q_2\}) & (q_0, 1, 0, D, q_1) \\ & (q_1, 1, 1, D, q_1) \\ & (q_1, 0, 0, D, q_2) \\ & (q_2, 1, 0, N/H, q_2) \end{array}$$

Secuencia de computación:

$$\begin{aligned} (q_0 \textcolor{red}{1} 1 1 0 1 1) &\rightarrow (0 \textcolor{blue}{q}_1 \textcolor{red}{1} 1 0 1 1) \rightarrow (01 \textcolor{blue}{q}_1 \textcolor{red}{1} 0 1 1) \rightarrow (011 \textcolor{blue}{q}_1 \textcolor{red}{0} 1 1) \rightarrow \\ &\rightarrow (0110 \textcolor{blue}{q}_2 \textcolor{red}{1} 1) \rightarrow (0110 \textcolor{blue}{q}_2 \textcolor{red}{0} 1) \end{aligned}$$

Termina en un estado final

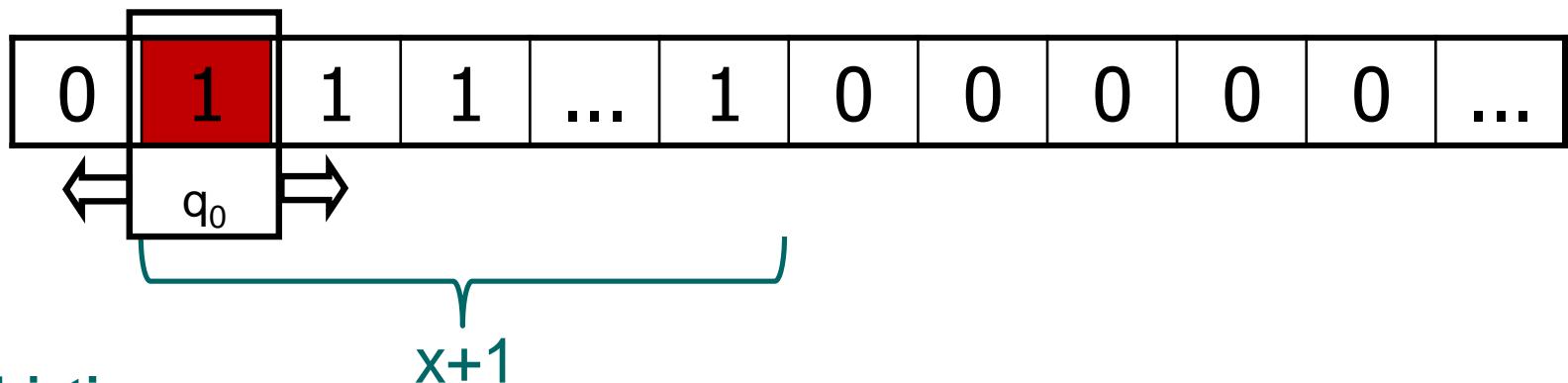
Número de 1's al final de la computación: **3**

MT: Ejemplo 2X



- Construye una MT M , tal que $\varphi_M^{(1)}(x) = 2x$ y haz la secuencia de computación para $\varphi_M^{(1)}(3)$.

Codificación de entrada



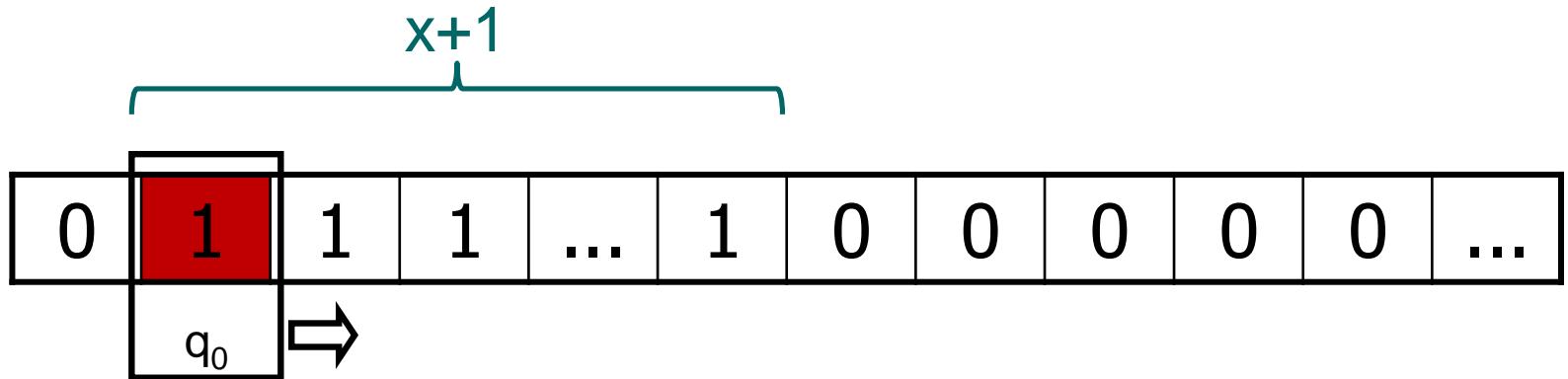
Objetivo:

- Terminar en un estado final con exactamente $2x$ 1's en la cinta

Idea:

- Eliminamos un 1 y duplicamos el resto

MT: Ejemplo 2X



- Eliminamos el primer 1 de la codificación de x y nos movemos a la derecha
- Si no cambiamos de estado, borraremos todos los 1's

