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

Etimología "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 de 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)
```

- Considerando todo lo anterior, 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. Posible 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 resolubilidad 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
  - Algo de Historia
  - 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 de letras y números que comienzan con mayúscula
  - ***Ejemplos***: *X1, X2, 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 := succ(Y)***
  - ***X := pred(Y)*** ← **¡OJO! *pred(0) = 0***
- **Sentencias while**:
  - ***while X ≠ Y do δ***
    - siendo $\boldsymbol{\delta}$ una sentencia cualquiera
    - $\boldsymbol{X \neq Y}$ es el test, y $\boldsymbol{\delta}$ es el cuerpo
- **Sentencia compuesta**:
  - ***begin*** $\boldsymbol{\delta_1}$; $\boldsymbol{\delta_2}$; ...; $\boldsymbol{\delta_n}$ ***end***
    - siendo $\boldsymbol{\delta_i}$ sentencias arbitrarias y $\mathbf{n \geq 0}$

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
```

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***:

$Z := X + Y$	$Z := X$
$Z := X \dot{-} Y$	$Z := X*Y$
$Z := X \text{ div } Y$	$Z := X \text{ mod } Y$
$Z := X**Y$	$Z := f(X_1, \ldots, X_m)$; *siendo f una función*

PW: Macros (Ejemplo Z:=X)

¿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

Z:=X

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

2. Añadir el valor de Y a Z

Z := Z+Y

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

PW: Macros (Ejemplo Z:=X+Y)

¿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

Z:=X+Y

```
begin
  Z := X;
  Z := Z+Y
end
```

Macro sentencias empleadas:
Asignación (Z:=X) y suma (Z:=Z+Y)

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

PW: Macros (Ejemplo Z :=X∸Y)

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

La diferencia acotada se define

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

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 Z:=X*Y)

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

Z := X*Y

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

Macro de la suma

PW: Macros (Ejemplo Z:=X*Y)

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 Z:=X div Y)

Construye un programa while que calcule **Z := X div Y**. La única macro que se permite utilizar es la de la diferencia acotada

Z := X div Y

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

Macro de la diferencia acotada

PW: Macros (Ejemplo Z:=X div Y)

Construye un programa while que calcule **Z := X div Y**. No se permite utilizar macros.

Z := X div Y

```
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
```

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

PW: Macros (Ejemplo Z:=X mod Y)

Construye un programa while que calcule **Z := X mod Y**

Z := X mod Y

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

El módulo es lo que queda en W antes de que valga 0, luego ahora no interesa retornar el nº de veces que restamos (como en Z:= X div Y) sino ese valor (valor de W antes de restar la última vez).

Macro de la asignación

Macro de la diferencia acotada

PW: Macro-test

X≠Y es el único test permitido en las sentencias while

Cualquier otro test diferente a ***X≠Y*** *es un* ***Macro-test***

Definición: Macro-test

- Básico: ***X<Y*** (X e Y números naturales o variables)
- Inductivo: sean T_1, T_2 *test*

i. $T_1 \wedge T_2$

ii. $T_1 \vee T_2$

iii. $\neg T_i$ $\quad$ ***i=1, 2***

También son Macro-test

Todo Macro-test ***T*** involucrando variables puede construirse a partir de la definición anterior.

PW: Macro-test

Proposición: Una sentencia de la forma:

while T do δ,

con δ sentencia arbitraria y T un test diferente de ***X≠Y*** es una macro sentencia.

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

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

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

PW: Macro-test

Pero….Cómo construir *E_T para un T cualquiera?*

- Indicación: Usando la definición inductiva de Macro-test

$T = (X < Y)$	$E_T = (Y \dot{-} X)$
$T = T1 \wedge T2$	$E_T = E_{T1} * E_{T2}$
$T = T1 \vee T2$	$E_T = E_{T1} + E_{T2}$
$T = \neg T1$	$E_T = 1 \dot{-} E_{T1}$

PW: Macro-test (Ejemplos)

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

$$E_{(Z\geq Y)v(Z>X)} = E_{Z\geq Y} + E_{Z>X} =$$
$$= (1 \dot{-} E_{Z<Y}) + E_{X<Z} =$$
$$= (1 \dot{-} (Y \dot{-} Z)) + (Z \dot{-} X)$$

- **Construir E_T para $\mathbf{T} = (\mathbf{X} = \mathbf{Y})$**

$$E_{X=Y} = E_{(X\leq Y)\wedge(X\geq Y)} = E_{X\leq Y} * E_{X\geq Y} =$$
$$= (1 \dot{-} E_{Y<X}) * (1 \dot{-} E_{X<Y}) =$$
$$= (1 \dot{-} (X \dot{-} Y)) * (1 \dot{-} (Y \dot{-} X))$$

PW: Macro-test (Ejemplos)

Deshacer los macro-test:

- Se permiten las macros de resta, suma y asignación.

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

1º. Calculamos la E_T

$$E_{(X<Y)\vee(Y>=Z)} = E_{X<Y} + E_{\neg(Y<Z)} =$$
$$= (Y \dot{-} X)+(1 \dot{-} E_{Y<Z}) =$$
$$= (Y \dot{-} X)+(1 \dot{-} (Z \dot{-} Y))$$

```
begin
  U:=Y ∸ X; // E X<Y
  V:=1;
  W:=Z ∸ Y;       } // E ¬(Y<Z)
  V:=V ∸ W;
  U:=U+V; // E (X<Y)v(Y>=Z)
  V:=0;
  while U≠V do
  begin
    X:=succ(X);   } δ;
    Y:=pred(Y);
    U:=Y ∸ X;
    V:=1;         } Se recalcula E (X<Y)v(Y>=Z)
    W:=Z ∸ Y;       Pues X e Y cambian e
    V:=V ∸ W;       influyen en T
    U:=U+V;
    V:=0;         Se compara E (X<Y)v(Y>=Z)
  end             con 0, por eso V:=0
end
```

PW: Sentencias estructuradas

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

$$\textit{If } T \textit{ then } \delta_1$$

$$\textit{If } T \textit{ then } \delta_1 \textit{ else } \delta_2$$

$$\textit{Repeat } \delta \textit{ until } T$$

donde **T** es un test y δ, δ_1 y δ_2 son sentencias, es una **macro sentencia**

PW: Sentencias estructuradas

¿Cómo deshacer las macro sentencias?

If T then δ_1

begin
$U:=E_T$;
$V:=0$;
while $U \neq V$ do
begin
$\delta 1$;
$U:=0$;
end
end

δ_1 sólo se ejecuta cuando E_T distinto de 0 (>0), es decir cuando el test T se evalúa a cierto.

If T then δ_1 else δ_2

begin
$U:=E_T$;
$V:=1 \dot{-} E_T$;
$W:=0$;
while $U \neq W$ do
begin
δ_1 ;
$U:=0$;
end
while $V \neq W$ do
begin
δ_2 ;
$V:=0$;
end
end

δ_1 sólo se ejecuta cuando E_T distinto de 0 (>0), es decir cuando el test T se evalúa a cierto.
δ_2 sólo se ejecuta cuando $1 \dot{-} E_T$ es distinto de 0 (>0) lo que ocurre cuando E_T igual a 0, es decir cuanto el test T se evalúa a falso.

Repeat δ until T

begin
δ;
while $\neg$ T do
begin
δ ;
end
end

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

Sin macro test

begin
δ;
$U:=1 \dot{-} E_T$;
$V:=0$;
while $U \neq V$ do
begin
δ ;
$U:=1 \dot{-} E_T$;
end
end

δ se ejecuta al menos 1 vez y hasta que $1 \dot{-} E_T$ sea igual a 0 lo que ocurre cuando E_T es distinto de 0 (>0), es decir cuando 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
  - Algo de Historia
  - 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: Funciones While Computables

- **Definición.** Una **sentencia** δ (en particular un programa while) se dice ***k-variables***, si utiliza un subconjunto de las variables $\{X_1, X_2, \ldots, X_k\}$

- **Definición.** Un **vector estado de la computación** de un programa while k-variables es un vector

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

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

- **Definición.** 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 $\boldsymbol{\hat{a}_i}$ son vectores estado y $\boldsymbol{A_j}$ son instrucciones de P
  - Una instrucción puede ser una sentencia básica o un test.
- $\hat{a}_0$ 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 $\boldsymbol{n}$ 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
end
```

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

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≠X4
$\hat{a}_2$=(3,3,0,2)
A_3=X3:=succ(X3)
$\hat{a}_3$=(3,3,1,2)
A_4=X2:=pred(X2)
$\hat{a}_4$=(3,2,1,2)
A_5=X3≠X2
$\hat{a}_5$=(3,2,1,2)
A_6=X3:=succ(X3)
$\hat{a}_6$=(3,2,2,2)
A_7=X3≠X2
$\hat{a}_7$=(3,2,2,2)
A_8=X2≠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**, $\boldsymbol{\varphi}_{\mathbf{P}}^{(\mathbf{j})}: \mathbb{N}^{\mathbf{j}} \rightarrow \mathbb{N}$ (j>0) de un programa **k** variables **P** se define como sigue:
  - Dado un vector de entrada $\hat{a} = (a_1, \ldots, a_j)$, la función $\boldsymbol{\varphi}_{\mathbf{P}}^{(\mathbf{j})}(\mathbf{a_1}, \ldots, \boldsymbol{a_j})$ se evalúa según las reglas siguientes:

1. **k ≥ j**: $\varphi_{\mathrm{P}}^{(\mathrm{j})}(\mathrm{a}_1, \ldots, a_j)$ se obtiene aplicando **P** al vector estado inicial $\hat{a}_0 = (a_1, \ldots, a_j, 0, 0, \ldots^{(k-j)} \ldots, 0)$
2. **k < j**: $\varphi_{\mathrm{P}}^{(\mathrm{j})}(\mathrm{a}_1, \ldots, a_j)$ se obtiene aplicando **P** al vector estado inicial $\hat{a}_0 = (a_1, \ldots, a_{\boldsymbol{k}})$

  - Si **P** para con ese vector, el contenido final de **X1** es el valor de $\varphi_{\mathrm{P}}^{(\mathrm{j})}(\mathrm{a}_1, \ldots, a_j)$
  - En otro caso, $\varphi_{\mathrm{P}}^{(\mathrm{j})}(\mathrm{a}_1, \ldots, a_j) = \perp$ (indeterminado)

PW: Función Semántica

- **Nótese** que 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.
- **¡OJO!** 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.** Como **k=4**

Caso **k ≥ j**: Vector estado inicial $\boldsymbol{\hat{a}_0 = (x, y, 0, 0)}$

'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** **(k=3)**

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** **(k=3)**

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

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

¿Qué queda en X1 al final de la ejecución?

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

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** **(k=3)**

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

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

¿Qué queda en X1 al final de la ejecución?

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

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

j=1 $\varphi_P^{(1)}(x)$

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

```
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
```

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

Vector estado tras 1er bucle:

1er Bucle

Tras cada iteración i:

***â*=(x, 0, 0, x-i, i)**

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

Si x es par:

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

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

Si x es impar:

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

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

2º Bucle

La condición de entrada no se cumple (X3==0)

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

PW: Función Semántica. Ejemplo

j=1 $\varphi_P^{(1)}(x)$

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

```
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
```

Secuencia computación
Vector estado inicial: $\hat{a}_0$=**(3, 0, 0, 0, 0)**

$\hat{a}_0$=(3,0,0,0,0)
X4:=X1+X2;
$\hat{a}_1$=(3,0,0,3,0)
X4>X5
$\hat{a}_2$=(3,0,0,3,0)
X5:=succ(X5);
$\hat{a}_3$=(3,0,0,3,1)
X4:=pred(X4);
$\hat{a}_4$=(3,0,0,2,1)
X4>X5
$\hat{a}_5$=(3,0,0,2,1)
X5:=succ(X5);
$\hat{a}_6$=(3,0,0,2,2)
X4:=pred(X4);
$\hat{a}_7$=(3,0,0,1,2)
X4>X5
$\hat{a}_8$=(3,0,0,1,2)
X1:=X4;
$\hat{a}_9$=(1,0,0,1,2)
X3≠0
$\hat{a}_{10}$=(1,0,0,1,2)

$$\varphi^{(1)}(3) = 1$$

Secuencia computación
Vector estado inicial: $\hat{a}_0$=**(4, 0, 0, 0, 0)**

$\hat{a}_0$=(4,0,0,0,0)
X4:=X1+X2;
$\hat{a}_1$=(4,0,0,4,0)
X4>X5
$\hat{a}_2$=(4,0,0,4,0)
X5:=succ(X5);
$\hat{a}_3$=(4,0,0,4,1)
X4:=pred(X4);
$\hat{a}_4$=(4,0,0,3,1)
X4>X5
$\hat{a}_5$=(4,0,0,3,1)
X5:=succ(X5);
$\hat{a}_6$=(4,0,0,3,2)
X4:=pred(X4);
$\hat{a}_7$=(4,0,0,2,2)
X4>X5
$\hat{a}_8$=(4,0,0,2,2)
X1:=X4;
$\hat{a}_9$=(2,0,0,2,2)
X3≠0
$\hat{a}_{10}$=(2,0,0,2,2)

$$\varphi^{(1)}(4) = 2$$

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

PW: Función Semántica. Ejemplo

j=1 $\varphi_P^{(3)}(x,y,z)$

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

