Capítulo 5

Propiedades de los Lenguajes de Contexto Libre

Índice General

5.1. Lema de Bombeo.

Al igual que al estudiar los lenguajes regulares, resulta interesante disponer de ciertas herramientas que permitan determinar cuando un lenguaje es de contexto libre, o bien qué conjunto de operaciones pueden ser aplicadas a un lenguaje de contexto libre para obtener como resultado otro lenguaje de contexto libre.

Una de las herramientas más poderosas para determinar que un lenguaje *no es de contexto libre* consiste en verificar que no cumple con las propiedades establecidas por el *lema de bombeo* para LCL.

Lema 5.1 (Lema de Bombeo) *Para todo LCL L, existe una constante* n*, dependiente únicamente de L, tal que si* z *es una cadena de L,* $\mid z \mid \geq n$*, entonces la cadena* z *se puede descomponer como* z = uvwxy *tal que:*

1. $\mid vx \mid \geq 1$,
2. $\mid vwx \mid \leq n$,
3. *Para todo* $i \geq 0$*, las cadenas* uv^iwx^iy *son todas cadenas de L.*

El lema de bombeo resulta de utilidad para demostrar que un lenguaje *no es de contexto libre*, si no lo cumple.

Ejemplo 1:

Demostrar que el lenguaje $\{a^n b^n c^n \mid n \geq 0\}$ *no es un lenguaje de contexto libre.*

Para poder comprobar si se cumplen las condiciones impuestas por el lema de bombeo, se debe localizar una cadena del lenguaje cuya longitud sea mayor que la constante del lema para este lenguaje.

Sea k *esa constante. La cadena* $z = a^k b^k c^k$*, pertenece al lenguaje y su longitud es mayor que* k, $\mid z \mid = 3k$*; por lo tanto,* z *se puede escribir como* uvwxy*, pero ¿cuáles son los símbolos de* z *que forman parte de la cadena* v *y de la cadena* x*?.*

Se deben analizar todos los casos posibles: si al menos uno de estos casos satisficiera las tres condiciones del lema, no se podría demostrar que no se cumple el lema para esta cadena.

1. *Si* v *y* x *están compuestas sólo de símbolos* a*'s, entonces las cadenas* $uv^i wx^i y$*, con* $i \geq 2$ *no pertenecen al lenguaje (contienen más* a*'s que* b*'s y* c*'s).*

$$\underbrace{aa\overset{(k)}{\ldots}aa}_{uvwx}\underbrace{b\overset{(k)}{\ldots}bbcc\overset{(k)}{\ldots}cc}_{y}$$

2. *Si* v *y* x *están compuestas sólo de símbolos* b*'s, entonces las cadenas* $uv^i wx^i y$*, con* $i \geq 2$ *no pertenecen al lenguaje (contienen más* b*'s que* a*'s y* c*'s).*

$$\underbrace{aa\overset{(k)}{\ldots}aa}_{u}\underbrace{bb\overset{(k)}{\ldots}bb}_{vwx}\underbrace{cc\overset{(k)}{\ldots}cc}_{y}$$

3. *Si* v *y* x *están compuestas sólo de símbolos* c*'s, entonces las cadenas* $uv^i wx^i y$*, con* $i \geq 2$ *no pertenecen al lenguaje (contienen más* c*'s que* a*'s y* b*'s).*

$$\underbrace{aa\overset{(k)}{\ldots}aabb\overset{(k)}{\ldots}bb}_{u}\underbrace{cc\overset{(k)}{\ldots}c}_{vwxy}$$

4. *Si* v *está compuesta por* a*'s y* x *por* b*'s, entonces las cadenas* $uv^i wx^i y$*, con* $i \geq 2$ *no pertenecen al lenguaje (contienen más* a*'s y* b*'s que* c*'s).*

$$\underbrace{aa\overset{(k)}{\ldots}aa}_{uv}\underbrace{bb\overset{(k)}{\ldots}bb}_{wx}\underbrace{cc\overset{(k)}{\ldots}cc}_{y}$$

5. *Si* v *está compuesta por* b*'s y* x *por* c*'s, entonces las cadenas* uv^iwx^iy*, con* $i \geq 2$ *no pertenecen al lenguaje (contienen más* b*'s y* c*'s que* a*'s).*

$$\underbrace{\underbrace{aa\ldots aa}^{(k)}\underbrace{bb\ldots bb}^{(k)}}_{u\quad v}\underbrace{\underbrace{cc\ldots cc}^{(k)}}_{wxy}$$

6. *No puede darse el caso que* v *sean símbolos* a*'s y* x *sean* c*'s, pues entonces* $\mid vwx \mid > k$ *(en* w *hay como mínimo* k *símbolos* b*'s).*

$$\underbrace{\underbrace{aa\ldots aa}^{(k)}\underbrace{bb\ldots bb}^{(k)}}_{uv\quad w}\underbrace{\underbrace{cc\ldots cc}^{(k)}}_{xy}$$