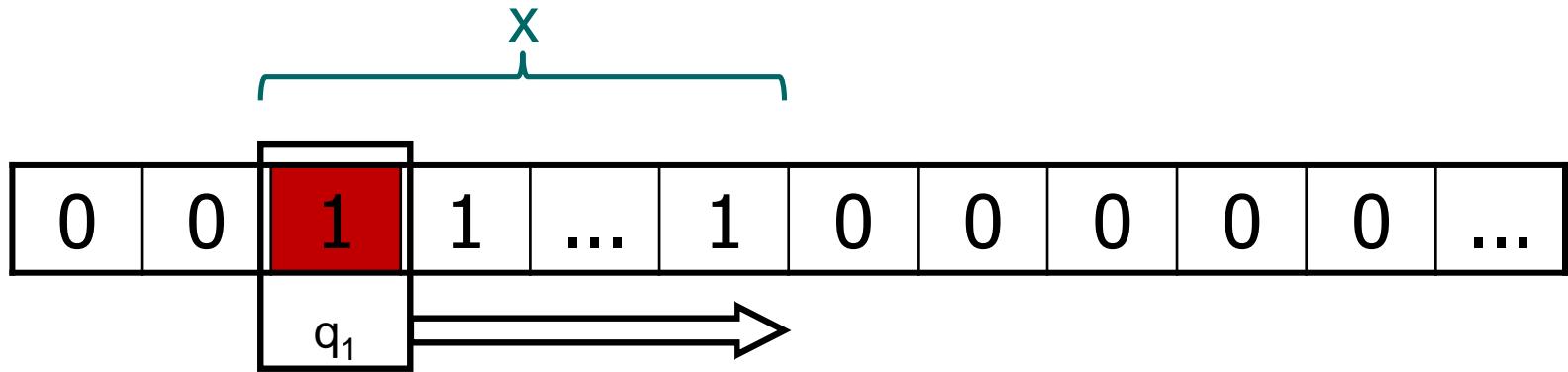
$$T(q_0, 1) = (0, D, q_1)$$

Escribe un 0

Se desplaza
a la dcha

Cambio de
estado

MT: Ejemplo 2X

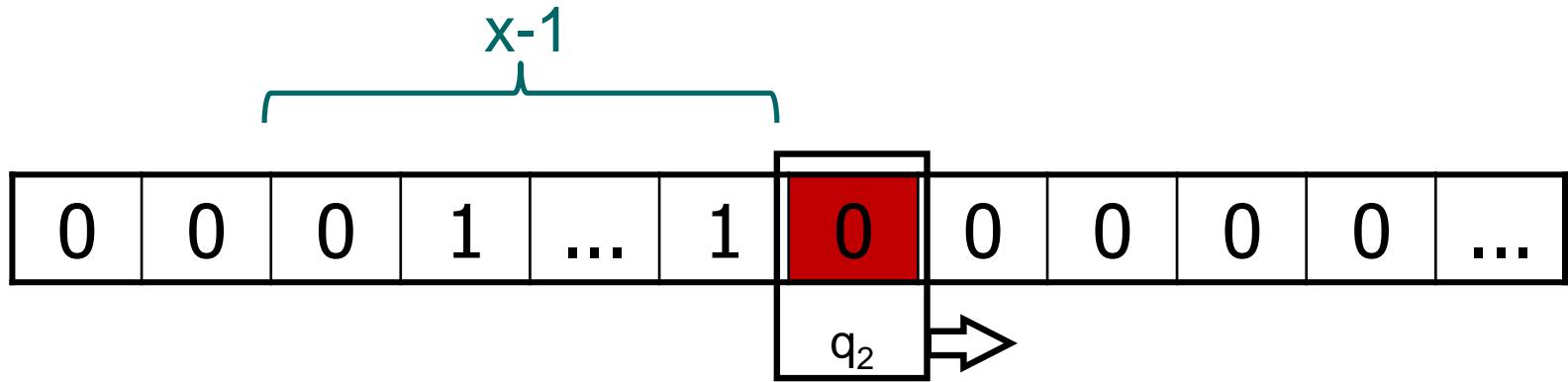


- Marcamos este 1 para duplicar
- Como solo hay 0's y 1's, lo marcamos poniéndolo a 0
- Nos movemos hacia la zona de 0's para “copiarlo”

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

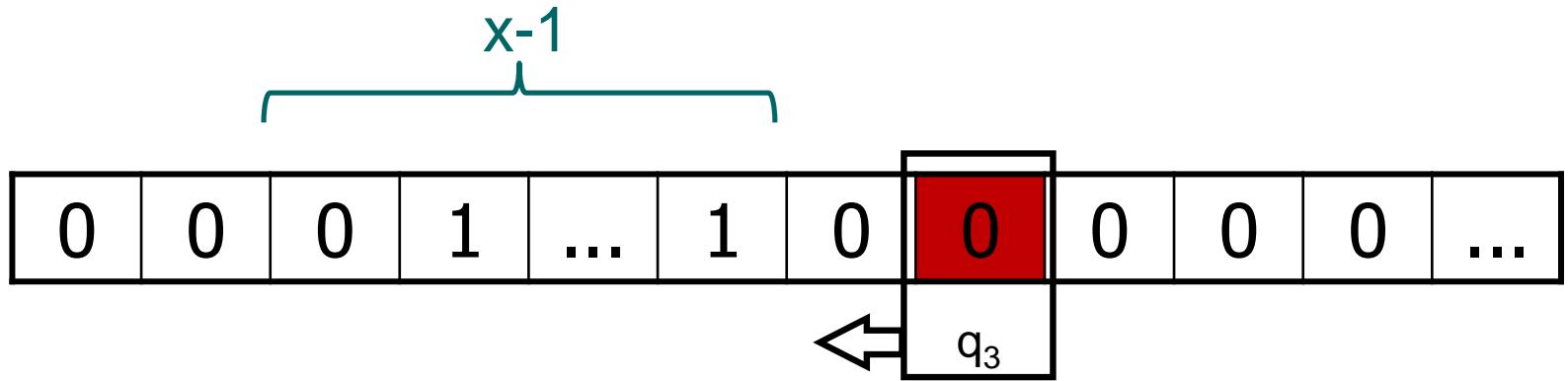
MT: Ejemplo 2X



- Dejamos un 0 para “recordar” dónde acaba x y comienza su copia
- Copiaremos el 1 a la derecha