```
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
```

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

1er Bucle

Tras cada iteración i:

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

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

Vector estado tras 1er bucle:

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

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

Vector estado tras 2º bucle:

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

2º Bucle

Tras cada iteración i, X1 se incrementa en 1 unidad y X3 se decrementa en 1 unidad

El bucle acaba tras X3 iteraciones.

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

PW: Función Semántica. Ejemplo

j=3 $\varphi_P^{(3)}(x,y,z)$ Vector estado inicial: $\hat{a}_0$=**(x, y, z, 0, 0)**

```
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
```

Secuencia computación
Vector estado inicial: $\hat{a}_0$=**(3, 2, 1, 0, 0)**

$\hat{a}_0$=(3,2,1,0,0)
X4:=X1+X2;
$\hat{a}_1$=(3,2,1,5,0)
X4>X5
$\hat{a}_2$=(3,2,1,5,0)
X5:=succ(X5);
$\hat{a}_3$=(3,2,1,5,1)
X4:=pred(X4);
$\hat{a}_4$=(3,2,1,4,1)
X4>X5
$\hat{a}_5$=(3,2,1,4,1)
X5:=succ(X5);
$\hat{a}_6$=(3,2,1,4,2)
X4:=pred(X4);
$\hat{a}_7$=(3,2,1,3,2)
X4>X5
$\hat{a}_8$=(3,2,1,3,2)
X5:=succ(X5);
$\hat{a}_9$=(3,2,1,3,3)
X4:=pred(X4);
$\hat{a}_{10}$=(3,2,1,2,3)
X4>X5
$\hat{a}_{11}$=(3,2,1,2,3)
X1:=X4;
$\hat{a}_{12}$=(2,2,1,2,3)
X3≠0
$\hat{a}_{13}$=(2,2,1,2,3)
X1=succ(X1)
$\hat{a}_{14}$=(3,2,1,2,3)
X3:=pred(X3);
$\hat{a}_{15}$=(3,2,0,2,3)
X3≠0
$\hat{a}_{16}$=(3,2,0,2,3)

$$\varphi^{(3)}(3,2,1) = 3$$

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

PW: Funciones While-Computables

Definición

Una función $f : \mathbb{N}^j \to \mathbb{N}$ es **while-computable**, o simplemente computable, si:

$$\varphi^{(j)}(a_1, \dots, a_j) = f(a_1, \dots, a_j)$$

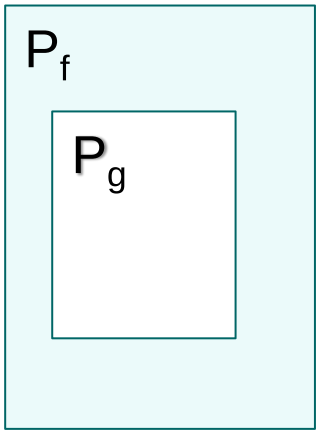
para algún programa while P y $\forall (a_1, \dots, a_j) \in \mathbb{N}^j$

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
  - Algo de Historia
  - 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?



- **Pf** computa una función de aridad j y es k1-variable:

  $$\hat{a}_0 = \left(a_1, \dots, a_j, 0, \dots^{(k1-j)} \dots, 0\right)$$

  - Deja su salida en **X1**
- **Pg** computa una función de aridad i y es k2-variable:

  $$\hat{a}_0 = \left(a_1, \dots, a_i, 0, \dots^{(k2-i)} \dots, 0\right)$$

  - Deja su salida en **X1**

- Debemos utilizar variables adicionales para asegurarnos que el vector de estado es en cada momento el que debe ser:
  - Antes de ejecutar **Pg**, debemos asegurarnos que el vector de estado sea el correcto.
  - Guardar la salida de **Pg**, en variables que no use **Pg**.

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:

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

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

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

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

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

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 $\mathbf{P_g}$

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

```
begin
  X3 := X1 ∸ X2;
  X2 := X1 + X2;
  X1 := X3;
  X3 := 0;
  …;
  Xk := 0;
  Pg;
  …
end
```

Antes de utilizar $\mathrm{P_g}$:

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

Después de utilizar $\mathrm{P_g}$:

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

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

Composición de PW: Ejemplo

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

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

```
begin
  Xk+1 := X1;
  Xk+2 := X3;
  X1 := X1 ∸ X2;
  X2 := X1 + X2;
  X3 := 0;
  …;
  Xk := 0;
  Pg;
  …
end
```

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

Antes de utilizar $\mathrm{P_g}$:

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

Después de utilizar $\mathrm{P_g}$:

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

Composición de PW: Ejemplo

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

- Nos preparamos para usar P_g por segunda vez

```
begin
  Xk+1 := X1;
  Xk+2 := X3;
  X1 := X1 ∸ X2;
  X2 := X1 + 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 Pg está en la variable X1. Lo guardamos para sumarlo al final

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

Sumamos $g(x \dot{-} y, x + y)$ (guardado en Xk+3) y $g(x \dot{-} z, x + z)$ (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 primera variable libre (Xk+1)
2. Preparar el vector estado inicial para P_g.
   - Si queremos calcular $\varphi_g^{(j)}(x_1, \ldots x_j)$, se colocan los valores correspondientes en las variables $X_1, X_2, \ldots, X_j$.
   - El resto de variables $(X_{j+1}, \ldots, X_k)$ 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:

Puedes emplear las macros: producto, suma y asignación

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

```
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
```

Guardamos x, y, z

El bucle computa el sumatorio de manera decreciente, comenzando desde i=Z (que está almacenado en X_{k+3})

Preparamos el vector estado inicial para P_g

Ejecutamos el código de P_g

Acumulamos y guardamos el resultado en una variable "segura"

Retornamos $\varphi_P^{(3)}(\mathrm{x,y,z})$ en X1

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, que vale >0 si es cierta y 0 si es falsa:

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

```
begin
  Xk+1 := X1;
  Xk+2 := X2;
  X1 := 0;
  X2 := Xk+2;
  P1;
  Xk+4 := X1;
  X2 := Xk+1;
  X3 := 0; …; Xk2 := 0;
  P2
  …
```

Guardamos x, y. La primera variable segura es Xk+1, con k = max{k1,k2}

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

Guardamos el resultado de P1

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

Composición de PW: Ejercicio II

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

```
begin
  Xk+1 := X1;
  Xk+2 := X2;
  X1 := 0;
  X2 := Xk+2;
  P1;
  Xk+4 := X1;
  X2 := Xk+1;
  X3 := 0; …; Xk2:=0;
  P2;
  …
```

```
 …
 while (X1 ≠ Xk+5) 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
```

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

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

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
  - Algo de Historia
  - 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.

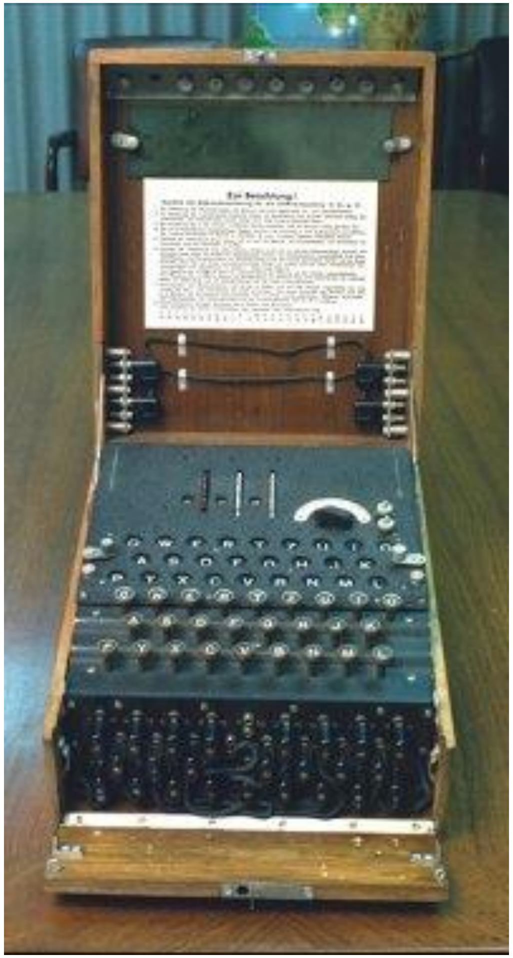
Algo de Historia: II Guerra Mundial

- Papel decisivo en descifrar los códigos 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 y en 1926 la armada alemana la adoptó para uso militar.
    - Ventaja: fácil manejo y cifrado difícil (supuestamente inviolable).
    - Los Polacos eran buenos descifrando códigos. Los Franceses compraron las claves, pero no pudieron hacer nada con ellas.
    - http://www.enigmaco.de/

Algo de Historia: II Guerra Mundial

- 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).
  - Basado en:
    - La velocidad y la fiabilidad de la tecnología electrónica.
    - La ineficiencia del diseño de máquinas diferentes para diferentes procesos lógicos.
- A partir de 1943 se pudieron descifrar los mensajes en Enigma.
- Churchill ordenó destruir todos los equipos

Algo de Historia: II Guerra Mundial



- Construcción primeros ordenadores programables: Colossus y Mark I.
- 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:

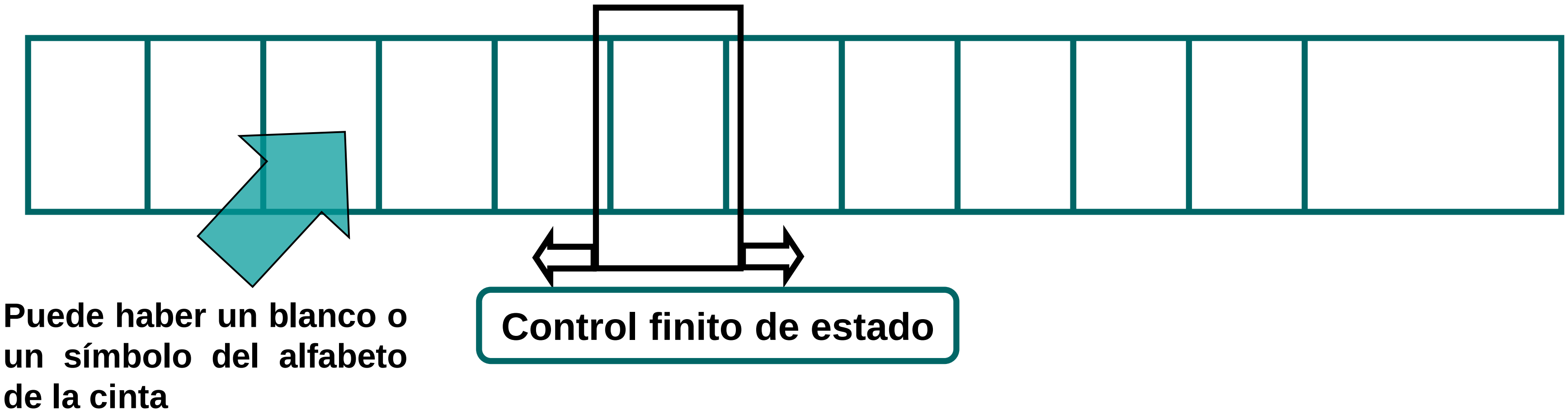
    Máquinas de Turing

- Tesis de Church-Turing

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
  - Algo de Historia
  - 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

- En la cinta se coloca el input codificado. La cabeza lectora/escritora se sitúa sobre el primer símbolo de la cadena de entrada y el estado de la máquina es uno 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.

