Tema 2: Otros Modelos de Cálculo: Tesis de Church-Turing

Serafín Moral

Universidad de Granada

Marzo, 2020

Contenido

El objetivo de este tema es estudiar otros modelos de cálculo sencillos y demostrar que los cálculos son equivalentes a los realizados por una MT. Esto apoyará la conocida como tesis de Church-Turing que indica que todos los modelos de cálculo pueden llevarse a cabo por una MT.

- Programas de Post Turing
- Programas con variables
- Cálculo con números enteros
- Tesis de Church Turing

Programas de Post-Turing

- En este apartado vamos a estudiar otro lenguaje de programación para manipulación de palabras, el sistema de Post-Turing. En este modelo se podrán escribir programas que actúan sobre una cinta en la que se pueden escribir o leer símbolos, ilimitada en ambas direcciones.
- En los cálculos asociados a este modelo siempre hay una casilla de la cinta que está activa en un momento dado. Este símbolo y solo este de la cinta se supone observado y en esta casilla solo se puede escribir un símbolo.
- En un paso de cálculo el símbolo de la casilla se puede leer, ejecutar una determinada instrucción en función de dicho símbolo, escribir un nuevo símbolo y entonces moverse a la casilla de la izquierda o la derecha.
- La diferencia con las MT es que ahora la estructura para especificar el control es más parecida a un programa en un lenguaje de programación convencional: una lista de instrucciones.

Estructura de los programas

- Un programa es un conjunto de instrucciones sobre un alfabeto de entrada A y un alfabeto de trabajo B que incluye el símbolo blanco.
- Cada instrucción puede tener una etiqueta opcional. Una etiqueta es una palabra en un alfabeto determinado. Las etiquetas las escribimos como [L] al principio de la instrucción.
- Hay cuatro tipo de instrucciones:
  - PRINT a, *Imprime el símbolo a en la casilla de la cinta donde está posicionado el cabezal.*
  - IF a GOTO L, *si el símbolo que se ve en la cinta es el a, entonces sigue por la instrucción cuya etiqueta es L.*
  - RIGHT, *mueve el cabezal de lectura a la derecha.*
  - LEFT, *mueve el cabezal de lectura a la izquierda.*
  - HALT, *Termina y acepta.*
- El programa comienza con una entrada u del alfabeto de entrada en una cinta ilimitada y rodeada por blancos y con el cabezal de lectura a la izquierda de la palabra.
- Empieza a ejecutar la primera instrucción. Cada vez que termina una instrucción ejecuta la siguiente, excepto si hace un salto con IF a GOTO L.
- Termina si tiene que ejecutar una instrucción que no existe o llega a HALT.

Calculabilidad Post Turing

Lenguaje Aceptado

El lenguaje aceptado por un programa Post Turing es el conjunto de palabras del alfabeto de entrada que comenzando en la configuración inicial llegan a la instrucción HALT.

Función Calculada

La función calculada por un programa Post Turing es la función $f : D \to B^*$ definida en el conjunto D de las entradas $u \in A^*$ tal que el programa Post Turing llega a HALT siendo $f(u)$ la palabra en la cinta excluyendo blancos $u \in D$.
Una función calculada por un programa Post Turing se dice parcialmente calculable Post Turing.
Si $D = A^*$ se dice que es calculable total Post Turing.

Ejemplo

```
LEFT
PRINT 0
LEFT
PRINT 1
HALT
```

El programa añade 10 al principio de la palabra de entrada.

Ejemplo

Alfabeto de entrada $A = \{a, b, c\}$. El de trabajo tiene además el símbolo #.

```
      LEFT
[A]   RIGHT
      IF a GOTO A
      IF b GOTO A
      IF c GOTO A
      PRINT a
      RIGHT
      PRINT b
[C]   LEFT
      IF a GOTO C
      IF b GOTO C
      IF c GOTO C
      RIGHT
      HALT
```

Escribimos *ab* al final de la palabra y volvemos al principio de la palabra.

Ejemplo

Alfabeto de entrada $A = \{0,1\}$.

```
      LEFT
[C]   RIGHT
      IF # GOTO E
      IF 0 GOTO A
      IF 1 GOTO C
[A]   PRINT #
      IF # GOTO C
[E]   HALT
```

Sustituye todos los 0s por blancos.

Ejemplo

Alfabeto de entrada $A = \{a\}$. Alfabeto de trabajo $B = \{a, \#, c\}$.

```
[A]     IF # GOTO E
        PRINT c
[B]     RIGHT
        IF a GOTO B
[C]     RIGHT
        IF a GOTO C
        PRINT a
[D]     LEFT
        IF a GOTO D
        IF # GOTO D
        PRINT a
        RIGHT
        IF a GOTO A
[E]     HALT
```

Si empieza con a^n, acaba con $a^n \# a^n$.

Macros

Macros

Una macro es un conjunto de instrucciones que realizan una tarea concreta y a las que se la un nombre.
No suponen una funcionalidad adicional del lenguaje Post-Turing.
Simplemente es una forma de resumir conjuntos de instrucciones para simplificar la presentación y escritura de programas.

Ejemplo

La macro RIGHT TO NEXT BLANK se expande como,

```
[A]    RIGHT
       IF # GOTO E
       GOTO A
```

Una vez definida esta macro, ya podríamos usar RIGHT TO NEXT BLANK como un resumen de su expansión.