$$T(q_2, 0) = (0, D, q_3)$$

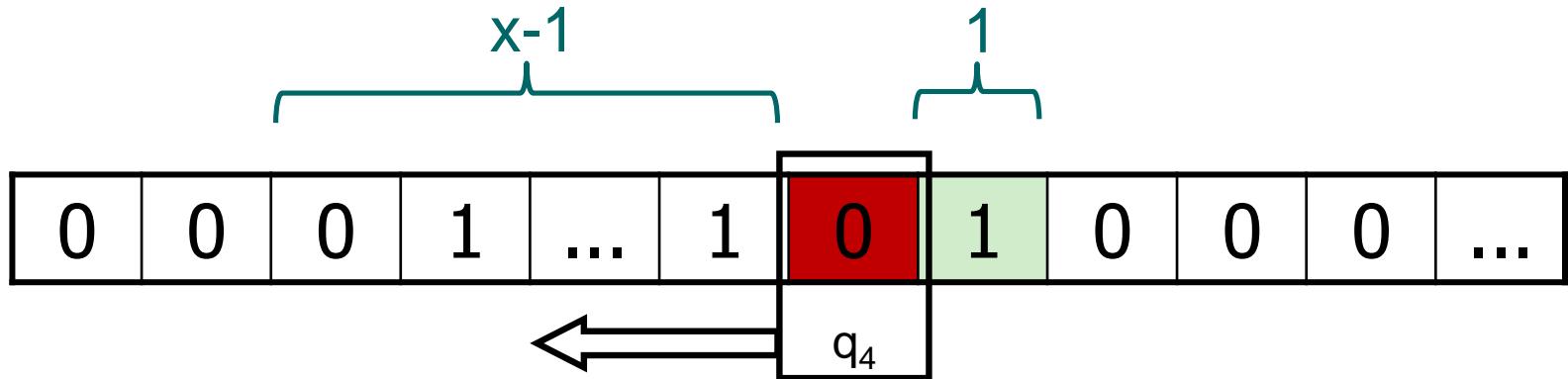
MT: Ejemplo 2X



- Escribimos un 1
- Volvemos a la izda para buscar el siguiente 1 a copiar

$$T(q_3, 0) = (1, l, q_4)$$

MT: Ejemplo 2X

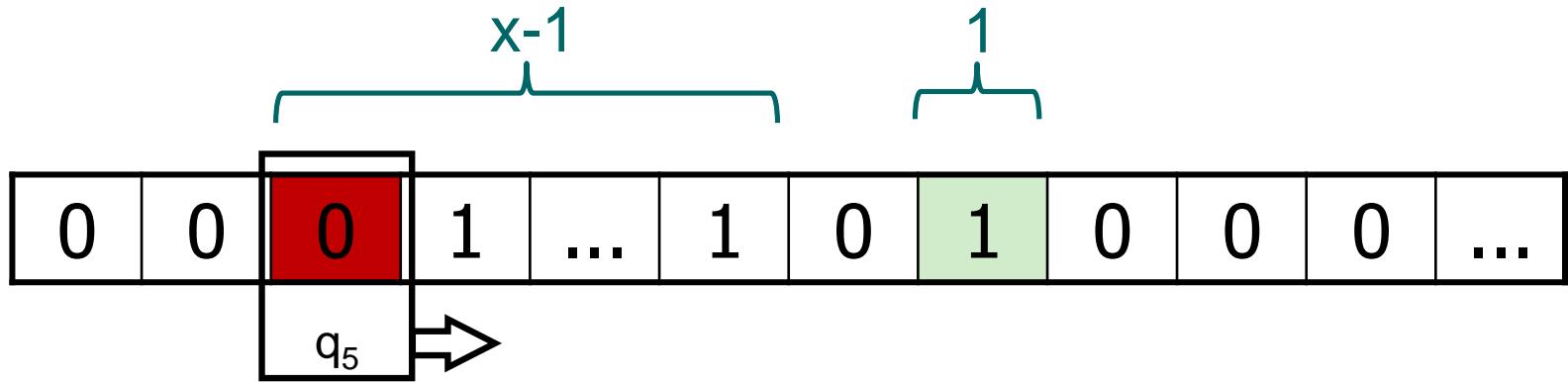


- Pasamos por encima del delimitador
- Buscamos la “marca” que dejamos (0) en el ultimo valor copiado

$$T(q_4, 0) = (0, l, q_5)$$

$$T(q_5, 1) = (1, l, q_5)$$

MT: Ejemplo 2X

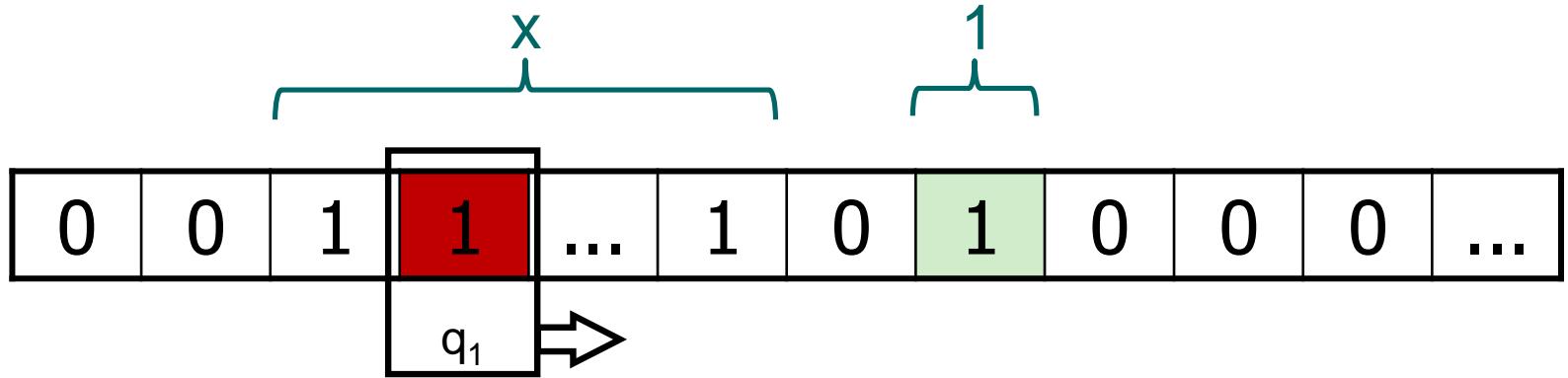


- Quitamos la marca (devolvemos el valor 1)
- Hemos marcado un 1, lo hemos duplicado y lo hemos restaurado. Lo que resta es repetir el proceso hasta que todos estén duplicados

$$T(q_5, 0) = (1, D, q_1)$$

Entramos en bucle:
Repetimos todos los
pasos anteriores

MT: Ejemplo 2X



$$T(q_0, 1) = (0, D, q_1)$$

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

$$T(q_2, 0) = (0, D, q_3)$$

$$T(q_3, 0) = (1, I, q_4)$$

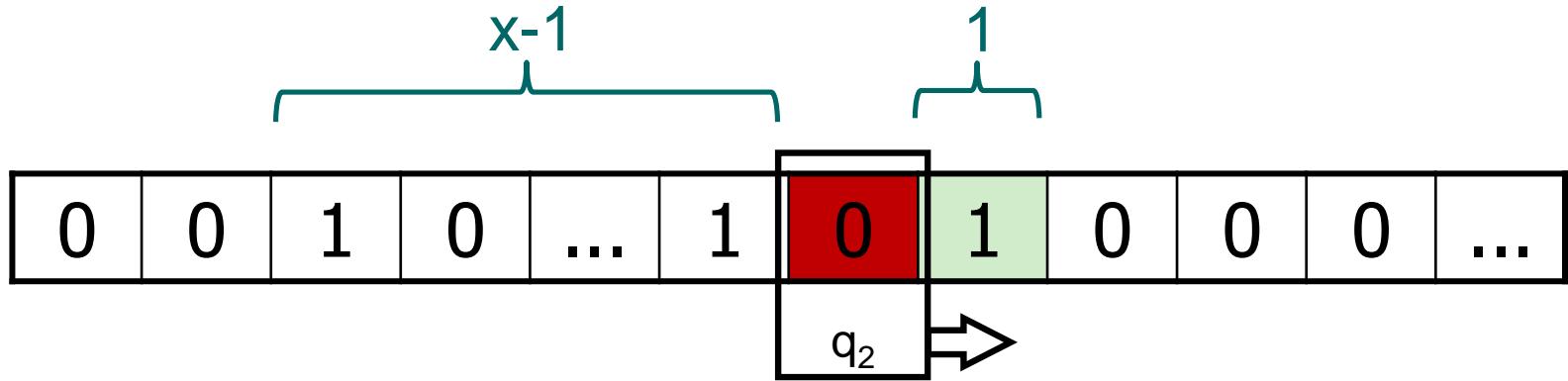
$$T(q_4, 0) = (0, I, q_5)$$

$$T(q_5, 1) = (1, I, q_5)$$

$$T(q_5, 0) = (1, D, q_1)$$

- Marcamos este 1 para duplicar
- Nos movemos hacia la zona de 0's para “copiarlo”