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):
    - Escribir un símbolo en la cinta.
    - Cambiar de estado.
    - 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** (como si no hubiese parado nunca)
  - Sin **no para**, el resultado se considera **indefinido**

Definición Formal de una MT

Definición

Una MT es una quíntupla: **(X, Q, T, i, F)** donde:

- **Q** es un conjunto finito no vacío de estados.
- **$i \in Q$** es el estado inicial.
- **$F \subseteq Q$** es el conjunto de estados finales.
- **X** es un alfabeto de símbolos. Posee un símbolo especial o símbolo blanco (B).
- **T** : $Q \times X :\rightarrow X \times \text{Acciones} \times Q$

  Acciones = {I, D, N, H}

Secuencia de computación en MT

- **Patrón de una descripción instantánea:**

$$y_1 \dots y_n \, q \, x_1 \dots x_m$$

  - **q**: Estado de la máquina
  - x_1:Símbolo sobre el que se encuentra la cabeza
  - $x_2 \dots x_m$: Símbolos a la derecha de la cabeza, con
  - x_m último símbolo no blanco tras el símbolo x_1.
  - $y_1 \dots y_m$: Símbolos a la izquierda de la cabeza

    con y_1 primer símbolo no blanco.

- **Descripción inicial:** $i \, x_1 \dots x_m$

Secuencia de computación en MT

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

$$y_1 \ldots y_n\, q\, x_1 \ldots x_m$$

Depende de $T(q, x_1)$

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

  $$y_1 \ldots y_n\, q\, x_1 \ldots x_m \rightarrow y_1 \ldots y_n\, x'_1\, q'\, x_2 \ldots x_m$$

- Si $T(q, x_1) = (x'_1, q', I)$

  $$y_1 \ldots y_n\, q\, x_1 \ldots x_m \rightarrow y_1 \ldots y_{n-1}\, q' y_n\, x'_1\, x_2 \ldots x_m$$

- Si $T(q, x_1) = (x'_1, q', N)$

  $$y_1 \ldots y_n\, q\, x_1 \ldots x_m \rightarrow y_1 \ldots y_n\, q' x'_1\, x_2 \ldots x_m$$

Secuencia de computación en MT

- Si $T(q, x_1) = (x'_1, q', H)$

$$y_1 \dots y_n\, q\, x_1 \dots x_m \rightarrow y_1 \dots y_n\, q' x'_1\, x_2 \dots x_m$$

Casos **"especiales"**:

- $T(q, x_1)=(x'_1, q', I)$: $q\, x_1 \dots x_m \rightarrow q' B x'_1\, x_2 \dots x_m$
- $T(q, x_1)=(x'_1, q', D)$: $y_1 \dots y_n\, q\, x_1 \rightarrow y_1 \dots y_n\, x'_1 q' B$

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

$$i\, x_1 \dots x_m$$

Computación con una MT

Codificación de la entrada:

Notación unaria (X = {0,1})

- Dado $n \in N$, éste se codifica en la cinta con **(n+1) 1's consecutivos**.
- Dado un vector $(\mathbf{n_1, \ldots, n_k})$ su codificación será:

1 .. $(n_1 + 1)$ **.. 101..** $(n_2 + 1)$ **.. 10 … 01..** $(n_k + 1)$ **.. 1**

Computación con una MT

Definición. Función Semántica

Dada una MT, **M=(X, Q, T, i, F)**, se define la j-aria función semática de M como sigue:

$$\varphi_M^{(j)} : N^j \rightarrow N$$

$\varphi_M^{(j)}(x_1,\ldots,x_j)$**= número de 1's** (**consecutivos o no**) de la cinta, tras la computación de M con entrada la codificación del vector $(x_1,\ldots,x_j)$, si dicha computación para en un estado final; e **indefinido** en otro caso.

PW: Función Turing-Computable

Definición

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

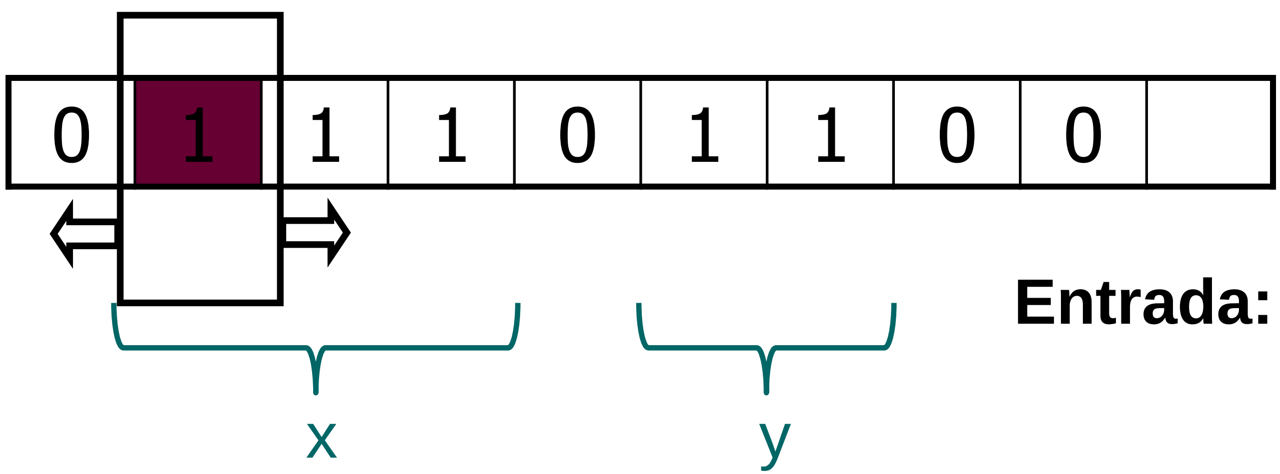
$$\varphi_M{}^{(j)}(x_1, \dots, x_j) = f(x_1, \dots, x_j) \quad \forall (x_1, \dots, x_j) \in \mathbb{N}^k$$

Ejemplo: MT para la función X+Y

Codificación de entrada (Notación unaria)

- Dado $n \in N$, se representa por un bloque de $(n+1)$ 1's consecutivos.
- Dado un vector $(n_1, \ldots, n_k)$, se codificará como

$$1 \,..\, (n_1+1) \,..\, 101 \,..\, (n_2+1) \,..\, 10 \ldots 01 \,..\, (n_k+1) \,..\, 1$$



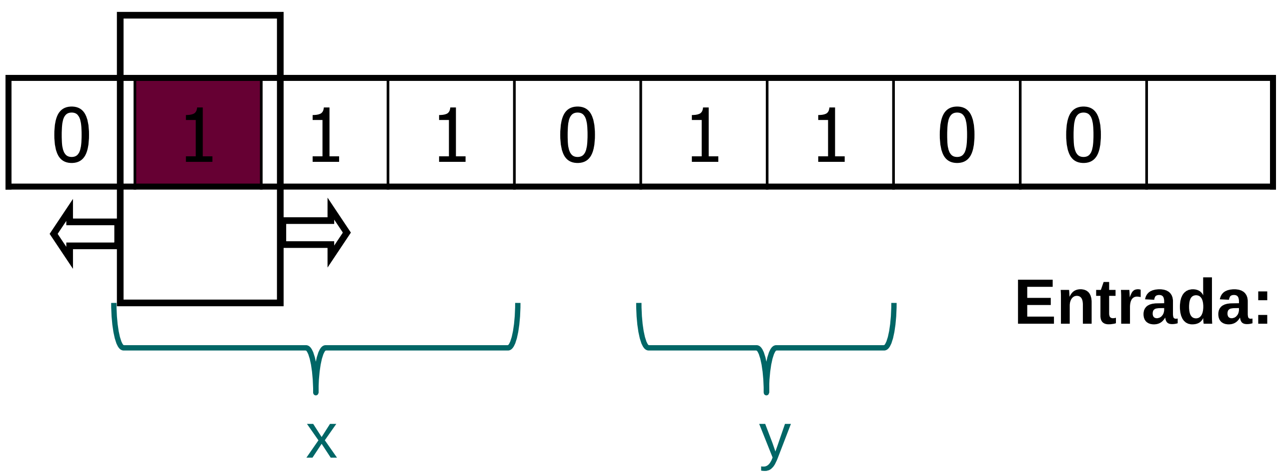
Entrada: (2,1)

Ejemplo: MT para la función X+Y

Codificación de entrada (Notación unaria)

- Dado $n \in N$, se representa por un bloque de (n+1) 1's consecutivos.
- Dado un vector $(n_1, \ldots, n_k)$, se codificará como

$$1 \, .. \, (n_1+1) \, .. \, 101 .. \, (n_2+1) \, .. \, 10 \ldots 01 .. \, (n_k+1) \, .. \, 1$$

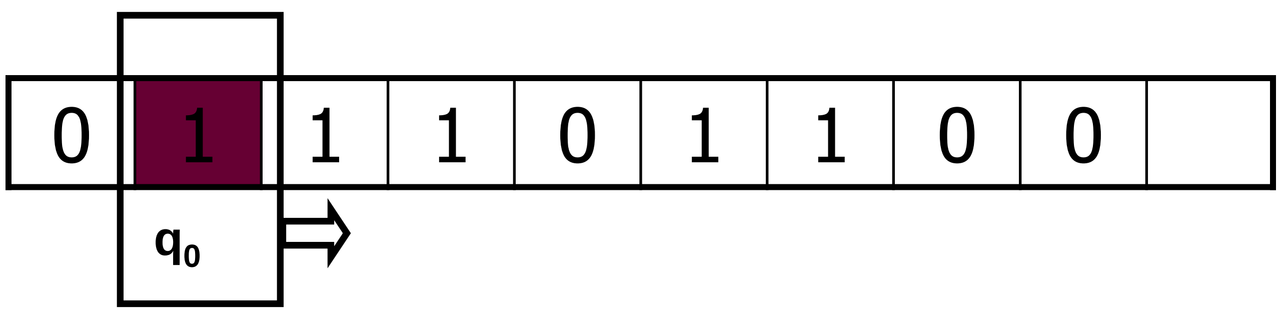


Entrada: (2,1)