Problemas

1. Construir un programa Post-Turing que calcule la función $f(u) = u^{-1}$ donde $u \in \{0,1\}^*$.
2. Construir un programa Post-Turing que dado un número u en binario calcule $u+1$.
3. Construir un programa Post-Turing que dadas dos cadenas ucv donde $u, v \in \{0,1\}^*$ calcule si la cadena u es una subcadena de la cadena v.

Programas con Variables

Un programa con variables, viene dado por un alfabeto A de entrada, un alfabeto B de trabajo y los siguientes elementos:

- Una variable X de entrada, un conjunto finito de variables de trabajo $Z_1, \ldots, Z_l$ y una variable Y de salida.
- Un conjunto de instrucciones, opcionalmente etiquetadas, de la siguiente forma:
  - A $\leftarrow$ aA, *añadir el símbolo a al principio de la variable A.*
  - A $\leftarrow$ A-, *Eliminar el último símbolo de A (si no es vacía).*
  - IF A ENDS a GOTO L, *Si el último símbolo de A es una a seguir por la instrucción con etiqueta L.*
  - HALT, *termina y acepta.*

Se supone que empieza con $X = u$ donde $u \in A^*$ es el valor de la entrada y con el resto de las variables conteniendo la palabra vacía ε. Acepta una palabra si llega a HALT.
Calcula una función parcial f si llega a HALT con $f(u)$ almacenado en Y definida en el conjunto de palabras para las que el programa termina.

Ejemplo

Programa que tiene como entrada $X = u$ y calcula $Y = u$ sobre el alfabeto $\{0,1\}$

```
        IF X ENDS 0 GOTO A
        IF X ENDS 1 GOTO B
        HALT
[A]     X ← X-
        Y ← 0Y
        IF X ENDS 0 GOTO A
        IF X ENDS 1 GOTO B
        HALT
[B]     X ← X-
        Y ← 1Y
        IF X ENDS 0 GOTO A
        IF X ENDS 1 GOTO B
        HALT
```

Macros

Un conjunto de instrucciones que se usan a menudo se pueden resumir en una nueva instrucción llamada macro. Esto no cambia la definición del lenguaje, solo es una forma reducida de escribir programas.

- La macro IF V $\neq \varepsilon$ GOTO L con expansión:

  IF V ENDS a_1 GOTO L
  IF V ENDS a_2 GOTO L
  $\vdots$
  IF V ENDS a_n GOTO L

  donde $\{a_1, \ldots, a_n\}$ es el alfabeto de trabajo.
- La macro V $\leftarrow \varepsilon$ tiene la expansión:

  [L] V $\leftarrow$ V-
  IF V $\neq \varepsilon$ GOTO L

- La macro GOTO L tiene la expansión:

  Z $\leftarrow \varepsilon$
  Z $\leftarrow$ aZ
  IF Z ENDS a GOTO L

Resumiendo Instrucciones

Algunos conjuntos de instrucciones se pueden resumir. Por ejemplo,

IF V ENDS a_1 GOTO L_1
IF V ENDS a_2 GOTO L_2
$\vdots$
IF V ENDS a_n GOTO L_n

las resumiremos como
IF V ENDS a_i GOTO L_i (i=1,. . . ,n)

Macros

La macro $U \leftarrow V$ se expande como:

$Z \leftarrow \varepsilon$
$U \leftarrow \varepsilon$
[A] IF V ENDS a_i GOTO B_i $(i=1,\ldots,n)$
GOTO C
$[B_i]$ $V \leftarrow V\text{-}$, $U \leftarrow a_i U$, $Z \leftarrow a_i Z$, GOTO A $\}$ $i=1,\ldots,n$
[C] IF Z ENDS a_i GOTO D_i $(i=1,\ldots,n)$
GOTO E
$[D_i]$ $Z \leftarrow Z\text{-}$, $V \leftarrow a_i V$, GOTO C $\}$ $i=1,\ldots,n$

E Es la etiqueta de la instrucción después de la macro.

Ejemplo: Sumar 1

Vamos a suponer que $A = \{a_1, \ldots, a_n\}$ y que vamos a calcular la función $f(n) = n+1$, donde se supone que el número n se codifica como $C(n)$ en dicho alfabeto.

[B] IF X ENDS a_i GOTO A_i ($i=1,\ldots,n$)
Y $\leftarrow a_1$Y
HALT

[A_i] X $\leftarrow$ X-
Y $\leftarrow a_{i+1}$Y
GOTO C
} $i=1,\ldots,n-1$

[A_n] X $\leftarrow$ X-
Y $\leftarrow a_1$Y
GOTO B

[C] IF X ENDS a_i GOTO D_i ($i=1,\ldots,n$)
HALT

[D_i] X $\leftarrow$ X-
Y $\leftarrow a_i$Y
GOTO C
} $i=1,\ldots,n$

Ejemplo: Restar 1

Vamos a suponer que $A = \{a_1, \ldots, a_n\}$ y que vamos a calcular la función $f(n) = n \dot{-} 1$ ($n-1$ si $n > 1$ y 0 si $n = 1$), donde se supone que el número n se codifica como $C(n)$ en dicho alfabeto.