MT: Ejemplo 2X



$$T(q_0, 1) = (0, D, q_1)$$

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

$$\textcolor{red}{T(q_2, 0) = (0, D, q_3)}$$



$$T(q_3, 0) = (1, I, q_4)$$

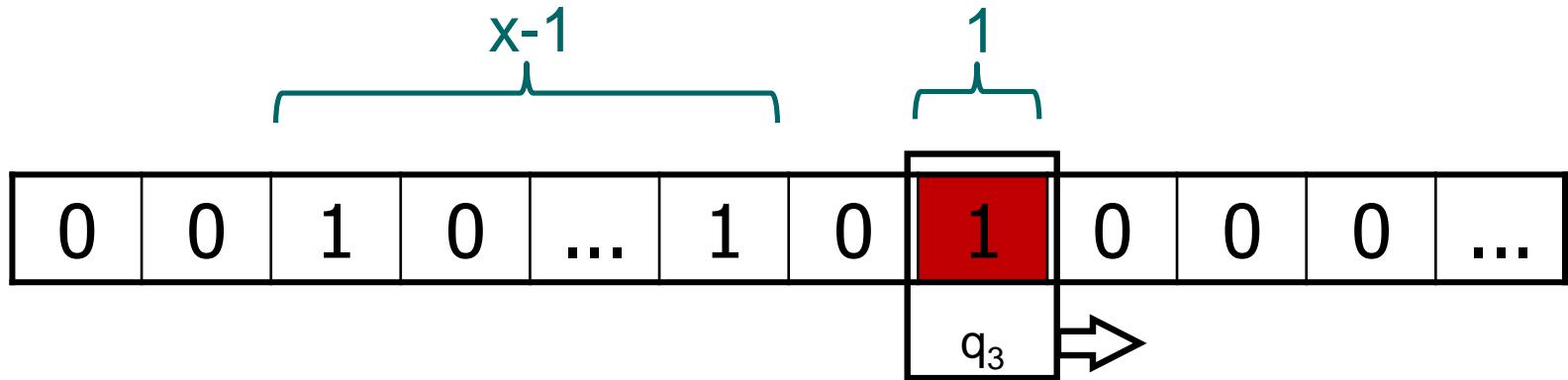
$$T(q_4, 0) = (0, I, q_5)$$

$$T(q_5, 1) = (1, I, q_5)$$

$$T(q_5, 0) = (1, D, q_1)$$

- Dejamos un 0 para “recordar” dónde acaba x y comienza su copia
- Copiaremos el 1 a la derecha

MT: Ejemplo 2X



$$T(q_0, 1) = (0, D, q_1)$$

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

$$T(q_2, 0) = (0, D, q_3)$$

$$T(q_3, 0) = (1, I, q_4)$$

$$T(q_4, 0) = (0, I, q_5)$$

$$T(q_5, 1) = (1, I, q_5)$$

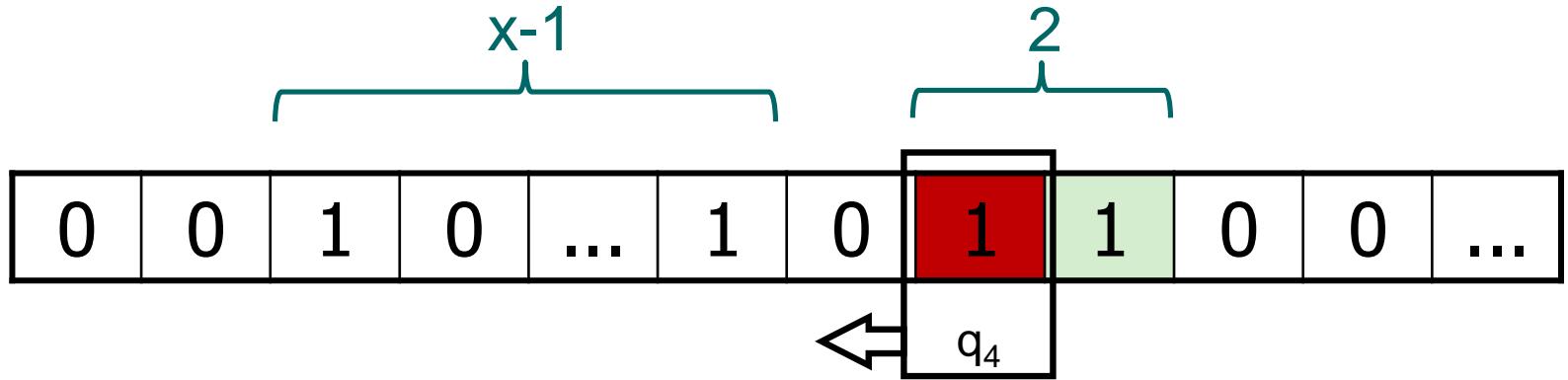
$$T(q_5, 0) = (1, D, q_1)$$

- Esta situación es nueva
- Buscamos el primer 0 “libre” para duplicar el 1
- Duplicamos el 1



$$T(q_3, 1) = (1, D, q_3)$$

MT: Ejemplo 2X



$$T(q_0, 1) = (0, D, q_1)$$

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

$$T(q_2, 0) = (0, D, q_3)$$

$$T(q_3, 0) = (1, I, q_4)$$

$$\mathbf{T(q_4, 0) = (0, I, q_5)}$$

$$T(q_5, 1) = (1, I, q_5)$$

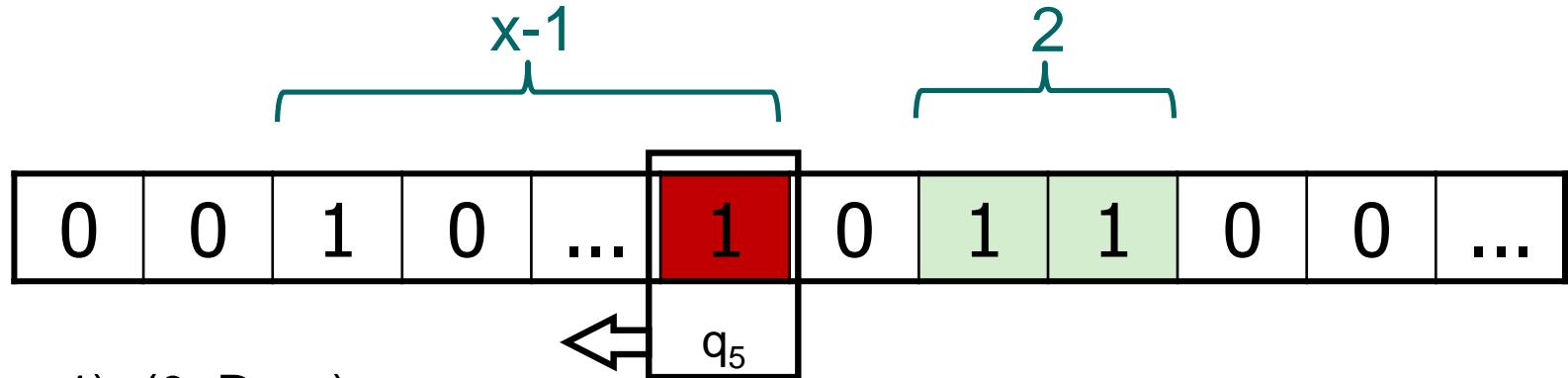
$$T(q_5, 0) = (1, D, q_1)$$

$$T(q_3, 1) = (1, D, q_3)$$

- Volvemos a la izda para buscar el siguiente 1 a copiar
- Hay que recorrer todos los 1's ya copiados
- Saltamos el delimitador