7. *No pueden mezclarse símbolos en* v *ni en* x *pues si se hiciese entonces las cadenas* $uv^iwx^iy,\ con\ i \geq 2$ *no pertenecerían al lenguaje (habría símbolos descolocados con respecto al formato de las cadenas del lenguaje)*

Como no hay más opciones posibles para la asignación de símbolos a las cadenas v *y* x *y ninguna de ellas es válida, entonces este lenguaje no cumple el lema de bombeo y, por lo tanto, el lenguaje*

$$\{a^nb^nc^n \mid n \geq 0\}$$

no es un lenguaje de contexto libre.

Ejemplo 2:

Demostrar que el lenguaje $\{a^{2n}b^{2m}c^{2n} \mid m > (100+n) \wedge n \geq 0\}$ *no es un lenguaje de contexto libre.*

Sea k *la constante del lema de bombeo para este lenguaje . La cadena* $z = a^{2k}b^{2(101+k)}c^{2k}$*, pertenece al lenguaje y su longitud es mayor que* k*; por lo tanto,* z *se puede escribir como* uvwxy*, y se considera, a continuación, cada uno de los posibles casos para ver qué símbolos de* z *forman parte de la subcadenas* v *y* x*.*

Como en el ejemplo anterior, si al menos uno de estos casos satisficiera las tres condiciones del lema, no se podría demostrar que no se cumple el lema para esta cadena.

1. *Si* v *y* x *están compuestas sólo de símbolos* a*'s, entonces las cadenas* $uv^iwx^iy,\ con\ i \geq 2$ *no pertenecen al lenguaje (contienen más* a*'s que* c*'s).*

$$\underbrace{\underbrace{aa\ldots aa}^{(2k)}\underbrace{abbbb\ldots bb}^{(2(101+k))}}_{uvwx\quad y}\underbrace{\underbrace{cc\ldots cc}^{(2k)}}$$

2. *Si* v *y* x *están compuestas sólo de símbolos* c*'s, entonces las cadenas* uv^iwx^iy, *con* $i \geq 2$ *no pertenecen al lenguaje (contienen más* c*'s que* a*'s).*

$$\underbrace{aa\ldots a}^{(2k)}\underbrace{abbbbb\ldots b}^{(2(101+k))}\underbrace{ccc\ldots c}^{(2k)}$$
$$u \qquad\qquad vwxy$$

3. *El caso en que* v *y* x *están compuestas sólo de símbolos* b*'s, es distinto; puede parecer que las cadenas* uv^iwx^iy *pertenecen al lenguaje (está aumentando el número de* b*'s), pero eso* ***sólo ocurre*** *si* $i \geq 2$*. En el caso* $i = 0$ *la cadena* uv^iwx^iy $(= uwy)$ ***no*** *pertenece al lenguaje (deja de cumplirse la restricción en el número de* b*'s) y con que haya un único valor de* i *para el que no se satisfagan las tres condiciones del lema al mismo tiempo, es suficiente.*

$$\underbrace{aa\ldots a}^{(2k)}\underbrace{abbbbb\ldots b}^{(2(101+k))}\underbrace{bbcc\ldots cc}^{(2k)}$$
$$u \qquad\qquad vwx \qquad\qquad y$$

4. *Si* v *está compuesta por símbolos* a*'s (al menos uno) y* x *por* b*'s, entonces las cadenas* uv^iwx^iy*, con i* $\geq$ *2 no pertenecen al lenguaje (contienen más* a*'s que* c*'s). En el caso de que* v *fuera la cadena vacía, se aplica el mismo razonamiento que en el caso (3): al bombear con* $i = 0$*, se pierden* b*'s.*

$$\underbrace{aa\ldots a}^{(2k)}\underbrace{abbbbb\ldots b}^{(2(101+k))}\underbrace{bbcc\ldots cc}^{(2k)}$$
$$uv \qquad\qquad wx \qquad\qquad y$$

5. *Algo similar ocurre si* v *está compuesta por* b*'s y* x *por símbolos* c*'s (al menos uno), entonces las cadenas* uv^iwx^iy*, con i* $\geq$ *2 no pertenecen al lenguaje (contienen más* c*'s que* a*'s). Y si* x *fuera la cadena vacía, al aplicar el razonamiento del caso (3) se llega también a que no es posible bombear con cualquier valor de* i *y mantenerse en el lenguaje: falla el valor* $i = 0$*.*

$$\underbrace{aa\ldots a}^{(2k)}\underbrace{abbbbb\ldots b}^{(2(101+k))}\underbrace{bbcc\ldots cc}^{(2k)}$$
$$u \qquad\qquad v \qquad\qquad wxy$$