[B] IF X ENDS a_i GOTO A_i (i=1,...,n)
HALT

$[A_i]$ $X \leftarrow X$-
$Y \leftarrow a_{i-1}Y$
GOTO C
} i=2,...,n

$[A_1]$ $X \leftarrow X$-
IF $X \neq \varepsilon$ GOTO C_2
HALT

$[C_2]$ $Y \leftarrow a_n Y$
GOTO B

[C] IF X ENDS a_i GOTO D_i (i=1,...,n)
HALT

$[D_i]$ $X \leftarrow X$-
$Y \leftarrow a_i Y$
GOTO C
} i=1,...,n

Entradas Múltiples

En muchas ocasiones queremos calcular una función $f(u_1, \ldots, u_n)$ que depende de varias variables. Esto se puede hacer de la siguiente forma:

- Transformarla en una función que dependa de un sólo argumento añadiendo un símbolo separador c que no esté en el alfabeto de entrada y calcular $f'(u_1 c \ldots c u_n)$. Se entiende que el cálculo de f es equivalente al cálculo de f'.
- En programas con variables es más sencillo suponer que existen n variables de entrada $X_1, \ldots, X_n$ y que cada una de ellas contiene al principio uno de los argumentos de entrada: $X_i = u_i$.

Equivalencia de Modelos

Teorema

Los siguientes hechos son equivalentes:

- f es parcialmente calculable por una MT
- f es parcialmente caculable por un programa Post-Turing.
- f es parcialmente calculable por un programa con variables.

Los siguientes hechos son equivalentes:

- L es aceptado por una MT
- L es aceptado por un programa Post-Turing.
- L es aceptado por un programa con variables.

Demostración

La demostración se basa en comprobar que los tres modelos se pueden simular entre si. Haremos las siguientes simulaciones:

1. Un programa con variables por un programa Post-Turing.
2. Un programa Post-Turing por una MT.
3. Una MT por un programa con variables.

Programa con Variables -> Programa Post-Turing (1)

Vamos a suponer que las variables son

$$X_1, \ldots, X_m, Z_1, \ldots, Z_k, Y$$

Vamos a suponer $l = m + k + 1$ y vamos a escribir las variables en el mismo orden como

$$V_1, \ldots, V_l$$

El programa Post-Turing va a contener el contenido de las variables de la siguiente forma en la cinta:

$$\#X_1\#\ldots\#X_m\#Z_1\#\ldots\#Z_k\#Y\#$$

que con la nueva notación sería

$$\underset{\uparrow}{\#}V_1\#\ldots\#V_j\#\ldots\#V_l\#$$

suponiendo que las variables nunca contienen un blanco. En otro caso habría que usar otro separador en lugar de $\#$.
Al pincipio de cada instrucción del programa de variables el cabezal de lectura estará situado en el blanco justo a la izquierda de V_1.

Programa con Variables -> Programa Post-Turing (2)

Vamos a simular cada instrucción del programa con variables como una macro del programa Post-Turing. Antes necesitamos una serie de macros:

- La macro GOTO L se expande como:

  IF a_1 GOTO L
  IF a_2 GOTO L
  ⋮
  IF a_n GOTO L

- La macro RIGHT TO NEXT BLANK se expande como

  [A] RIGHT
  IF # GOTO E
  GOTO A

  [E] es la etiqueta de la instrucción inmediatamente después de la macro.

Programa con Variables -> Programa Post-Turing (3)

- La macro LEFT TO NEXT BLANK se expande como

```
[A]     LEFT
        IF # GOTO E
        GOTO A
```

[E] es la etiqueta de la instrucción inmediatamente después de la macro.

- La macro MOVE BLOCK RIGHT se expande como:

```
[C]     LEFT
        IF # GOTO A0
        IF ai GOTO Ai      (i=1,...,n)
[Ai]    RIGHT              }
        PRINT ai           }  i=1,...,n
        LEFT               }
        GOTO C             }
[A0]    RIGHT
        PRINT #
        LEFT
```

$$\ldots\#0011 \quad (\uparrow \text{ sobre el último } 1)$$

$$\ldots\#001\# \quad (\uparrow \text{ sobre el último } \#)$$

Programa con Variables -> Programa Post-Turing (3)

- La macro LEFT TO NEXT BLANK se expande como

```
[A]     LEFT
        IF # GOTO E
        GOTO A
```

[E] es la etiqueta de la instrucción inmediatamente después de la macro.

- La macro MOVE BLOCK RIGHT se expande como:

```
[C]     LEFT
        IF # GOTO A0
        IF ai GOTO Ai      (i=1,...,n)
[Ai]    RIGHT   }
        PRINT ai  } i=1,...,n
        LEFT    }
        GOTO C  }
[A0]    RIGHT
        PRINT #
        LEFT
```

$$\ldots\#0011 \underset{\uparrow}{} \quad \Longrightarrow \quad \ldots\#\underset{\uparrow}{\#}001$$

$$\ldots\#001\underset{\uparrow}{\#}$$

Programa con Variables -> Programa Post-Turing (3)

- La macro LEFT TO NEXT BLANK se expande como

```
[A]     LEFT
        IF # GOTO E
        GOTO A
```

[E] es la etiqueta de la instrucción inmediatamente después de la macro.