$$\mathbf{T(q_4, 1) = (1, I, q_4)}$$

MT: Ejemplo 2X



$$T(q_0, 1) = (0, D, q_1)$$

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

$$T(q_2, 0) = (0, D, q_3)$$

$$T(q_3, 0) = (1, I, q_4)$$

$$T(q_4, 0) = (0, I, q_5)$$

$$\textcolor{red}{T(q_5, 1) = (1, I, q_5)}$$

$$\textcolor{red}{T(q_5, 0) = (1, D, q_1)}$$

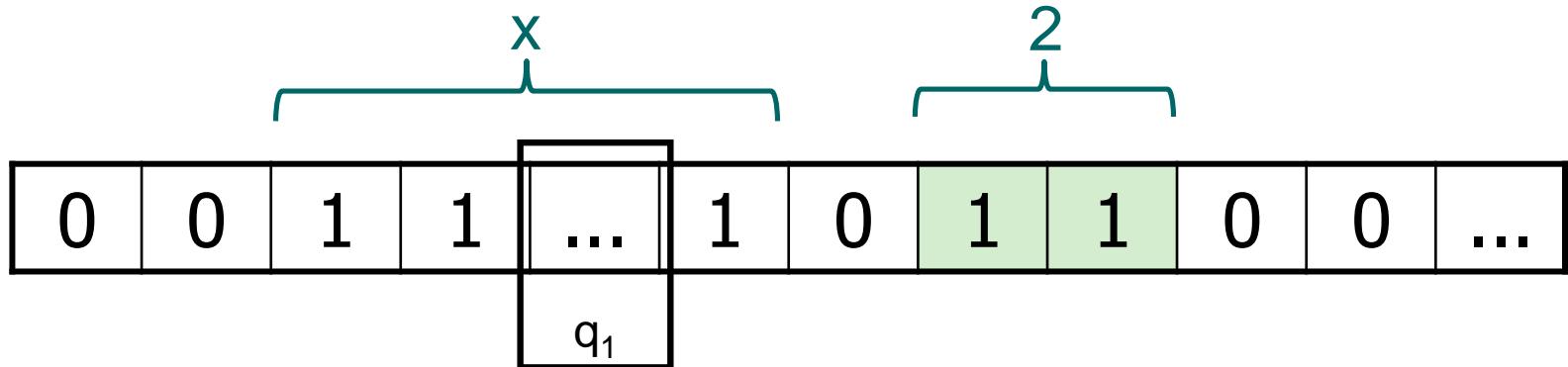
$$T(q_3, 1) = (1, D, q_3)$$

$$T(q_4, 1) = (1, I, q_4)$$

- Buscamos la “marca” que dejamos (0) en el ultimo valor copiado
- Quitamos la marca
- Repetimos el proceso



MT: Ejemplo 2X



$$T(q_0, 1) = (0, D, q_1)$$

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

$$T(q_2, 0) = (0, D, q_3)$$

$$T(q_3, 0) = (1, I, q_4)$$

$$T(q_4, 0) = (0, I, q_5)$$

$$T(q_5, 1) = (1, I, q_5)$$

$$T(q_5, 0) = (1, D, q_1)$$

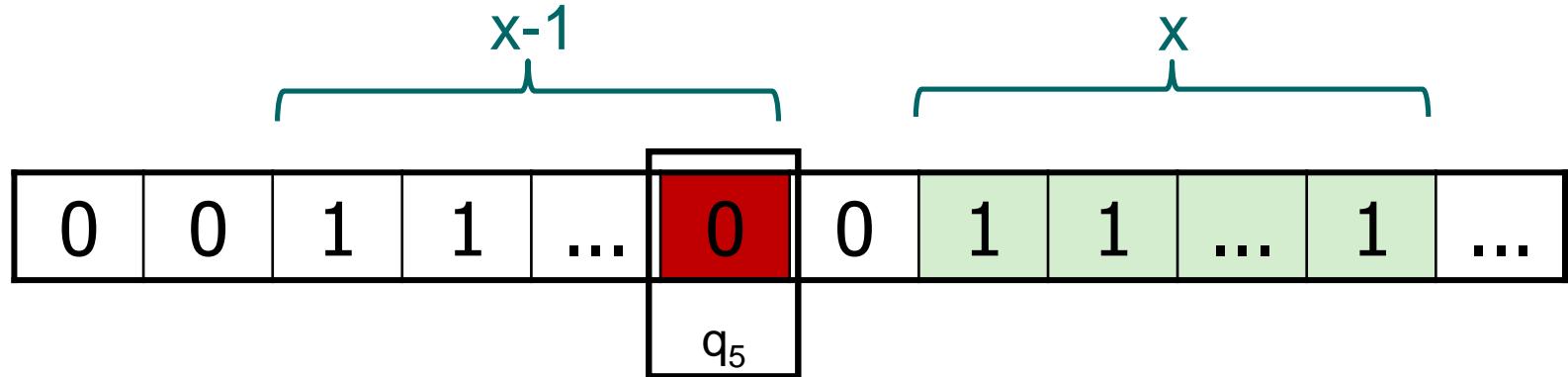
$$T(q_3, 1) = (1, D, q_3)$$

$$T(q_4, 1) = (1, I, q_4)$$

➤ ¿Cuándo se detiene el proceso?

- Veamos que ocurre cuando se acaban de copiar todos los 1's de la izda

MT: Ejemplo 2X

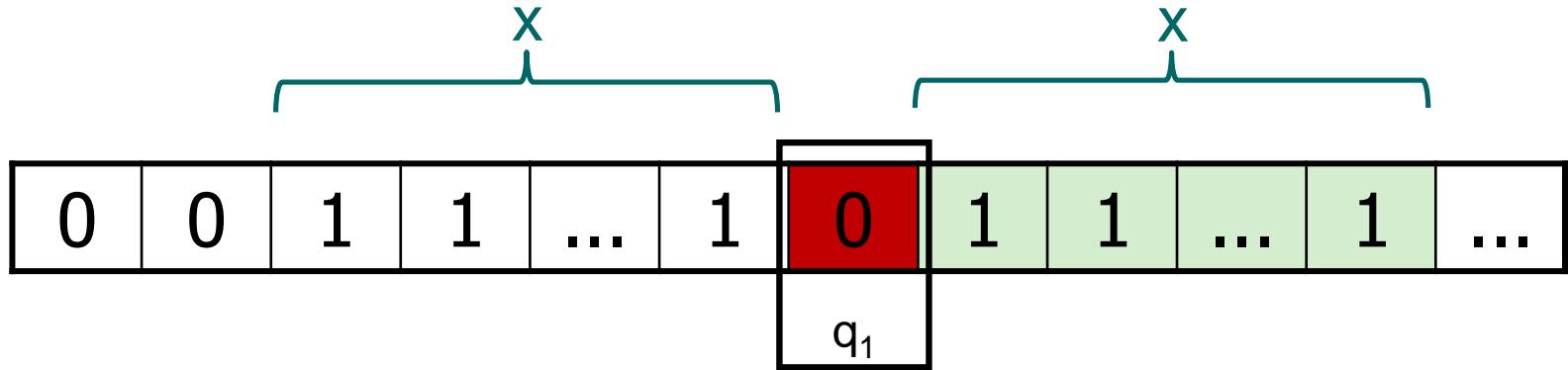


- $T(q_0, 1) = (0, D, q_1)$
- $T(q_1, 1) = (0, D, q_2)$
- $T(q_2, 1) = (1, D, q_2)$
- $T(q_2, 0) = (0, D, q_3)$
- $T(q_3, 0) = (1, I, q_4)$
- $T(q_4, 0) = (0, I, q_5)$
- $T(q_5, 1) = (1, I, q_5)$
- $T(q_5, 0) = (1, D, q_1)$**
- $T(q_3, 1) = (1, D, q_3)$
- $T(q_4, 1) = (1, I, q_4)$

➤ ¿Cuándo se detiene el proceso?