6. *No puede darse el caso que* v *sean símbolos* a*'s y* x *sean* c*'s, pues entonces* $\mid vwx \mid > k$ *(en* w *hay como mínimo* 202 + 2k *símbolos* b*'s).*

$$\underbrace{aa\ldots a}^{(2k)}\underbrace{abbbbb\ldots b}^{(2(101+k))}\underbrace{bbcc\ldots cc}^{(2k)}$$
$$uv \qquad\qquad w \qquad\qquad xy$$

7. *No pueden mezclarse símbolos en* v *ni en* x *pues si se hiciese entonces las cadenas* uv^iwx^iy, *con* $i \geq 2$ *no pertenecerían al lenguaje (habría símbolos descolocados con respecto al formato de las cadenas del lenguaje)*

Como no hay más opciones posibles y ninguna de las estudiadas es válida, se concluye que el lenguaje no es de contexto libre.

5.2. Propiedades de Clausura.

Las propiedades de clausura para LCL, son aquellas operaciones que, aplicadas sobre LCL, dan como resultado otro LCL. Tal y como ocurría al estudiar los lenguajes regulares, dichas propiedades pueden ayudar a determinar si un lenguaje es, o no es, de contexto libre.

Teorema 5.1 *Los LCL son cerrados bajo las operaciones de unión, concatenación, clausura e inversión.*

Teorema 5.2 *Los LCL son cerrados bajo la operación de sustitución.*

Ejemplo:

Sea $\Sigma = \{a, b\}$ *y* $\Delta = \{0, 1, 2\}$*, tal que*

$$f(a) = L_a = \{0^n 1^n \mid n \geq 1\},\ y$$
$$f(b) = L_b = \{xx^{-1} \mid x \in (0+2)^*\}.$$

Sea $L = \{x \in (a+b)^* \mid S(x,a) = S(x,b)\}$*. Calcular una GCL que genere f(L).*

Sea G_a *la GCL definida por las siguientes producciones:*

$$S_a \to 0S_a1 \mid 01$$

Sea G_b *la GCL definida por las siguientes producciones:*

$$S_b \to 0S_b0 \mid 2S_b2 \mid \lambda$$

Sea G la GCL definida por las siguientes producciones:

$$S \to aSbS \mid bSaS \mid SS \mid \lambda$$

Entonces $G' = \langle \Sigma'_A, \Sigma'_T, P', S \rangle$*, donde*

$$\Sigma'_A = \{S, S_a, S_b\}$$
$$\Sigma'_T = \{0, 1, 2\}$$
$$P' = \{S \to S_aSS_bS \mid S_bSS_aS \mid SS \mid \lambda;$$
$$S_a \to 0S_a1 \mid 01; S_b \to 0S_b0 \mid 2S_b2 \mid \lambda\}$$

Con esta definición, sea y=abab *(por lo tanto,* f(y)=f(a)f(b)f(a)f(b)*) y sea* $x \in f(y)$*, tal que* x=001102200122 *con* $0011 \in f(a)$*,* $0220 \in f(b)$*,* $01 \in f(a)$ *y* $22 \in f(b)$*. La cadena* x *se puede generar en* G' *mediante la secuencia*

$$S \to S_aSS_bS \to S_aS_bS \to S_aS_bS_aSS_bS \to S_aS_bS_aS_bS$$
$$\to S_aS_bS_aS_b = \theta \overset{*}{\Rightarrow} 001102200122$$

Corolario 5.1 *Los LCL son cerrados bajo la operación de homomorfismo (caso particular de la sustitución).*

Teorema 5.3 *Los LCL son cerrados bajo la operación de homomorfismo inverso.*

Teorema 5.4 *Los LCL no son cerrados bajo la operación de intersección.*

Demostración:

Es suficiente con encontrar un contraejemplo.

Sea $L_1 = \{a^n b^n c^m \mid n \geq 1, m \geq 1\}$, el LCL generado por el siguiente conjunto de producciones de contexto libre

$$P_1 = \{S \rightarrow AB; A \rightarrow aAb \mid ab; B \rightarrow cB \mid c\}.$$

Sea $L_2 = \{a^n b^m c^m \mid n \geq 1, m \geq 1\}$, el LCL generado por el siguiente conjunto de producciones de contexto libre

$$P_2 = \{S \rightarrow CD; C \rightarrow aC \mid a; D \rightarrow bDc \mid bc\}.$$

El resultado de la intersección es el lenguaje $L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 1\}$, que por el lema de bombeo se conoce que no es un LCL.

c.q.d.

Corolario 5.2 *Los LCL no son cerrados bajo la operación de complementación.*

Demostración:

Sean L_1 y L_2 LCL; entonces, si la complementación fuese una operación de clausura, también serían LCL los lenguajes $\overline{L_1}$ y $\overline{L_2}$.