- La macro MOVE BLOCK RIGHT se expande como:

```
[C]     LEFT
        IF # GOTO A0
        IF ai GOTO Ai      (i=1,...,n)
[Ai]    RIGHT          }
        PRINT ai       }
        LEFT           }  i=1,...,n
        GOTO C         }
[A0]    RIGHT
        PRINT #
        LEFT
```

$$\ldots\#0011 \underset{\uparrow}{} \quad \Longrightarrow \quad \ldots\#\underset{\uparrow}{\#}001$$

$$\ldots\#001\underset{\uparrow}{\#} \quad \Longrightarrow \quad \ldots\#\underset{\uparrow}{\#}001$$

Programa con Variables -> Programa Post-Turing (4)

- La macro ERASE BLOCK se expande como

```
[A]     RIGHT
        IF # GOTO E
        PRINT #
        GOTO A
```

Si tenemos que repetir la misma instrucción varias veces, esto lo valos a representar con un [i] después de la instrucción donde i es el número de veces que hay que repetir la instrucción. Así RIGHT TO NEXT BLANK [3] es una forma resumida de poner:

```
RIGHT TO NEXT BLANK
RIGHT TO NEXT BLANK
RIGHT TO NEXT BLANK
```

Programa con Variables -> Programa Post-Turing (5)

La instrucción $V_j \leftarrow a_i V_j$ se simula como:

RIGHT TO NEXT BLANK [l]
MOVE BLOCK RIGHT [l-j+1]
RIGHT
PRINT a_i
LEFT TO NEXT BLANK [j]

$$\underset{\uparrow}{\#}\, V_1 \# \ldots \# V_j \# \ldots \# V_l \#$$

Programa con Variables -> Programa Post-Turing (5)

La instrucción $V_j \leftarrow a_i V_j$ se simula como:

```
RIGHT TO NEXT BLANK [l]
MOVE BLOCK RIGHT [l-j+1]
RIGHT
PRINT a_i
LEFT TO NEXT BLANK [j]
```

$$\underset{\uparrow}{\#}\, V_1 \# \ldots \# V_j \# \ldots \# V_l \#$$

$$\# V_1 \# \ldots \# V_j \# \ldots \# V_l \underset{\uparrow}{\#}$$

Programa con Variables -> Programa Post-Turing (5)

La instrucción $V_j \leftarrow a_i V_j$ se simula como:

RIGHT TO NEXT BLANK [l]
MOVE BLOCK RIGHT [l-j+1]
RIGHT
PRINT a_i
LEFT TO NEXT BLANK [j]

$$\begin{array}{l} \underset{\uparrow}{\#}\, V_1 \# \ldots \# V_j \# \ldots \# V_l \# \\ \#\, V_1 \# \ldots \# V_j \# \ldots \# V_l \underset{\uparrow}{\#} \\ \#\, V_1 \# \ldots \underset{\uparrow}{\#} \# V_j \# \ldots \# V_l \# \end{array}$$

Programa con Variables -> Programa Post-Turing (5)

La instrucción $V_j \leftarrow a_i V_j$ se simula como:

```
RIGHT TO NEXT BLANK [l]
MOVE BLOCK RIGHT [l-j+1]
RIGHT
PRINT a_i
LEFT TO NEXT BLANK [j]
```

$$\begin{array}{l}
\underset{\uparrow}{\#}\, V_1 \# \ldots \# V_j \# \ldots \# V_l \# \\
\# V_1 \# \ldots \# V_j \# \ldots \# V_l \underset{\uparrow}{\#} \\
\# V_1 \# \ldots \underset{\uparrow}{\#} \# V_j \# \ldots \# V_l \# \\
\# V_1 \# \ldots \# \underset{\uparrow}{a_i}\, V_j \# \ldots \# V_l \#
\end{array}$$

Programa con Variables -> Programa Post-Turing (5)

La instrucción $V_j \leftarrow a_i V_j$ se simula como:

RIGHT TO NEXT BLANK [l]
MOVE BLOCK RIGHT [l-j+1]
RIGHT
PRINT a_i
LEFT TO NEXT BLANK [j]

$$\begin{array}{l}
\underset{\uparrow}{\#}\, V_1 \# \ldots \# V_j \# \ldots \# V_l \# \\
\#\, V_1 \# \ldots \# V_j \# \ldots \# V_l \underset{\uparrow}{\#} \\
\#\, V_1 \# \ldots \underset{\uparrow}{\#} \# V_j \# \ldots \# V_l \# \\
\#\, V_1 \# \ldots \# \underset{\uparrow}{a_i}\, V_j \# \ldots \# V_l \# \\
\underset{\uparrow}{\#}\, V_1 \# \ldots \# a_i\, V_j \# \ldots \# V_l \#
\end{array}$$

Programa con Variables -> Programa Post-Turing (6)

La instrucción $V_j \leftarrow V_j$- se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF # GOTO C
        MOVE BLOCK RIGHT [j]
        RIGHT
        GOTO E
[C]     LEFT TO NEXT BLANK [j-1]
```

$$\underset{\uparrow}{\#}\, V_1 \,\#\, \ldots \,\#\, 0011 \,\#\, \ldots \,\#\, V_l \,\#$$

Programa con Variables -> Programa Post-Turing (6)

La instrucción $V_j \leftarrow V_j$- se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF # GOTO C
        MOVE BLOCK RIGHT [j]
        RIGHT
        GOTO E
[C]     LEFT TO NEXT BLANK [j-1]
```

$$\begin{array}{l} \#\, V_1\, \# \ldots \# 0011 \# \ldots \# V_l \# \\ \uparrow \\ \#\, V_1\, \# \ldots \# 0011 \# \ldots \# V_l \# \\ \qquad\qquad\quad \uparrow \end{array}$$