- Veamos que ocurre cuando se acaban de copiar todos los 1's de la izda
- El ultimo 1 de x ha sido marcado y copiado.
- La máquina, en estado q_5 , trata de eliminar la marca y comenzar una nueva iteración

MT: Ejemplo 2X



$$T(q_0, 1) = (0, D, q_1)$$

$$T(q_1, 1) = (0, D, q_2)$$

$$T(q_2, 1) = (1, D, q_2)$$

$$T(q_2, 0) = (0, D, q_3)$$

$$T(q_3, 0) = (1, I, q_4)$$

$$T(q_4, 0) = (0, I, q_5)$$

$$T(q_5, 1) = (1, I, q_5)$$

$$\textcolor{red}{T(q_5, 0) = (1, D, q_1)}$$

$$T(q_3, 1) = (1, D, q_3)$$

$$T(q_4, 1) = (1, I, q_4)$$

➤ ¿Cuándo se detiene el proceso?

- La máquina se encuentra en una nueva situación ($T(q_1, 0)$)
- La máquina termina
- La cinta tiene exactamente **2x 1's**

Opción 1: Definir q_1 como estado final

Opción 2: Definir $T(q_1, 0) = (0, H, q_f)$ con q_f estado final

MT: Ejemplo 2X



- Construye una MT \mathbf{M} , tal que $\varphi_{\mathbf{M}}^{(1)}(x) = 2x$ y haz la secuencia de computación para $\varphi_{\mathbf{M}}^{(1)}(3)$.

$$\mathbf{M} = (\{0,1\}, \{q_0, q_1, q_2, q_3, q_4, q_5, q_f\}, T, q_0, \{q_f\})$$

$$\begin{array}{llll} (q_0, 1, 0, D, q_1) & (q_2, 1, 1, D, q_2) & (q_3, 1, 1, D, q_3) & (q_5, 1, 1, I, q_5) \\ (q_1, 1, 0, D, q_2) & (q_2, 0, 0, D, q_3) & (q_4, 0, 0, I, q_5) & (q_5, 0, 1, D, q_1) \\ (q_1, 0, 0, H, q_f) & (q_3, 0, 1, I, q_4) & (q_4, 1, 1, I, q_4) & \end{array}$$

Secuencia de computación:

$$\begin{aligned} (\textcolor{red}{q_0}1111) &\rightarrow (\textcolor{red}{0}\textcolor{blue}{q_1}111) \rightarrow (\textcolor{red}{0}\textcolor{blue}{q_2}11) \rightarrow (01\textcolor{red}{q_2}1) \rightarrow (011\textcolor{red}{q_2}0) \rightarrow (0110\textcolor{red}{q_3}0) \rightarrow \\ &\rightarrow (011\textcolor{red}{q_4}01) \rightarrow (01\textcolor{red}{q_5}101) \rightarrow (0\textcolor{blue}{q_5}1101) \rightarrow (\textcolor{red}{q_5}01101) \rightarrow (\textcolor{red}{1}\textcolor{blue}{q_1}1101) \rightarrow \\ &\rightarrow (\textcolor{red}{1}0\textcolor{blue}{q_2}101) \rightarrow (101\textcolor{red}{q_2}01) \rightarrow (1010\textcolor{red}{q_3}1) \rightarrow (10101\textcolor{red}{q_3}0) \rightarrow (1010\textcolor{red}{q_4}11) \rightarrow \\ &\rightarrow (101\textcolor{red}{q_4}011) \rightarrow (10\textcolor{blue}{q_5}1011) \rightarrow (1\textcolor{blue}{q_5}01011) \rightarrow (11\textcolor{red}{q_1}1011) \rightarrow (110\textcolor{red}{q_2}011) \rightarrow \\ &\rightarrow (1100\textcolor{red}{q_3}11) \rightarrow (11001\textcolor{red}{q_3}1) \rightarrow (110011\textcolor{red}{q_3}0) \rightarrow (11001\textcolor{red}{q_4}11) \rightarrow \\ &\rightarrow (1100\textcolor{red}{q_4}111) \rightarrow (110\textcolor{blue}{q_4}0111) \rightarrow (11\textcolor{blue}{q_5}00111) \rightarrow (111\textcolor{red}{q_1}0111) \rightarrow (111\textcolor{red}{q_f}0111) \end{aligned}$$

Contenidos

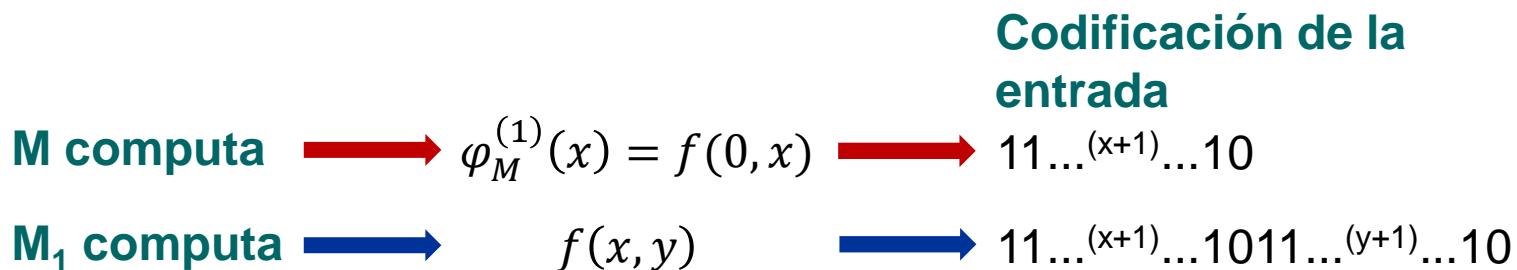
- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

Composición de MT

- ¿Qué ocurre si queremos usar una MT, $M_1 = (\{0,1\}, Q_1, T_1, i_1, F_1)$, para construir otra MT M ?
- Para calcular la función $\varphi_{M_1}^{(j)}(x_1, \dots, x_n)$ desde M , debemos:
 - Crear la MT M tal que $M = (\{0,1\}, Q \cup Q_1, T \cup T_1, i, F)$
 - $Q \cap Q_1 = \emptyset$.
 - No se definen transiciones contradictorias. T y T_1 no deben asignar valores distintos para el mismo par (estado, símbolo)
 - Preparar el input (x_1, \dots, x_n) en la cinta
 - Colocar la cabeza en el primer 1 que codifica x_1
 - Poner M en el estado inicial de M_1 .

Composición de MT: Ejercicio I

- Tenemos una MT, $M_1 = (\{0,1\}, \{q_0, q_1, q_2, q_3\}, T_1, q_0, \{q_3\})$, que calcula una función $f(x, y)$. Construye otra MT, M , cuya función semántica unaria sea $\varphi_M^{(1)}(x) = f(0, x)$.