Ejemplo: MT para la función X+Y

Función de Transiciones

T :Q x X → X x {I,D,N,H} x Q

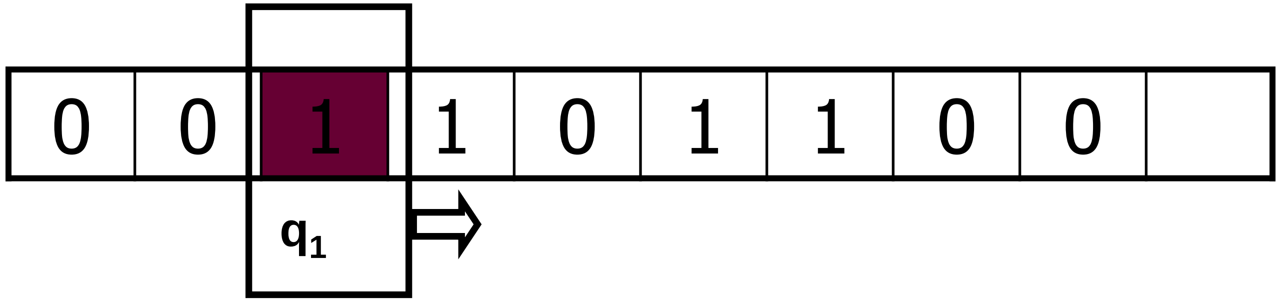


T(q_0, 1)=(0,D, q_1) **q_0 Estado Inicial**

Ejemplo: MT para la función X+Y

Funcionamiento de la Máquina de Turing

T :Q x X → X x {I,D,N,H} x Q

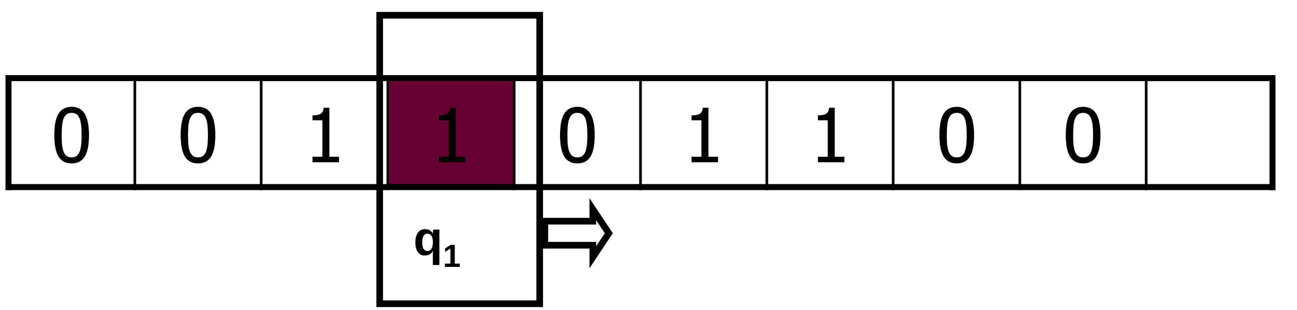


$T(q_1, 1)=(1, D, q_1)$

Ejemplo: MT para la función X+Y

Funcionamiento de la Máquina de Turing

T :Q x X → X x {I,D,N,H} x Q

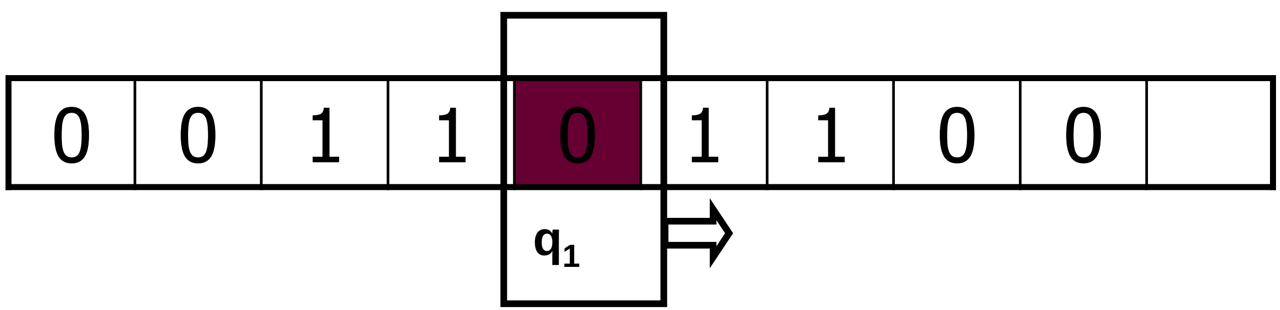


$T(q_1, 1)=(1, D, q_1)$

Ejemplo: MT para la función X+Y

Funcionamiento de la Máquina de Turing

$\mathbf{T} : Q \times X \rightarrow X \times \{I,D,N,H\} \times Q$

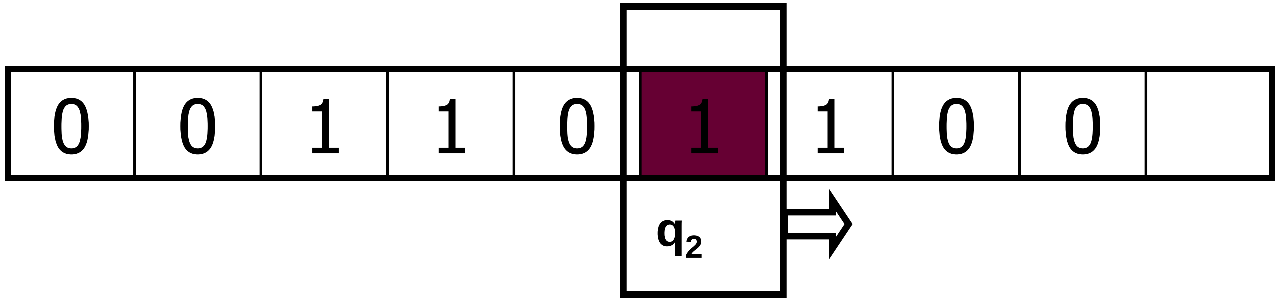


$T(q_1, 0)=(0, D, q_2)$

Ejemplo: MT para la función X+Y

Funcionamiento de la Máquina de Turing

T :Q x X → X x {I,D,N,H} x Q

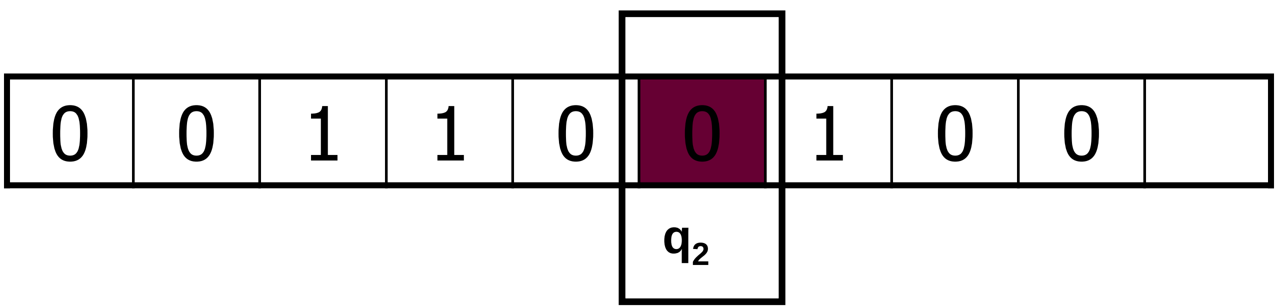


$T(q_2, 1)=(0, N, q_2)$

Ejemplo: MT para la función X+Y

Funcionamiento de la Máquina de Turing

T :Q x X → X x {I,D,N,H} x Q



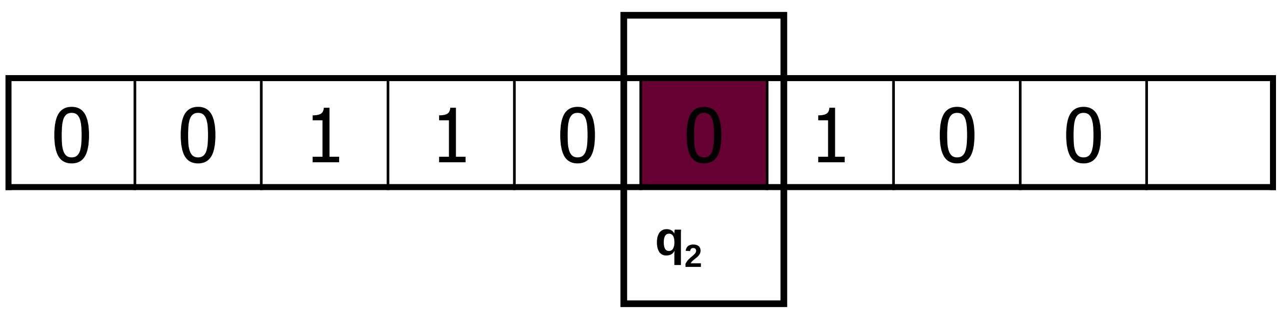
T(q_2, 0) no definido → la MT se para

q_2 Estado Final

Ejemplo: MT para la función X+Y

Valor de salifa (notación unaria)

Número total de 1’s (consecutivos o no) en la cinta



q_2 Estado Final

Salida: 3

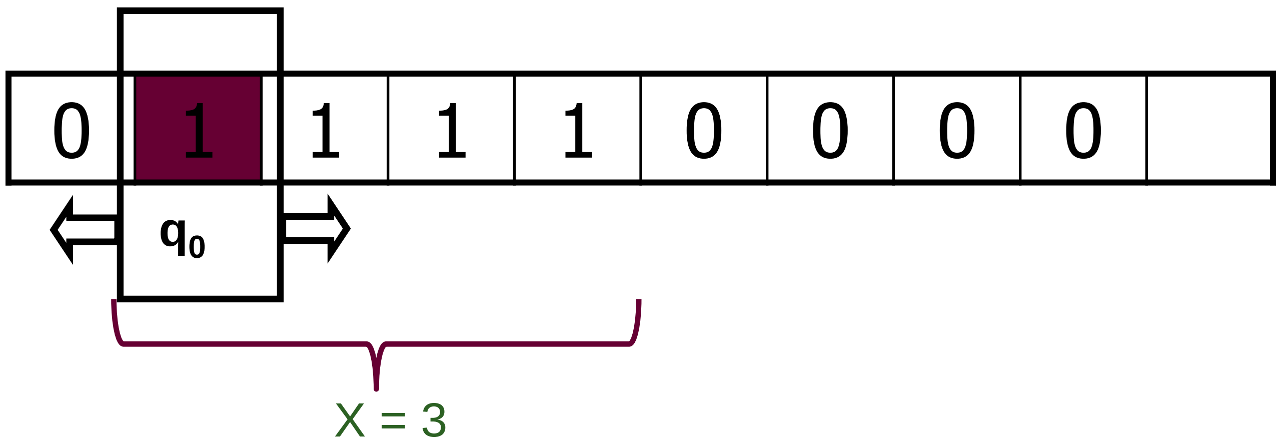
Ejemplo: MT para la función X+Y

Función de Transiciones

$T : Q \times X \rightarrow X \times \{I,D,N,H\} \times Q$

$T(q_0, 1)=(0, D, q_1)$	$(q_0, 1, 0, D, q_1)$
$T(q_1, 1)=(1, D, q_1)$	$(q_1, 1, 1, D, q_1)$
$T(q_1, 0)=(0, D, q_2)$	$(q_1, 0, 0, D, q_2)$
$T(q_2, 1)=(0, H, q_2)$	$(q_2, 1, 0, H, q_2)$