Programa con Variables -> Programa Post-Turing (6)

La instrucción $V_j \leftarrow V_j$- se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF # GOTO C
        MOVE BLOCK RIGHT [j]
        RIGHT
        GOTO E
[C]     LEFT TO NEXT BLANK [j-1]
```

$$\begin{array}{l} \underset{\uparrow}{\#}\, V_1 \# \ldots \# 0011 \# \ldots \# V_l \# \\ \# V_1 \# \ldots \# 0011 \underset{\uparrow}{\#} \ldots \# V_l \# \\ \underset{\uparrow}{\#}\, V_1 \# \ldots \# 011 \# \ldots \# V_l \# \end{array}$$

Programa con Variables -> Programa Post-Turing (6)

La instrucción $V_j \leftarrow V_j$- se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF # GOTO C
        MOVE BLOCK RIGHT [j]
        RIGHT
        GOTO E
[C]     LEFT TO NEXT BLANK [j-1]
```

$$\underset{\uparrow}{\#}\, V_1 \# \ldots \# 0011 \# \ldots \# V_l \#$$

$$\# V_1 \# \ldots \# 0011 \underset{\uparrow}{\#} \ldots \# V_l \#$$

$$\underset{\uparrow}{\#}\, V_1 \# \ldots \# 011 \# \ldots \# V_l \#$$

$$\underset{\uparrow}{\#}\, V_1 \# \ldots \# \# \ldots \# V_l \#$$

Programa con Variables -> Programa Post-Turing (6)

La instrucción $V_j \leftarrow V_j$- se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF # GOTO C
        MOVE BLOCK RIGHT [j]
        RIGHT
        GOTO E
[C]     LEFT TO NEXT BLANK [j-1]
```

$$\underset{\uparrow}{\#}\, V_1 \,\#\, \ldots \,\#\, 0011 \,\#\, \ldots \,\#\, V_l \,\#$$

$$\#\, V_1 \,\#\, \ldots \,\#\, 0011 \underset{\uparrow}{\#}\, \ldots \,\#\, V_l \,\#$$

$$\underset{\uparrow}{\#}\, V_1 \,\#\, \ldots \,\#\, 011 \,\#\, \ldots \,\#\, V_l \,\#$$

$$\underset{\uparrow}{\#}\, V_1 \,\#\, \ldots \,\#\,\#\, \ldots \,\#\, V_l \,\#$$

$$\#\, V_1 \,\#\, \ldots \underset{\uparrow}{\#}\#\, \ldots \,\#\, V_l \,\#$$

Programa con Variables -> Programa Post-Turing (6)

La instrucción $V_j \leftarrow V_j$- se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF # GOTO C
        MOVE BLOCK RIGHT [j]
        RIGHT
        GOTO E
[C]     LEFT TO NEXT BLANK [j-1]
```

$$\#\,V_1\,\#\,\ldots\,\#\,0011\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,0011\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,011\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$

Programa con Variables -> Programa Post-Turing (7)

La instrucción IF V_j ENDS a_i GOTO L se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF a_i GOTO C
        GOTO D
[C]     LEFT TO NEXT BLANK [j]
        GOTO L
[D]     RIGHT
        LEFT TO NEXT BLANK [j]
```

$$\underset{\uparrow}{\#}\, V_1 \# \ldots \# 0011 \# \ldots \# V_l \#$$

Por último, la instrucción HALT ha de simularse borrando la entrada y las variables intermedias mediante ERASE BLOCK [l-1] y parando.
Si es para aceptar un lenguaje, no hace falta borrar la entrada y las variables intermedias.

Programa con Variables -> Programa Post-Turing (7)

La instrucción IF V_j ENDS a_i GOTO L se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF a_i GOTO C
        GOTO D
[C]     LEFT TO NEXT BLANK [j]
        GOTO L
[D]     RIGHT
        LEFT TO NEXT BLANK [j]
```

$$\#\,V_1\,\#\,\ldots\,\#\,0011\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,0011\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$

Por último, la instrucción HALT ha de simularse borrando la entrada y las variables intermedias mediante ERASE BLOCK [l-1] y parando.
Si es para aceptar un lenguaje, no hace falta borrar la entrada y las variables intermedias.

Programa con Variables -> Programa Post-Turing (7)

La instrucción IF V_j ENDS a_i GOTO L se simula como:

```
        RIGHT TO NEXT BLANK [j]
        LEFT
        IF a_i GOTO C
        GOTO D
[C]     LEFT TO NEXT BLANK [j]
        GOTO L
[D]     RIGHT
        LEFT TO NEXT BLANK [j]
```

$$\#\,V_1\,\#\,\ldots\,\#\,0011\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,0011\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$
$$\#\,V_1\,\#\,\ldots\,\#\,0111\,\#\,\ldots\,\#\,V_l\,\#$$
$$\uparrow$$

Por último, la instrucción HALT ha de simularse borrando la entrada y las variables intermedias mediante ERASE BLOCK [l-1] y parando.
Si es para aceptar un lenguaje, no hace falta borrar la entrada y las variables intermedias.