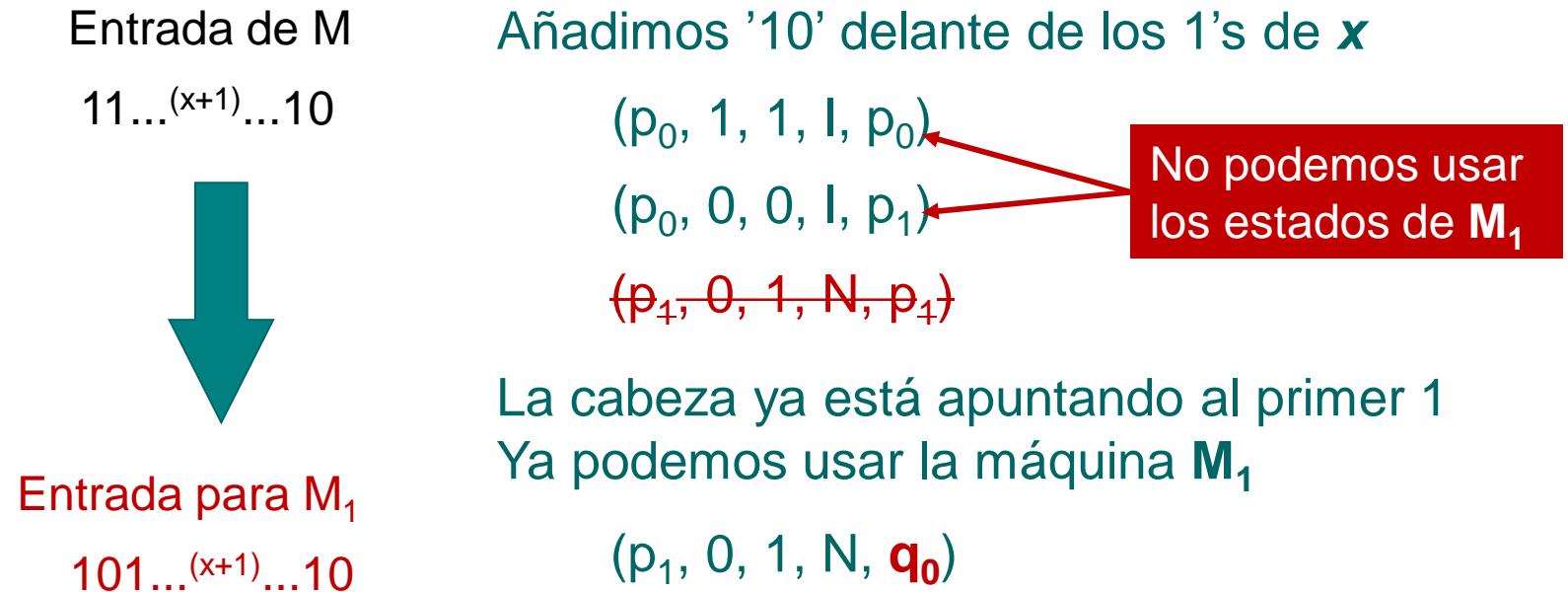
Para computar la salida de M , necesitamos usar M_1 con entrada $(0,x)$

Definir las transiciones que convierten la entrada de M en la entrada $(0,x)$ para M_1



Composición de MT: Ejercicio I

- Tenemos una MT, $M_1 = (\{0,1\}, \{q_0, q_1, q_2, q_3\}, T_1, q_0, \{q_3\})$, que calcula una función $f(x, y)$. Construye otra MT, M , cuya función semántica unaria sea $\varphi_M^{(1)}(x) = f(0, x)$.



Composición de MT: Ejercicio I

- Tenemos una MT, $\mathbf{M}_1 = (\{0,1\}, \{q_0, q_1, q_2, q_3\}, T_1, q_0, \{q_3\})$, que calcula una función $f(x, y)$. Construye otra MT, \mathbf{M} , cuya función semántica unaria sea $\varphi_M^{(1)}(x) = f(0, x)$.

Por tanto, la MT \mathbf{M} con $\varphi_M^{(1)}(x) = f(0, x)$ se define como:

$\mathbf{M} = (\{0,1\}, \{p_0, p_1, q_0, q_1, q_2, q_3\}, T \cup \mathbf{T}_1, p_0, \{q_3\})$, donde T es:

$(p_0, 1, 1, I, p_0)$

$(p_0, 0, 0, I, p_1)$

$(p_1, 0, 1, N, q_0)$

Composición MT: Ejercicio II



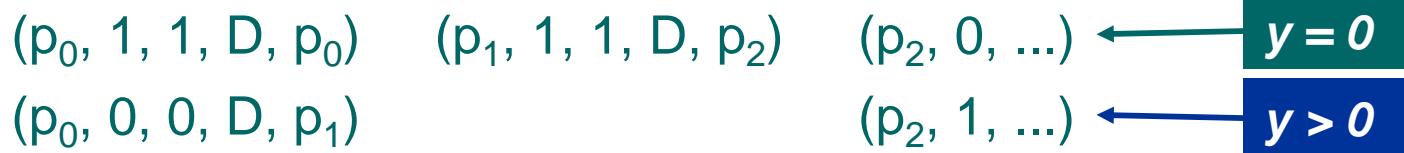
- Tenemos una MT, $M_1 = (\{0,1\}, \{q_0, q_1, q_2, q_3\}, T_1, q_0, \{q_3\})$, que calcula una función $g(x, y)$. Construye otra MT, M , cuya función semántica binaria sea:

$$\varphi_M^{(2)}(x, y) = \begin{cases} x + 1 & \text{si } y = 0 \\ g(x, y) & \text{si } y > 0 \end{cases}$$

M computa → $\varphi_M^{(2)}(x, y)$ → $11\dots^{(x+1)}\dots1011\dots^{(y+1)}\dots10$

Para computar la salida de M, lo primero es comprobar si $y=0$ ó $y>0$:

- Sin alterar el input, movemos la cabeza a la zona de 1's de y
- Saltamos el delimitador para ver si y vale 0 o no



Composición MT: Ejercicio II



$$M_1 = (\{0,1\}, \{q_0, q_1, q_2, q_3\}, T_1, q_0, \{q_3\}) \longrightarrow g(x, y)$$

$$M = (\{0,1\}, Q \cup Q_1, T \cup T_1, p_0, F) \longrightarrow \varphi_M^{(2)}(x, y) = \begin{cases} x + 1 & \text{si } y = 0 \\ g(x, y) & \text{si } y > 0 \end{cases}$$

$(p_0, 1, 1, D, p_0)$ $(p_0, 0, 0, D, p_1)$ $(p_1, 1, 1, D, p_2)$

$y = 0$

Debemos dejar $x+1$ 1's en la cinta.

Eliminamos el 1 que delimita y

$(p_2, 0, 0, I, p_3)$

$(p_3, 1, 0, N, p_f)$

$y > 0$

La cinta está lista para usar M_1 .