Ejemplo: MT para la función 2X

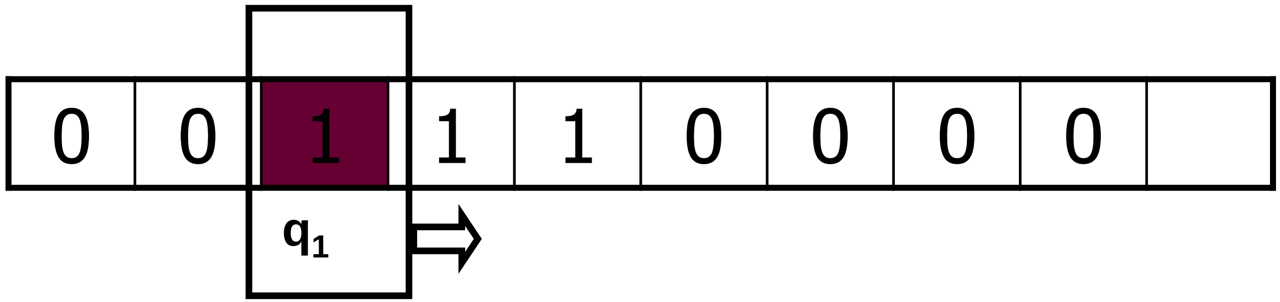


$T(q_0, 1)=(0,D, q_1)$ **# Quitar el 1 debido a la codificación**

Estado Inicial

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

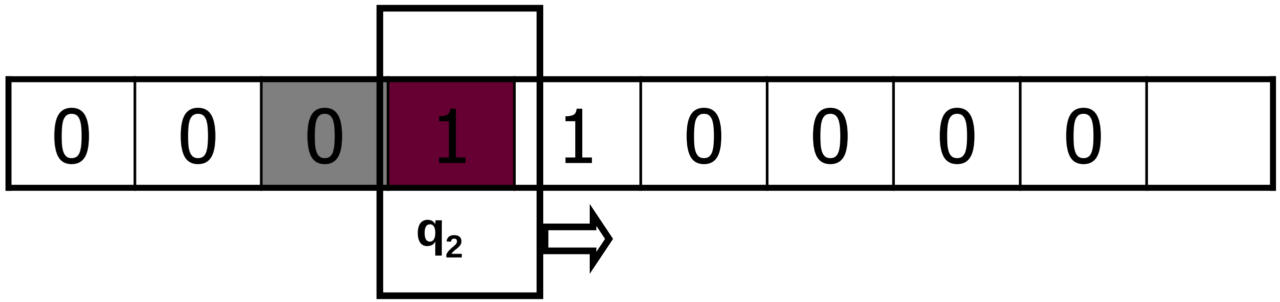
Ejemplo: MT para la función 2X



$T(q_1, 1)=(0,D, q_2)$ **# Marcar el 1 a duplicar**

$(q_0, 1 , 0,D, q_1)$
$(q_1, 1 , 0,D, q_2)$

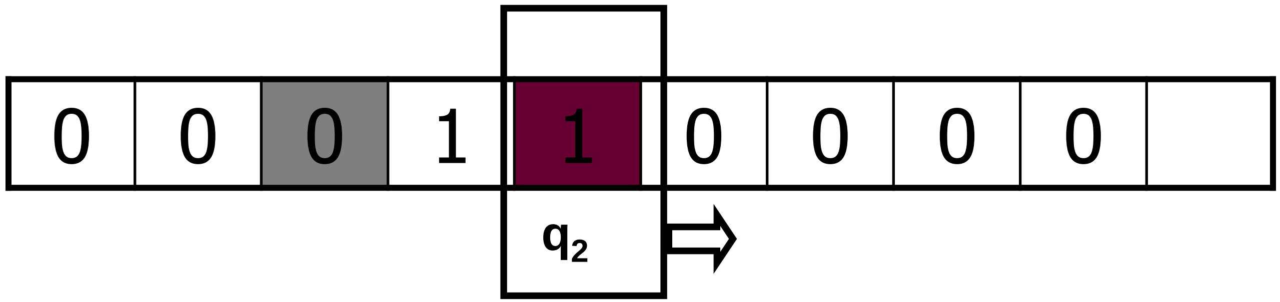
Ejemplo: MT para la función 2X



$T(q_2, 1)=(1,D, q_2)$ **# Interior de la codificación**

$(q_0, 1 , 0,D, q_1)$
$(q_1, 1 , 0,D, q_2)$
$(q_2, 1 , 1,D, q_2)$

Ejemplo: MT para la función 2X

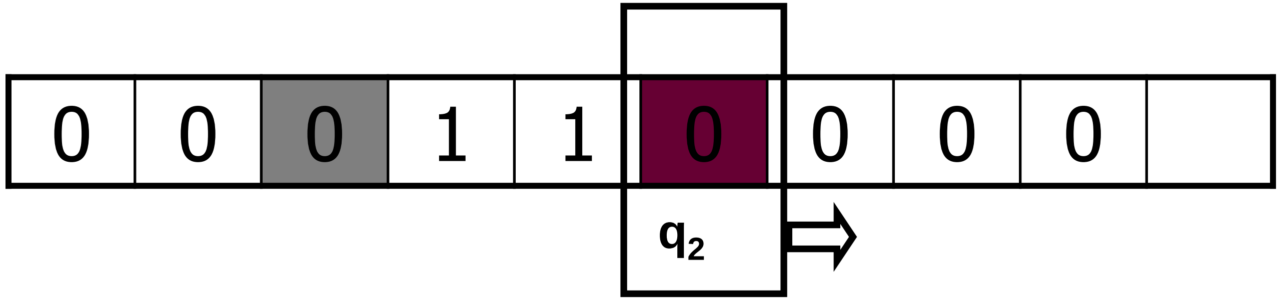


$T(q_2, 1)=(1, q_2, D)$

 # Interior de la codificación

$(q_0, 1, 0, D, q_1)$
$(q_1, 1, 0, D, q_2)$
$(q_2, 1, 1, D, q_2)$

Ejemplo: MT para la función 2X

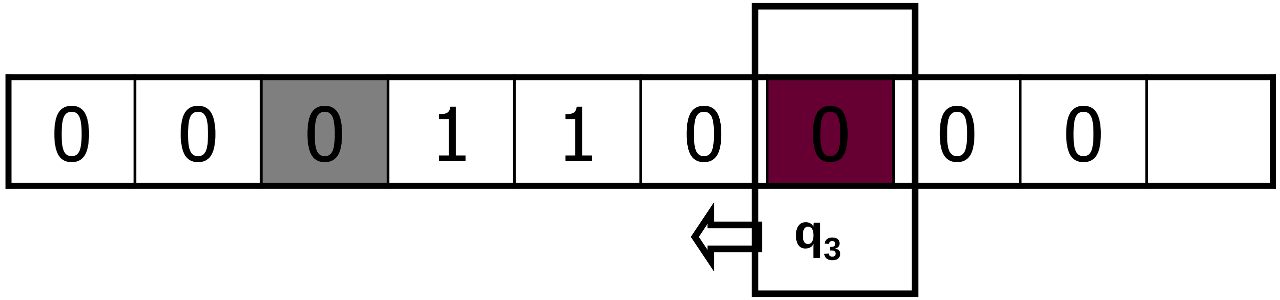


$T(q_2, 0)=(0,D, q_3)$

 # Fin de la codificación; nuevo estado

$(q_0, 1 , 0,D, q_1)$
$(q_1, 1 , 0,D, q_2)$
$(q_2, 1 , 1,D, q_2)$
$(q_2, 0 , 0,D, q_3)$

Ejemplo: MT para la función 2X

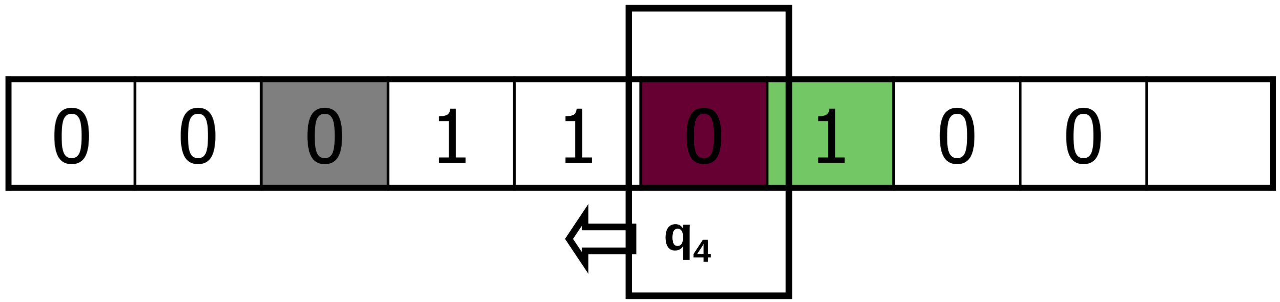


$T(q_3, 0)=(1,I, q_4)$

 # buscar una celda para escribir 1. Duplicar 1

$(q_0, 1 , 0,D, q_1)$
$(q_1, 1 , 0,D, q_2)$
$(q_2, 1 , 1,D, q_2)$
$(q_2, 0 , 0,D, q_3)$
$(q_3, 0 , 1,I, q_4)$

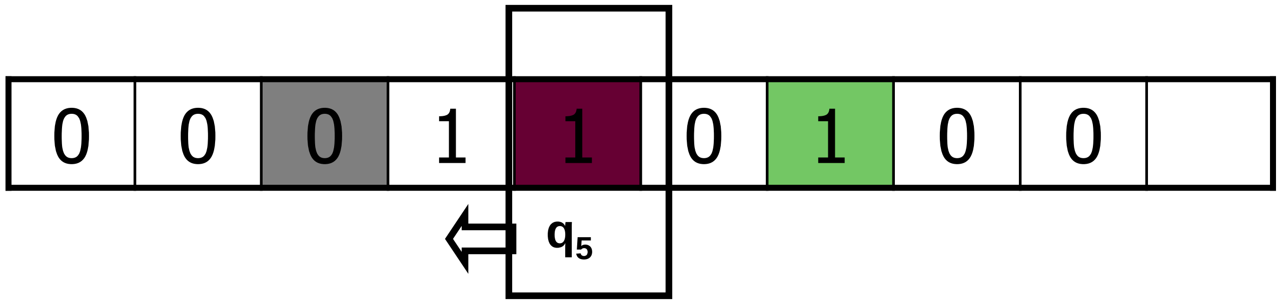
Ejemplo: MT para la función 2X



$T(q_4, 0)=(0,I, q_5)$ **# separador encontrado**

$(q_0, 1 , 0,D, q_1)$
$(q_1, 1 , 0,D, q_2)$
$(q_2, 1 , 1,D, q_2)$
$(q_2, 0 , 0,D, q_3)$
$(q_3, 0 , 1,I, q_4)$
$(q_4, 0 , 0,I, q_5)$

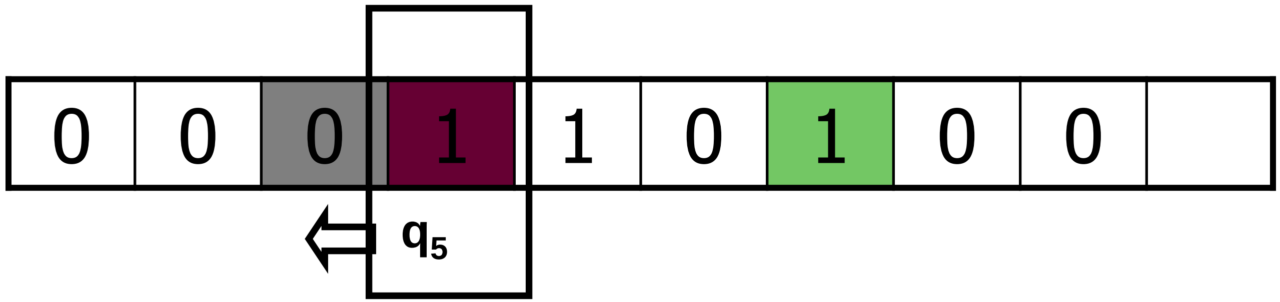
Ejemplo: MT para la función 2X



$T(q_5, 1)=(1,I, q_5)$ # **Interior de la codificación**

$(q_0, 1, 0,D, q_1)$ $(q_5, 1, 1,I, q_5)$
$(q_1, 1, 0,D, q_2)$
$(q_2, 1, 1,D, q_2)$
$(q_2, 0, 0,D, q_3)$
$(q_3, 0, 1,I, q_4)$
$(q_4, 0, 0,I, q_5)$

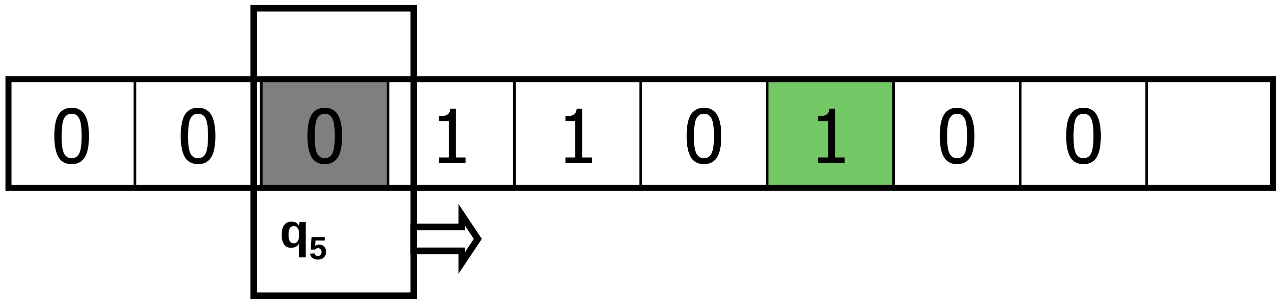
Ejemplo: MT para la función 2X



$T(q_5, 1)=(1,I, q_5)$ # **Interior de la codificación**

$(q_0, 1 , 0,D, q_1)$ $(q_5, 1 , 1,I, q_5)$
$(q_1, 1 , 0,D, q_2)$
$(q_2, 1 , 1,D, q_2)$
$(q_2, 0 , 0,D, q_3)$
$(q_3, 0 , 1,I, q_4)$
$(q_4, 0 , 0,I, q_5)$

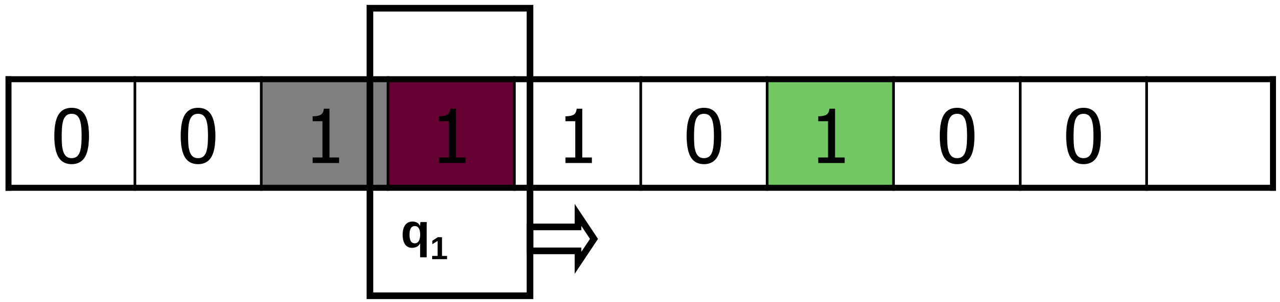
Ejemplo: MT para la función 2X



$T(q_5, 0)=(1,D, q_1)$ # marca encontrada

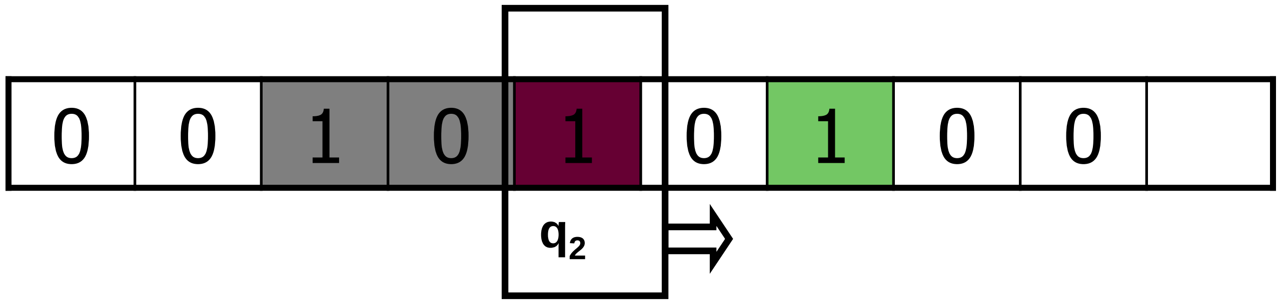
$(q_0, 1 , 0,D, q_1)$ $(q_5, 1 , 1,I, q_5)$
$(q_1, 1 , 0,D, q_2)$ $(q_5, 0 , 1,D, q_1)$
$(q_2, 1 , 1,D, q_2)$
$(q_2, 0 , 0,D, q_3)$
$(q_3, 0 , 1,I, q_4)$
$(q_4, 0 , 0,I, q_5)$