Programa Post Turing -> MT (1)

- La MT tendrá los mismos alfabetos de entrada y de trabajo que el programa Post Turing.
- La MT tendrá un estado q_i por cada instrucción I_i del programa Post Turing, más un estado q_f que será el estado final y otro estado q_{k+1} sin transiciones donde k es el número de instrucciones del programa.
- El estado inicial es el que corresponde a la primera instrucción.
- Si la instrucción I_i es PRINT a_i entonces pondremos las transiciones:
  $\delta(q_i, a) = (q_{i+1}, a_i, S), \quad \forall a \in B$
- Si la instrucción I_i es RIGHT entonces pondremos las transiciones:
  $\delta(q_i, a) = (q_{i+1}, a, D), \quad \forall a \in B$
- Si la instrucción I_i es LEFT entonces pondremos las transiciones:
  $\delta(q_i, a) = (q_{i+1}, a, I), \quad \forall a \in B$

Programa Post Turing -> MT (2)

- Si la instrucción I_i es IF a_k GOTO I_j entonces pondremos las transiciones:

  $\delta(q_i, a_k) = (q_j, a_k, S)$ y $\delta(q_i, a) = (q_{i+1}, a, S)$, si $a \neq a_k$.

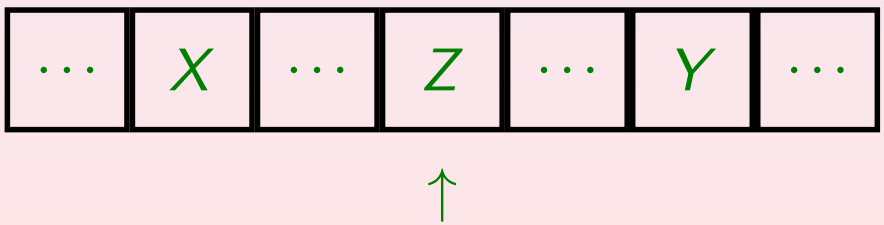
- Si la instrucción I_i es HALT ponemos las transiciones:

  $\delta(q_i, a) = (q_f, a, S), \quad \forall a \in B$

Con estas instrucciones la MT funciona exactamente igual que el programa Post Turing.

MT -> Programa con variables (1)

- La MT y el programa con variables tendrán los mismos alfabetos. Supongamos que el alfabeto de trabajo $B = \{a_1, \ldots, a_n\}$ incluyendo el blanco.
- El programa con variables tendrá tres variables básicas X, Z, Y (más otras auxiliares que aparezcan como expansión de macros) Inicialmente X tendrá la palabra de entrada.
- La idea es que, en cada momento, la variable X contenga lo que hay a la izquierda del cabezal de lectura, Z el símbolo que ve el cabezal de lectura e Y lo que hay a la derecha.



MT -> Programa con variables (2)

Necesitamos una serie de macros:

La macro $V \leftarrow$ -V (eliminar el primer símbolo de V si la variable no es vacía) se expande como (U es una nueva variable auxiliar específica para la macro):

$U \leftarrow \varepsilon$

[A] IF V ENDS a_i GOTO B_i $\quad (i=1,\ldots,n)$

GOTO C

$[B_i]$ $V \leftarrow V$-

$U \leftarrow a_i U$

GOTO A

$\}\ i=1,\ldots,n$

[C] IF U ENDS a_i GOTO D_i $\quad (i=1,\ldots,n)$

GOTO E

$[D_i]$ $U \leftarrow U$-

IF $U \neq \varepsilon$ GOTO F_i

GOTO C

$[F_i]$ $V \leftarrow a_i V$

GOTO C

$\}\ i=1,\ldots,n$

E es la etiqueta de la primera instrucción después de la macro.

Se copia V en U símbolo a símbolo y después se copia U en V excepto el último símbolo (que iría al principio de V)

MT -> Programa con variables (3)

La macro $V \leftarrow Va_j$ (añade a_i) al final de V tiene la expansión:

$U \leftarrow \varepsilon$
$U \leftarrow a_j U$
[A] IF V ENDS a_i GOTO B_i $(i=1,\ldots,n)$
GOTO C
$[B_i]$ $V \leftarrow V\text{-}$, $U \leftarrow a_i U$, GOTO A $\}$ $i=1,\ldots,n$
[C] IF U ENDS a_i GOTO D_i $(i=1,\ldots,n)$
GOTO E
$[D_i]$ $U \leftarrow U\text{-}$, $V \leftarrow a_i V$, GOTO C $\}$ $i=1,\ldots,n$

[E] es la instrucción después de la macro.

Se añade a_j a U y después todos los símbolos de V a U símbolo a símbolo (con eso a_j queda al final de U, después se copian todos los símbolos de U en V (a_j quedaría al final de V)

MT -> Programa con variables (4)

La IF V STARS a_i GOTO L (Si V comienza con a_i seguir por la instruccion [L] tiene la expansión:

	$U \leftarrow \varepsilon$	
[A]	IF V ENDS a_j GOTO B_j	(j=1,. . . ,n)
	GOTO E	
[B_j]	$V \leftarrow V$-	} j=1,. . . ,n
	$U \leftarrow a_j U$	
	IF $V \neq \varepsilon$ GOTO A	
	$V \leftarrow U$	
	GOTO E (si $i \neq j$)	
	GOTO L (si $i = j$)	

E es la etiqueta de la primera instrucción después de la macro.