Movemos la cabeza al principio de x

$(p_2, 1, 1, I, p_4)$

$(p_5, 1, 1, I, p_5)$

$(p_4, 1, 1, I, p_4)$

$(p_5, 0, 0, D, q_0)$

$(p_4, 0, 0, I, p_5)$

Composición MT: Ejercicio II



- Tenemos una MT, $\mathbf{M}_1 = (\{0,1\}, \{q_0, q_1, q_2, q_3\}, T_1, q_0, \{q_3\})$, que calcula una función $g(x, y)$. Construye otra MT, \mathbf{M} , cuya función semántica binaria sea:

$$\varphi_M^{(2)}(x, y) = \begin{cases} x + 1 & \text{si } y = 0 \\ g(x, y) & \text{si } y > 0 \end{cases}$$

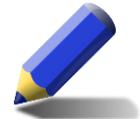
Por tanto, la MT \mathbf{M} se define como:

$\mathbf{M} = (\{0,1\}, \{p_0, p_1, p_2, p_3, p_4, p_5, p_f, q_0, q_1, q_2, q_3\}, T \cup \mathbf{T}_1, p_0, \{p_f, q_3\})$, con T :

$(p_0, 1, 1, D, p_0)$	$(p_2, 0, 0, I, p_3)$	$(p_4, 1, 1, I, p_4)$
$(p_0, 0, 0, D, p_1)$	$(p_3, 1, 0, N, p_f)$	$(p_4, 0, 0, I, p_5)$
$(p_1, 1, 1, D, p_2)$	$(p_2, 1, 1, I, p_4)$	$(p_5, 1, 1, I, p_5)$

$(p_5, 0, 0, D, q_0)$

Composición MT: Ejercicio III



- Tenemos una MT, $M_1 = (\{0,1\}, Q_1, T_1, q_0, F_1)$, que calcula una función $g(x, y)$. Construye otra MT, M , cuya función semántica binaria sea $\varphi_M^{(2)}(x, y) = g(2x, y)$

Entrada de M

$11 \dots^{(x+1)} \dots 1011 \dots^{(y+1)} \dots 10$



Entrada para M_1

$1 \dots^{(2x+1)} \dots 101 \dots^{(y+1)} \dots 10$

Para computar la salida de M , debemos:

- Añadir x 1's a la izda de la cinta
- No puede haber separación entre los nuevos 1's y los 1's que codifican x

Composición MT: Ejercicio III



$$M_1 = (\{0,1\}, Q_1, T_1, q_0, F_1) \longrightarrow g(x, y)$$

$$M = (\{0,1\}, \{p_0, p_1, p_2, p_3, p_4\} \cup Q_1, T \cup T_1, p_0, F_1) \text{ con } T:$$

Preparamos la cinta

- ($p_0, 1, 1, D, p_1$) Saltamos el 1 que delimita x
- ($p_1, 1, 0, I, p_2$) Marcamos un 1 para copiar
- ($p_2, 1, 1, I, p_2$) Buscamos el primer 0 a la izquierda para copiar el 1
- ($p_2, 0, 1, D, p_3$) Copiamos el 1
- ($p_3, 1, 1, D, p_3$) Volvemos a buscar la marca
- ($p_3, 0, 1, D, p_1$) Quitamos la marca y repetimos el proceso

Ponemos la cabeza en el primer 1 y usamos M_1

- ($p_1, 0, 0, I, p_4$) Saltamos el separador
- ($p_4, 1, 1, I, p_4$) Pasamos sobre todos los 1's
- ($p_4, 0, 0, D, q_0$) Dejamos la cabeza sobre el primer 1 y pasamos a q_0

Contenidos

- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

Equivalencia MT – Programas While

- **Teorema:** Toda función Turing-computable es While-computable.

Demostración:

Simular el funcionamiento de una MT mediante un PW:

- Símbolos: Codificados con números
- Estados: Codificados con números
- Acciones: Codificadas con números
- Instrucciones: Cinco números
- Contenido de la cinta: Dos listas; una para lo que está a la izquierda de la cabeza y otra para lo que está a la derecha
- Inicializar variables y estructuras
- Operar según instrucciones y contenido de la cinta
- Detectar llegada a estado final
- Sumar número de 1's y colocar valor en X1

Equivalencia MT – Programas While

- **Teorema:** Toda función While-computable es Turing-computable.

Demostración:

Simular la ejecución de un PW mediante una MT:

- Cada variable será un bloque de 1's en la cinta
- Mantener siempre bloques de 1's separados por un solo 0
- Sentencia básica $X_i := 0$:
 - Hacer 0 el bloque i (convertirlo en un sólo 1)
 - Desplazar los otros bloques para que la separación sea un solo 0
- Sentencias básicas $X_i := \text{succ}(X_k)$ y $X_i := \text{pred}(X_k)$
 - Sustituir el bloque i-ésimo por el successor/predecesor del k-ésimo
 - Desplazar para crear espacio o para reducir separación

Equivalencia MT – Programas While

- **Teorema:** Toda función While-computable es Turing-computable.

Demostración:

Simular la ejecución de un PW mediante una MT:

- Sentencias compuestas (***begin* $\delta_1; \delta_2; \dots; \delta_n$ *end***)
 - Componer MT's para $\delta_1, \delta_2, \dots, \delta_n$
 - Cambiar nombres de estados
 - Cambiar estados finales/iniciales
- ***while X_i ≠ X_k do δ***:
 - Comparar X_i y X_k
 - Si son distintos: Concatenar con una MT para δ
 - Repetir hasta que sean iguales
- Finalmente, borrar 1's sobrantes
 - Todas los bloques excepto X_1 , y el delimitador de X_1

Otros Modelos de Computación

- Existen muchos modelos de computación:

- Programas While
- Máquinas de Turing
- Máquinas de Turing multicinta
- Máquinas de Turing multidimensionales
- Máquinas de Turing no deterministas
- Cálculo lambda (Church y Kleene)
- Máquinas de registros (URM)
- Sistemas de Post
- Funciones Parciales Recursivas
- Autómatas Celulares
- Lógica combinatoria (Haskell Curry)
- Algoritmos de Markov
- Autómatas finitos con dos pilas
- Gramáticas de Tipo 0
- ...

**¡ Todos ellos son
equivalentes entre
sí !**

Contenidos

- Programas While
 - Modelo de los Programas While
 - Sentencias
 - Macros y macro-tests
 - Sentencias estructuradas
 - Funciones While Computables
 - Composición de Programas While
- Máquinas de Turing
 - Definición de Máquina de Turing
 - Definición formal
 - Secuencia de computación
 - Funciones Turing Computables
 - Composición de Máquinas de Turing
- Equivalencia entre PW y MT
- Tesis de Church

Tesis de Church

- Teniendo en cuenta los modelos vistos, nuestra idea de algoritmo y la experiencia informática, podríamos saber si una función o una tarea es computable, sin necesidad de escribir un programa específico.
- En la década de los años 30, esto se formuló como la tesis de Church:

Todo algoritmo o procedimiento efectivo
es computable

“Every effectively calculable function is a computable function”

Tesis de Church

- ¿Por qué se conoce como un Tesis y no como un teorema?
 - Realmente no existen los medios necesarios para que sea probada, ya que “algoritmo” o “procedimiento efectivo” no están perfectamente definidos en Teoría Matemática alguna, a partir de la cual extraer razonamiento demostrativo.
 - Cuando se intenta, se termina por admitir que procedimiento efectivo y programa es lo mismo.

Tesis de Church

■ Implicaciones filosóficas

- ¿Es la mente una máquina de Turing?
- ¿Está nuestro conocimiento limitado?

■ Implicaciones físicas

- ¿Es el universo una máquina de Turing (o un autómata celular)?
- ¿Hay procesos físicos que no sean simulables mediante MT?
- ¿Los podemos utilizar para hacer cálculos?