Ejemplo: MT para la función 2X



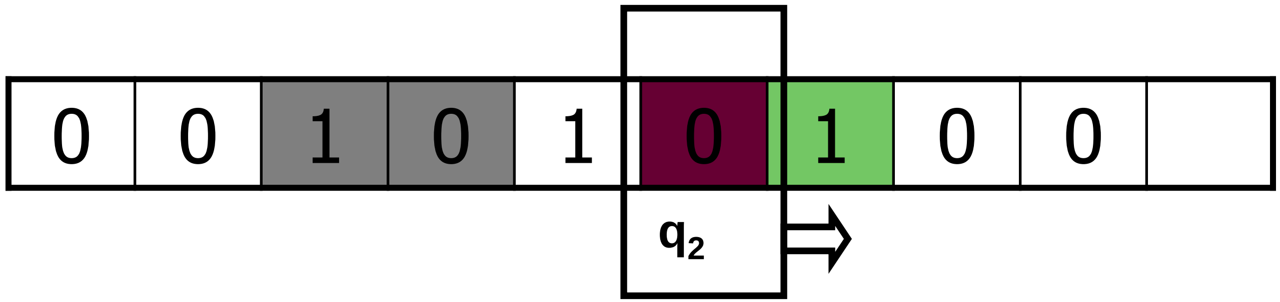
(q_0, 1 , 0,D, q_1)
(q_1, 1 , 0,D, q_2)
(q_2, 1 , 1,D, q_2)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_4, 0 , 0,I, q_5)
(q_5, 1 , 1,I, q_5)
(q_5, 0 , 1,D, q_1)

Ejemplo: MT para la función 2X



$(q_0, 1, 0, D, q_1)$
$(q_1, 1, 0, D, q_2)$
$(q_2, 1, 1, D, q_2)$
$(q_2, 0, 0, D, q_3)$
$(q_3, 0, 1, I, q_4)$
$(q_4, 0, 0, I, q_5)$
$(q_5, 1, 1, I, q_5)$
$(q_5, 0, 1, D, q_1)$

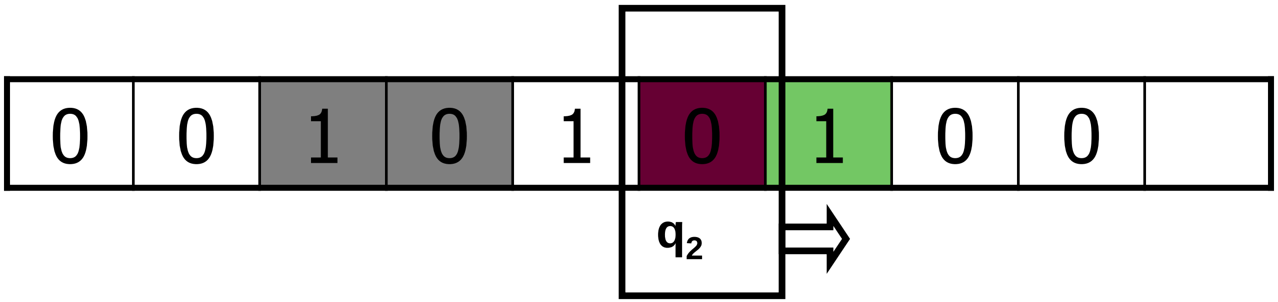
Ejemplo: MT para la función 2X



$(q_0, 1, 0, D, q_1)$
$(q_1, 1, 0, D, q_2)$
$(q_2, 1, 1, D, q_2)$
$(q_2, 0, 0, D, q_3)$
$(q_3, 0, 1, I, q_4)$
$(q_4, 0, 0, I, q_5)$

$(q_5, 1, 1, I, q_5)$
$(q_5, 0, 1, D, q_1)$

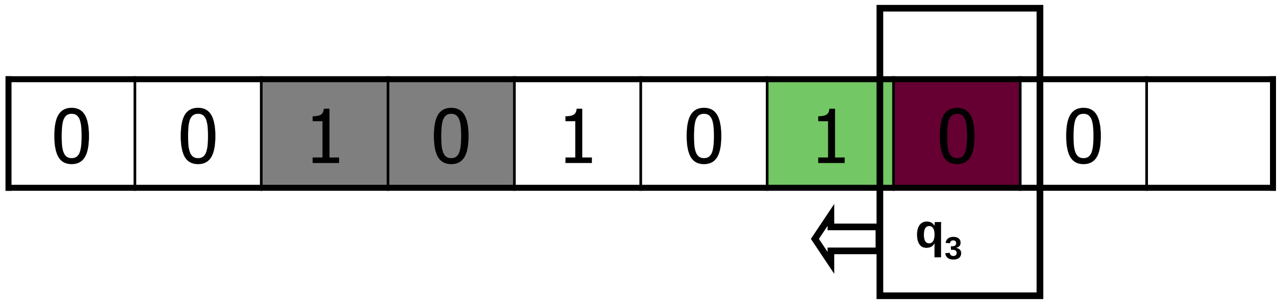
Ejemplo: MT para la función 2X



$(q_0, 1 , 0,D, q_1)$
$(q_1, 1 , 0,D, q_2)$
$(q_2, 1 , 1,D, q_2)$
$(q_2, 0 , 0,D, q_3)$
$(q_3, 0 , 1,I, q_4)$
$(q_4, 0 , 0,I, q_5)$

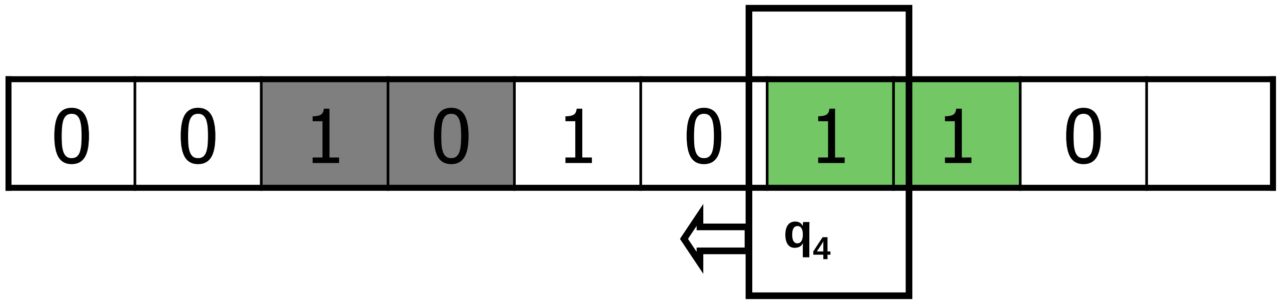
$(q_5, 1 , 1,I, q_5)$
$(q_5, 0 , 1,D, q_1)$

Ejemplo: MT para la función 2X



(q_0, 1 , 0,D, q_1)
(q_1, 1 , 0,D, q_2)
(q_2, 1 , 1,D, q_2)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)
(q_5, 1 , 1,I, q_5)
(q_5, 0 , 1,D, q_1)

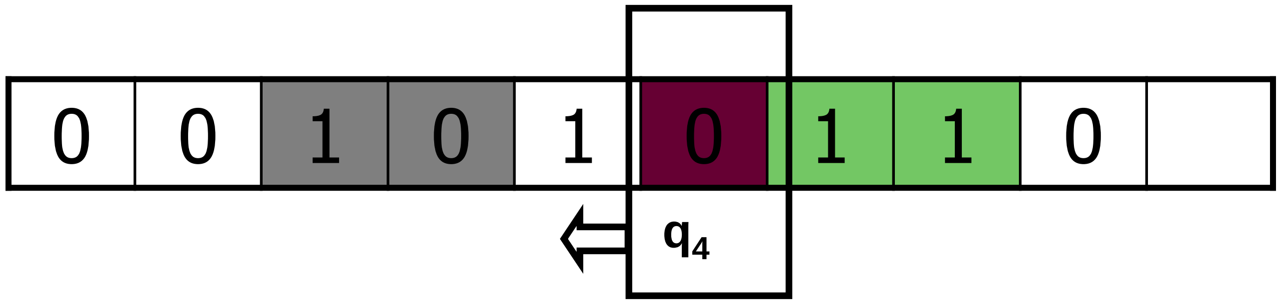
Ejemplo: MT para la función 2X



T(q_4, 1)=(1, q_4, I) **# buscando el separador**

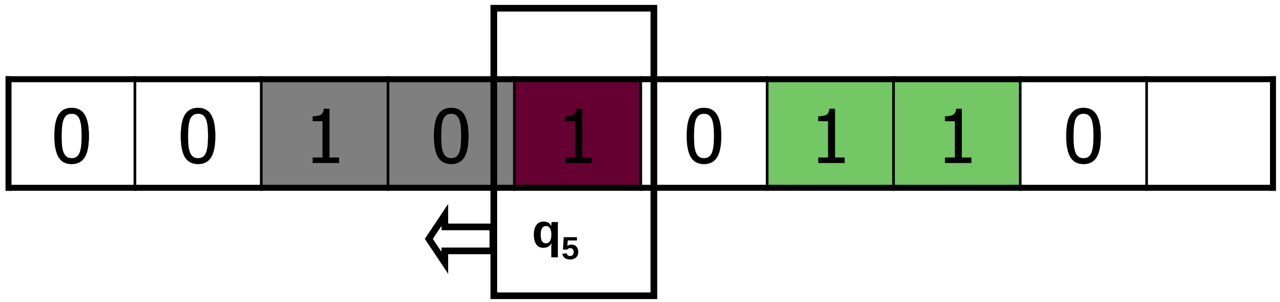
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



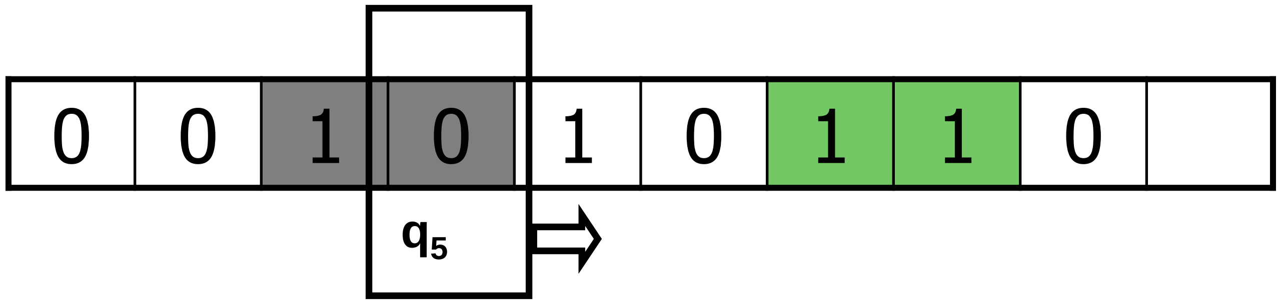
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