Se copia V en U símbolo a símbolo hasta que veamos el primero, entonces volvemos a recopiar U en V se ejecuta la acción que corresponda a ese símbolo

MT -> Programa con variables (5)

Inicialmente se ejecutarán las instrucciones:

$Y \leftarrow \varepsilon$
$Z \leftarrow \varepsilon$
$Z \leftarrow \#Z$
[A] IF X ENDS a_i GOTO B_i $(i=1,\ldots,n)$
GOTO A_0
$[B_i]$ $X \leftarrow X-$
$Z \leftarrow \varepsilon$
$Z \leftarrow a_i Z$
IF $X \neq \varepsilon$ GOTO D_i
GOTO A
$[D_i]$ $Y \leftarrow a_i Y$
GOTO A

} $i=1,\ldots,n$

Inicialmente toda la entrada está en X y hay que vaciar X colocar su primer símbolo en Y y el resto en Z.

MT -> Programa con variables (6)

- Asociaremos a cada estado q_i una etiqueta A_i y a cada par (q_i, a_j) otra etiqueta B_{ij} donde se simulará la transición $\delta(q_i, a_j)$. Todas las transiciones no definidas se pueden asociar a la misma etiqueta E.
- En las etiquetas A_i hay las siguientes instrucciones para los estados no finales:

A_i IF Z ENDS a_1 GOTO B_{i1}
IF Z ENDS a_2 GOTO B_{i2}
$\vdots$
IF Z ENDS a_n GOTO B_{in}

En las etiquetas A_i correspondientes a un estado final tendremos las instrucciones que ponen toda la cinta en Y:

IF Z ENDS a_i GOTO C_i (i=1,...,n)

$[C_i]$ $Y \leftarrow a_i Y$
GOTO B
} i=1,...,n

[B] IF X ENDS a_i GOTO D_i (i=1,...,n)
HALT

$[D_i]$ $Y \leftarrow a_i Y$
$X \leftarrow X$-
GOTO B
} i=1,...,n

Antes de parar habría que quitar los blancos a la derecha y a la izquierda de Y. Podeis añadir las instrucciones necesarias como ejercicio.

MT -> Programa con Variables (7)

- Si tenemos la transición $\delta(q_i, a_j) = (q_m, a_k, D)$, entonces ponemos el grupo de instrucciones:

$[B_{ij}]$ $X \leftarrow Xa_k$
$Z \leftarrow \varepsilon$
IF Y STARS a_l GOTO C_{ml} $(l=1,\ldots,n)$
$Z \leftarrow \#Z$
GOTO A_m

$[C_{ml}]$ $Y \leftarrow -Y$
$Z \leftarrow a_l Z$
GOTO A_m
$\Big\}\ l=1,\ldots,n$

- Si tenemos la transición $\delta(q_i, a_j) = (q_m, a_k, I)$, entonces ponemos el grupo de instrucciones:

$[B_{ij}]$ $Y \leftarrow a_k Y$
$Z \leftarrow \varepsilon$
IF X ENDS a_l GOTO D_{ml} $(l=1,\ldots,n)$
$Z \leftarrow \#Z$
GOTO A_m

$[D_{ml}]$ $X \leftarrow X-$
$Z \leftarrow a_l Z$
GOTO A_m
$\Big\}\ l=1,\ldots,n$

El primer bloque de instrucciones correspondientes a las transiciones serán las de la etiqueta A_0 correspondiente al estado inicial q_0. A continuación se pueden poner todas las de los estados y después los grupos de instrucciones que simulan cada una de las transiciones.

Cálculo con Números

- Hasta ahora, hemos visto que los modelos de cálculo trabajan con palabras, pero ¿es posible considerar modelos que trabajan con números?
- Ya hemos visto una aplicación biyectiva entre las palabras de cualquier alfabeto A^* y el conjunto de los números naturales $\mathbb{N}$ que habíamos llamado Z (el número asociado a una palabra), siendo su inversa C (la palabra asociada a un número).
- De esta forma cualquier modelo de cálculo con palabras lo podemos interpretar como un modelo de cálculo con números.Una función numérica $f(n)$ definida sobre los números naturales podemos considerar que se calcula con un modelo de palabras en el que:
  - Ponemos como entrada $u = C(n)$, la palabra que codifica el número n
  - Si la salida es w, la interpretamos como el número $f(n) = Z(w)$ (el número representado por w).

Cálculo con Números

Ejemplo

Si queremos calcular $f(n) = n^2$, representamos cada número n en un alfabeto, p.e. $\{a, b\}$ como la palabra $C(n)$ y hacemos un programa (MT, programa Post-Turing, etc.) que calcule la palabra w que represente el valor n^2.

- Así se puede definir el concepto de función parcialmente calculable de números: cuando exista una codificación en un alfabeto que la calcule.
- Una función es recursiva o calculable cuando es parcialmente calculable y total (está definida en los números naturales).
- También se puede definir el concepto de conjunto numérico recursivo o recursivamente enumerable: cuando el conjunto de todas sus codificaciones en un alfabeto sea recursivo o recursivamente enumerable.
- Estas definiciones no dependen del alfabeto que se use para la codificacion.