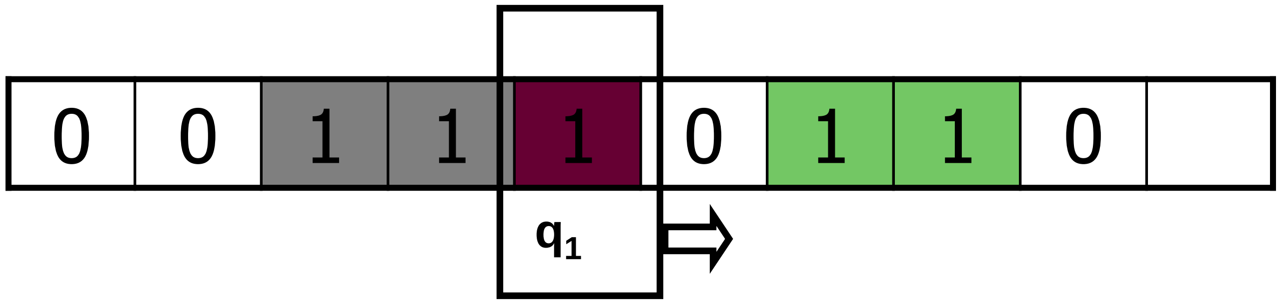
$(q_0, 1, 0, D, q_1)$ $(q_4, 1, 1, I, q_4)$
$(q_1, 1, 0, D, q_2)$ $(q_5, 1, 1, I, q_5)$
$(q_2, 1, 1, D, q_2)$ $(q_5, 0, 1, D, q_1)$
$(q_2, 0, 0, D, q_3)$
$(q_3, 0, 1, I, q_4)$
$(q_3, 1, 1, D, q_3)$
$(q_4, 0, 0, I, q_5)$

Ejemplo: MT para la función 2X



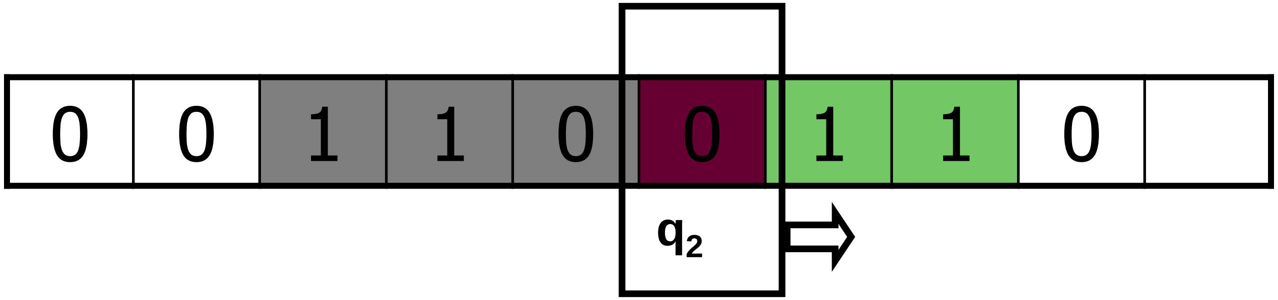
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



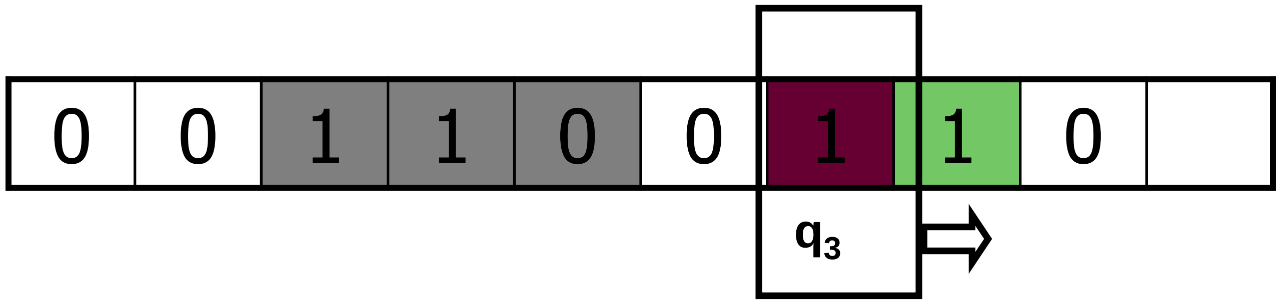
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



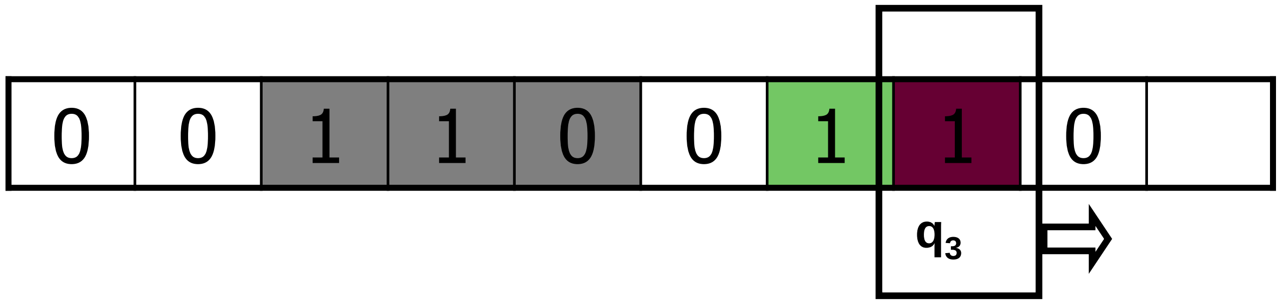
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



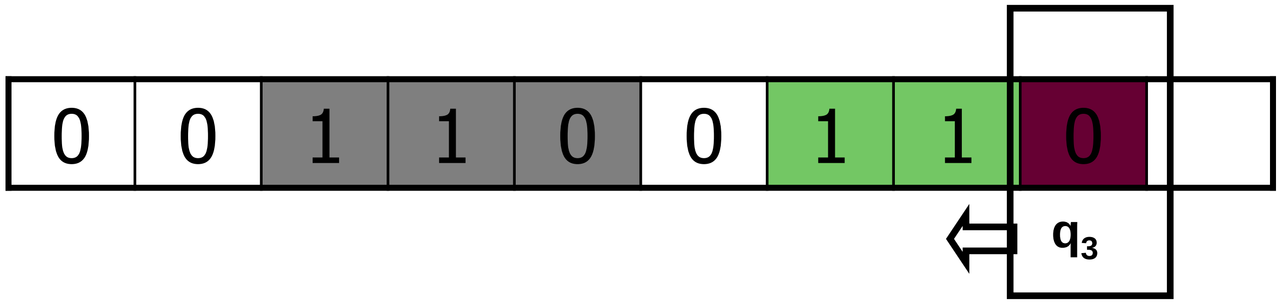
$(q_0, 1, 0, D, q_1)$ $(q_4, 1, 1, I, q_4)$
$(q_1, 1, 0, D, q_2)$ $(q_5, 1, 1, I, q_5)$
$(q_2, 1, 1, D, q_2)$ $(q_5, 0, 1, D, q_1)$
$(q_2, 0, 0, D, q_3)$
$(q_3, 0, 1, I, q_4)$
$(q_3, 1, 1, D, q_3)$
$(q_4, 0, 0, I, q_5)$

Ejemplo: MT para la función 2X



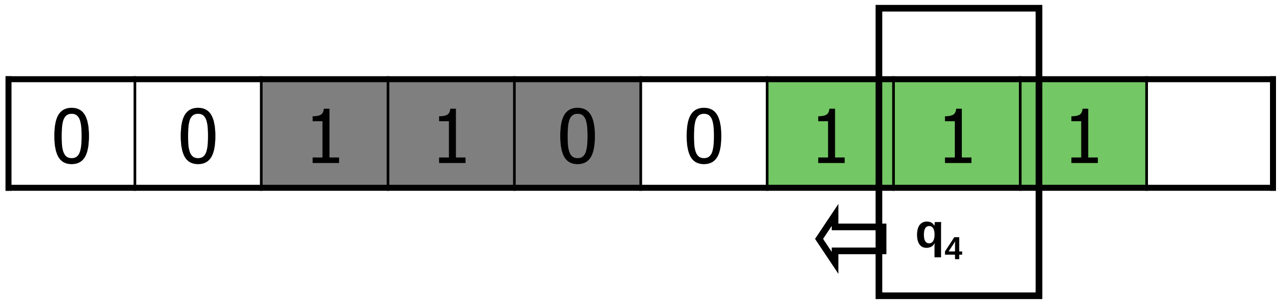
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



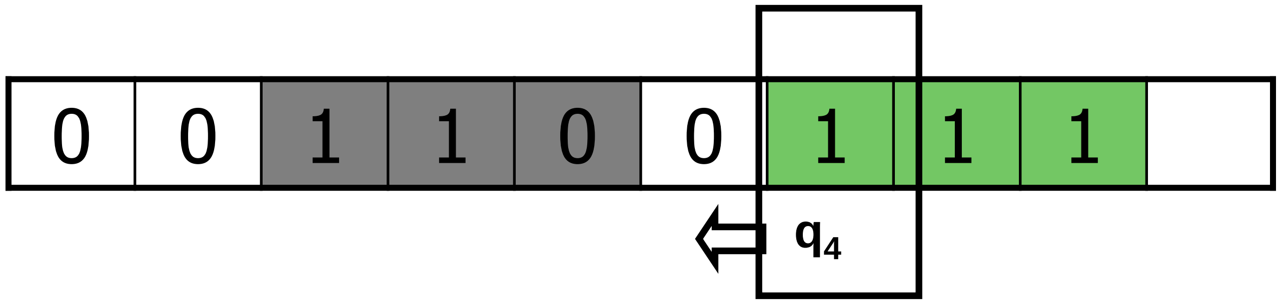
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



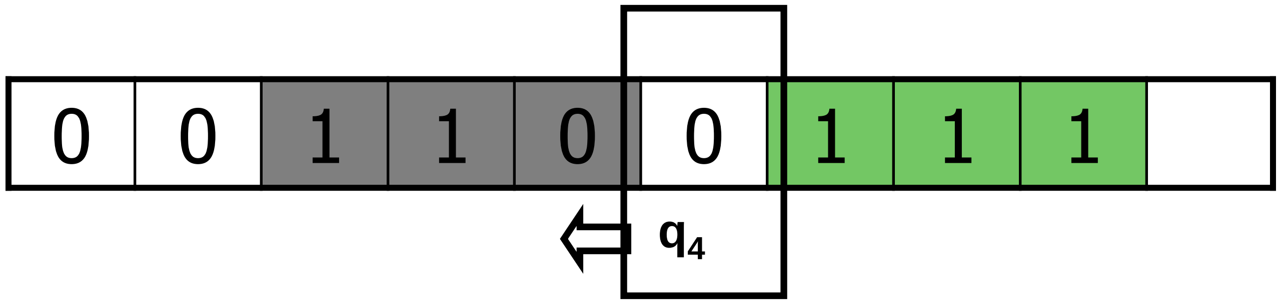
(q_0, 1 , 0,D, q_1) **(q_4, 1 , 1,I, q_4)**
(q_1, 1 , 0,D, q_2) **(q_5, 1 , 1,I, q_5)**
(q_2, 1 , 1,D, q_2) **(q_5, 0 , 1,D, q_1)**
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



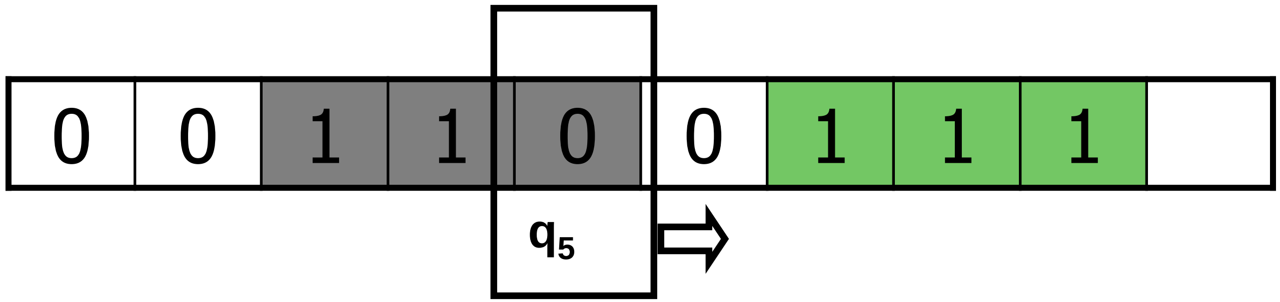
(q_0, 1 , 0,D, q_1) **(q_4, 1 , 1,I, q_4)**
(q_1, 1 , 0,D, q_2) **(q_5, 1 , 1,I, q_5)**
(q_2, 1 , 1,D, q_2) **(q_5, 0 , 1,D, q_1)**
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



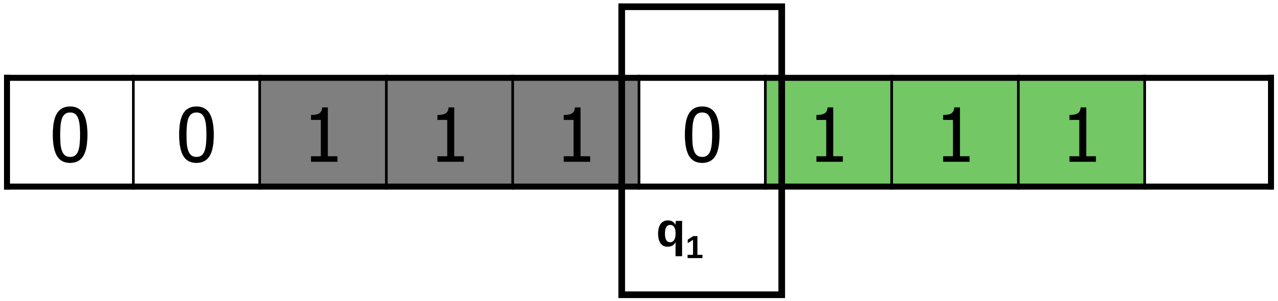
(q_0, 1 , 0,D, q_1) (q_4, 1 , 1,I, q_4)
(q_1, 1 , 0,D, q_2) (q_5, 1 , 1,I, q_5)
(q_2, 1 , 1,D, q_2) (q_5, 0 , 1,D, q_1)
(q_2, 0 , 0,D, q_3)
(q_3, 0 , 1,I, q_4)
(q_3, 1 , 1,D, q_3)
(q_4, 0 , 0,I, q_5)

Ejemplo: MT para la función 2X



$(q_0, 1, 0, D, q_1)$ $(q_4, 1, 1, I, q_4)$
$(q_1, 1, 0, D, q_2)$ $(q_5, 1, 1, I, q_5)$
$(q_2, 1, 1, D, q_2)$ $(q_5, 0, 1, D, q_1)$
$(q_2, 0, 0, D, q_3)$
$(q_3, 0, 1, I, q_4)$
$(q_3, 1, 1, D, q_3)$
$(q_4, 0, 0, I, q_5)$

Ejemplo: MT para la función 2X



$(q_0, 1, 0, D, q_1)$
$(q_1, 1, 0, D, q_2)$
$(q_1, 0, 0, H, f)$
$(q_2, 1, 1, D, q_2)$
$(q_2, 0, 0, D, q_3)$
$(q_3, 0, 1, I, q_4)$
$(q_3, 1, 1, D, q_3)$
$(q_4, 0, 0, I, q_5)$
$(q_4, 1, 1, I, q_4)$
$(q_5, 1, 1, I, q_5)$
$(q_5, 0, 1, D, q_1)$

Composición MT: Ejercicio I

Tenemos una máquina de Turing M1 que calcula una función f(X, Y). Queremos construir otra máquina de Turing M cuya función semántica unaria asociada sea g(X) = f(0, X). Si los estados de M1 son {q0, q1, q2, q3}, q0 es su estado inicial y q3 su estado final, añade las transiciones que sean necesarias para obtener M a partir de M1.

Quíntupla que define a una MT(X, Q, T, i, F)

M1 computa ⇒ f(X,Y) ⇒ M1: ({0,1},T1, {q0, q1, q2, q3}, q0, q3)

M computa ⇒ g(X) ⇒ M: ({0,1},T1 **∪ T'**, {q0, q1, q2, q3} **∪ Q′** , **p0**, q3)

Para computar la salida de M, necesitamos ejecutar **M1** con entrada **(0,X)**

Para ello, hemos de convertir la entrada de **M** en una entrada para **M1** que en el primer argumento reciba el valor 0 y en el segundo X.

¡OJO! Se ha de tener cuidado para no definir transiciones contradictorias para el mismo estado. Por eso los nuevos estados de M comienzan en **p0**.

Composición MT: Ejercicio I

Tenemos una máquina de Turing M1 que calcula una función f(X, Y). Queremos construir otra máquina de Turing M cuya función semántica unaria asociada sea g(X) = f(0, X). Si los estados de M1 son {q0, q1, q2, q3}, q0 es su estado inicial y q3 su estado final, añade las transiciones que sean necesarias para obtener M a partir de M1.

Para computar la salida de M, necesitamos invocar a **M1** con entrada **(0,X)**

Entrada: X=3

M computa → g(X) → 01…(X+1) …10 → 011110

Definir las transiciones que convierten la entrada de M en la entrada (0,X) para M1

Entrada: X=0,Y=3

M1 entrada → f(X,Y) → 01…(X+1) …101…(Y+1) …10 → 01011110

Composición MT: Ejercicio I

Tenemos una máquina de Turing M1 que calcula una función f(X, Y). Queremos construir otra máquina de Turing M cuya función semántica unaria asociada sea g(X) = f(0, X). Si los estados de M1 son {q0, q1, q2, q3}, q0 es su estado inicial y q3 su estado final, añade las transiciones que sean necesarias para obtener M a partir de M1.

Para computar la salida de M, necesitamos invocar a **M1** con entrada **(0,X)**

Entrada: X=3

M computa → g(X) → 011110

Traza para X=3

0 p_0 11110

p_1 011110

p_2 0011110

q_0 1011110

Entrada: X=0,Y=3

M1 entrada → f(X,Y) → 01011110

Definimos las transiciones que convierten la entrada de M en la entrada (0,X) para M1

(p_0, 1 , 1, I, p_1)

(p_1, 0 , 0, I, p_2)

(p_2, 0 , 1, N, q_0)

Composición MT: Ejercicio II

Sea g(X,Y) una función computada por una Máquina de Turing M1, con los siguientes estados {q0, q1, q2, q3}; siendo q0 el estado inicial y q3 único estado final de M1 . Constrúyase una Máquina de Turing M cuya función semántica binaria sea:

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

Quíntupla que define a una MT(X, Q, T, i, F)

M1 computa	→	g(X,Y)	→	M1: ({0,1},T1, {q0, q1, q2, q3}, q0, q3)
M computa	→	f(X,Y)	→	M: ({0,1},T1 **∪ T'**, {q0, q1, q2, q3} ∪ **Q'** , **p0**, q3 ∪ F')

Salida de M

Para ver que salida computa M, lo primero es comprobar si y==0 ó y>0.

- Si **y==0** hemos de borrar el 1 de **Y**, dejando a la salida en la cinta **X+1.**
- Si **y>0** hemos de invocar a **M1** con la entrada **(X,Y)** dejándola en su estado inicial (q0) y sobre el primer 1 de la X. Cuando termine **M1** dejará en la cinta g(X,Y) **.**

¡OJO! Se ha de tener cuidado para no definir transiciones contradictorias para el mismo estado. Por eso los estados de M comienzan en **p0**.

Composición MT: Ejercicio II

Sea g(X,Y) una función computada por una Máquina de Turing M1, con los siguientes estados {q0, q1, q2, q3}; siendo q0 el estado inicial y q3 único estado final de M1 . Constrúyase una Máquina de Turing M cuya función semántica binaria sea:

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

Comprobamos si y es 0 o mayor. Para ello miramos si tiene sólo el 1 de la codificación (en cuyo caso será igual a 0) o más de un 1 (en cuyo caso será >0). Transiciones que se añaden a las de M1:

(p_0, 1 , 1, D, p_0) // Voy al separador

(p_0, 0 , 0, D, p_1) // Lo salto

(p_1, 1 , 1, D, p_2) // Salto el 1 de la codificación entrada de Y

(p_2, 0 , 0, I, p_3) // Si no hay un 2º uno es que Y==0.

(p_3, 1 , 0, H, p_f) // Borro el 1 de la codificación de entrada de Y, y termino (en la cinta queda X+1,

// pues no borramos el 1 de la codificación de entrada de X).

Compruebo que y==0, dejo en la cinta X+1 y termino (paro en un estado final p_f de M)

(p_2, 1 , 1, I, p_4) // Vuelvo al 1 de la codificación de Y

(p_4, 1 , 1, I, p_4) // Lo salto

(p_4, 0 , 0, I, p_5) // Salto el separador

(p_5, 1 , 1, I, p_5) // Vuelvo al principio de X

(p_5, 0 , 0, D, q_0) // Coloco q_0 en el primer uno de X para enlazar con M1

Si y>0 (hay un 2º uno, es decir cuando estoy en p_2 leo un 1), dejo todo como está, y voy a dejar al principio de la X el estado q_0 para enlazar con M1.

Composición MT: Ejercicio III

Dada una Máquina de Turing Mf cuyo estado inicial es q0, constrúyase una Máquina de Turing Mg cuya función binaria semántica sea g(X,Y) = f(2X,Y)

Quíntupla que define a una MT(X, Q, T, i, F)

Mf computa → f(X,Y) → Mf: ({0,1},T_1, Q_f, q0, F_f)

Mg computa → g(X,Y) → M: ({0,1}, T1 ∪ T', Q_f ∪ **Q'** , **p0**, F_f ∪ **F'**)

Para computar la salida de Mg, necesitamos invocar a **Mf** con entrada **(2X,Y)**

Para ello, hemos de convertir la entrada de **Mg** en una entrada para **Mf** que en el primer argumento reciba el valor 2X y en el segundo Y.

¡OJO! Se ha de tener cuidado para no definir transiciones contradictorias para el mismo estado. Por eso los estados de Mg comienzan en p_0.

Entrada Mg **Entrada Mf**

X=2 Y=3
111 **0** 1111
|
p_0 → Transiciones que conviertan la entrada (2,3) para Mg en la entrada (4,3) para Mf → X=4 Y=3
11111 **0** 1111
|
q_0

Composición MT: Ejercicio III

Dada una Máquina de Turing Mf cuyo estado inicial es q0, constrúyase una Máquina de Turing Mg cuya función binaria semántica sea g(X,Y) = f(2X,Y)

Saltamos el 1 de la codificación de entrada de X y copiamos el resto de 1's de X a su izquierda

(p_0, 1 , 1, D, p_1) // Saltamos el 1 de la codificación de entrada de X

(p_1, 1 , 0, I, p_2) // Marco un 1 para copiar

(p_2, 1 , 1, I, p_2) // Recorro los 1's para ir a copiar al primer 0 que encuentre

(p_2, 0 , 1, D, p_3) // Copio el 1

(p_3, 1 , 1, D, p_3) // Vuelvo a la marca

(p_3, 0 , 1, D, p_1) // Desmarco y repito el proceso con el resto de 1's de X

Cuando hemos copiado todos los 1's de X, hemos duplicado X (tenemos 2X), estamos con p_1 en el separador y no hay más unos que copiar de X. Volvemos al inicio de 2X a dejar el estado inicial de Mf sobre el primer 1.

(p_1, 0 , 0, I, p_4) // Salto el separador

(p_4, 1 , 1, I, p_4) // Vuelvo al extremo izquierdo de 2X

(p_4, 0 , 0, D, q_0) // Dejo sobre el primer 1 q_0, para enlazar con Mf

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 Programas While - MT

- **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
- Mantendremos siempre bloques de 1's separados por un solo 0
- 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
- X_i=succ(X_k):
  - Sustituir el bloque i-ésimo por el sucesor del k-ésimo
  - Desplazar para crear espacio o para reducir separación
- X_i=pred(X_k):
  - Similar al anterior

Equivalencia Programas While - MT

- ## Teorema:

Toda función While-computable es Turing-computable.

- ## Demostración:

Simular la ejecución de un PW mediante una MT:

  - begin i_1 $i_{2...}$ i_n end:
    - Componer MT's para i_1 $i_{2...}$ i_n
    - Cambiar nombres de estados
    - Cambiar estados finales/iniciales
  - while $x_n \neq x_m$ do i:
    - Comparar x_n y x_m
    - Si son distintos:
      - Concatenar con una MT para i
      - Repetir
  - Finalmente, borrar 1's sobrantes

Más 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 !

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, que hoy nos puede parecer evidente, se formuló como la tesis de Church:

“Todo algoritmo o procedimiento efectivo es computable”

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.

Implicaciones de la 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 son simulables mediante MT?
  - ¿Los podemos utilizar para hacer cálculos?