Ejemplo: Sumar 1

Vamos a suponer que $A = \{a_1, \ldots, a_n\}$ y que vamos a calcular la función $f(n) = n+1$, donde se supone que el número n se codifica como $C(n)$ en dicho alfabeto.

[B] IF X ENDS a_i GOTO A_i ($i=1,\ldots,n$)
Y $\leftarrow a_1$Y
HALT

[A_i] X $\leftarrow$ X-
Y $\leftarrow a_{i+1}$Y
GOTO C
} $i=1,\ldots,n-1$

[A_n] X $\leftarrow$ X-
Y $\leftarrow a_1$Y
GOTO B

[C] IF X ENDS a_i GOTO D_i ($i=1,\ldots,n$)
HALT

[D_i] X $\leftarrow$ X-
Y $\leftarrow a_i$Y
GOTO C
} $i=1,\ldots,n$

Cálculo con números

Acabamos de ver cómo se puede definir el concepto de función numérica calculable, representando los números como palabras de un alfabeto, pero también es posible diseñar modelos de cálculo que trabajen directamente con números, y definir el concepto de función numérica calculable usando esos modelos.
Eso es lo que se hace a continuación, introduciendo un lenguaje de programación sencillo que se supone que trabaja sólo con números.

Programas con Variables Numéricas

Un programa con variables numéricas tiene los siguientes elementos:

- Un conjunto $X_1, \ldots, X_k$ de variables de entrada, un conjunto finito de variables de trabajo $Z_1, \ldots, Z_l$ y una variable Y de salida.
- Un conjunto de instrucciones opcionalmente etiquetadas de la siguiente forma:
  - A $\leftarrow$ A+1, *Añade 1 al valor entero almacenado en A.*
  - A $\leftarrow$ A-1, *Resta 1 del valor almacenado en A (si es 0 sigue siendo 0).*
  - IF A $\neq$ 0 GOTO L, *Si el valor de A no es 0, seguir por la instrucción con etiqueta L.*
  - HALT, *termina y acepta.*

Se supone que empieza con $X_i = n_i$ donde $u \in \mathbb{N}$ son los valores de entrada y el resto de las variables conteniendo 0. Acepta una entrada si llega a HALT y calcula una función parcial f si llega a HALT con $f(n_1, \ldots, n_k)$ almacenado en Y cuando f está definida y no para en otro caso.

Ejemplo

Programa que tiene como entrada $X = n$ y calcula $Y = n$, excepto para $X = 0$ que $Y = 1$. Es decir, calcula la función $f(n) = n$ si $n \neq 0$ y $f(0) = 1$.

```
[A]   X ← X-1
      Y ← Y-1
      IF X ≠ 0 GOTO A
      HALT
```

Macros

Un conjunto de instrucciones que se usan a menudo se pueden resumir en una nueva instrucción llamada macro. Esto no cambia la definición del lenguaje, solo es una forma reducida de escribir programas.

- La macro $V \leftarrow 0$ con expansión:

  [A] $V \leftarrow V\text{-}1$
  IF $V \neq 0$ GOTO A

- La macro GOTO L con expansión:

  [A] $Z \leftarrow Z+1$
  IF $Z \neq 0$ GOTO L

Macros

- La macro $V \leftarrow U$ tiene la expansión:

```
        V ← 0
        Z ← 0
[A]     IF U ≠ 0 GOTO B
        IF Z ≠ 0 GOTO C
        HALT
[B]     V ← V+1
        Z ← Z+1
        U ← U-1
        GOTO A
[C]     U ← U+1
        Z ← Z-1
        IF Z ≠ 0 GOTO C
        HALT
```

Equivalencia

Teorema

Una función $f : A \to \mathbb{N}$ es parcialmente calculable por un programa con números si y solo si es parcialmente calculable utilizando una codificación con palabras.

- No lo vamos a demostrar. La demostración se basa en simular las instrucciones de un programa con números mediante un programa con cadenas que trabaje con las codificaciones de esos números en un alfabeto y recíprocamente. Algunas de estas simulaciones se han visto ya y otras se han propuesto como ejercicios.
- Se pueden definir conceptos de subconjunto $A \subseteq \mathbb{N}$ r.e. y recursivo (calculable), por ejemplo considerando que un número n es aceptado cuando para $X = n$ el programa termina en HALT con $Y = 1$.

Tesis de Church-Turing

- Hemos visto que estos modelos de cálculo tienen todos la misma capacidad para calcular funciones o aceptar lenguges.
- De hecho, eso mismo se ha comprobado para cualquier modelo *razonable* de cálculo que se ha introducido.
- La generalización de este hecho lleva a la conocida como tesis de Church Turing.

Tesis de Church Turing

Toda función efectivamente calculable (calculable mediante un proceso mecánico bien definido) puede ser calculada por una Máquina de Turing.

Es una tesis que no puede ser comprobada matemáticamente si no se da una definición formal de *efectivamente calculable*, pero eso es precisamente lo que intenta hacer la tesis de Church Turing.
Es una tesis que es aceptada como cierta. Aunque consideremos modelos de computación cuántica, no permitirían hacer cálculos más allá de los realizados por una MT (aunque quizá en menos tiempo que una MT).