Por lo tanto, como la unión de LCL es una operación de clausura, también $\overline{L_1} \cup \overline{L_2}$ es un LCL y, por lo tanto, también lo sería el lenguaje $\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$, lo que se sabe que es falso.

c.q.d.

Teorema 5.5 *Los LCL son cerrados bajo la operación de intersección con lenguajes regulares.*

Este teorema resulta de gran utilidad para determinar que un lenguaje no es de contexto libre.

Ejemplo 1:

Se considera el lenguaje $L = \{x \in (a+b+c)^* \mid S(x,a) = S(x,b) = S(x,c)\}$ *y se quiere estudiar si es o no es un LCL.*

Aplicar el lema de bombeo presenta el inconveniente de que, al no saber cuál es la estructura -el patrón- del lenguaje (¿dónde están a*'s,* b*'s y* c*'s? ¿símbolos iguales agrupados, o intercalados, o...?), no se puede estudiar qué símbolos formarán parte de cada una de las subcadenas que se pueden bombear ni se puede asegurar, por lo tanto, si es posible o no realizar dicho "bombeo".*

La intersección con la expresión regular $a^*b^*c^*$ *puede ayudar. El lenguaje* $L \cap a^*b^*c^*$ *tiene que ser un lenguaje en el que* $S(x,a) = S(x,b) = S(x,c)$ *y, además, primero presente todas las* a*'s, después todas las* b*'s y, por último, todas las* c*'s. Es decir,*

$$L \cap a^*b^*c^* = \{a^n b^n c^n \mid n \geq 0\}$$

Se sabe que dicho lenguaje no es de contexto libre (ejemplo 1 del lema de bombeo); por lo tanto, L *tampoco puede serlo.*

Ejemplo 2:

Determinar si el lenguaje $L = \{xx \mid x \in (a+b)^*\}$ *es un lenguaje de contexto libre. De nuevo, el lenguaje no permite establecer claramente cuál es el patrón de símbolos; por ello, se aplica la propiedad 5.5.*

El lenguaje $a^+b^+a^+b^+$ *es un lenguaje regular. Si L fuese un lenguaje de contexto libre, también lo sería el lenguaje* $L \cap a^+b^+a^+b^+$*. Pero este lenguaje es,*

$$L \cap a^+b^+a^+b^+ = \{a^n b^m a^n b^m \mid n, m \geq 1\}$$

que se demuestra que no es de contexto libre al no cumplir el lema de bombeo para lenguajes de contexto libre.

Sea k *la constante del lema para L. Entonces* $z = a^k b^k a^k b^k$ *pertenece a L y su longitud es mayor que* k*; por lo tanto,* z *se puede escribir como* uvwxy*, pero ¿cuáles son los símbolos de* z *que componen la subcadena* v *y cuáles los que componen la subcadena* x*?. Se analizan todos los casos posibles:*

1. *Si* v *y* x *están compuestas sólo de símbolos* a*'s del principio de* z*, entonces las cadenas* $uv^iwx^iy,\ i \geq 2$ *no pertenecen al lenguaje (contienen más* a*'s en el primer bloque que en el segundo).*

$$\underbrace{aa\overset{(k)}{\ldots}aab\overset{(k)}{\ldots}bbb\overset{(k)}{\ldots}baaab\overset{(k)}{\ldots}bb}_{uvwx}\qquad\qquad\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{y}$$

Algo similar sucedería si se escogieran a*'s del segundo bloque.*

2. *Si* v *y* x *están compuestas sólo de símbolos* b*'s del principio de* z*, entonces las cadenas* $uv^iwx^iy,\ i \geq 2$ *no pertenecen al lenguaje (contienen más* b*'s en el primer bloque que en el segundo).*

$$\underbrace{aa\overset{(k)}{\ldots}aabb\overset{(k)}{\ldots}bbaa\overset{(k)}{\ldots}abb\overset{(k)}{\ldots}bb}_{u\qquad vwx\qquad\qquad y}$$

Algo similar sucedería si se escogieran b*'s del segundo bloque.*

3. *Si* v *está compuesta por* a*'s y* x *por* b*'s, ambas en la primera mitad de* z*, entonces las cadenas* $uv^iwx^iy,\ i \geq 2$ *no pertenecen al lenguaje (contienen más* a*'s y* b*'s en la primera mitad que en la segunda).*

$$\underbrace{aa}_{u}\underbrace{\overset{(k)}{\ldots}a}_{v}\underbrace{abb}_{w}\underbrace{\overset{(k)}{\ldots}bb}_{x}\underbrace{aa\overset{(k)}{\ldots}abb\overset{(k)}{\ldots}bb}_{y}$$

Algo similar sucedería si se escogieran a*'s y* b*'s de la segunda mitad; o* b*'s de la primera mitad y* a*'s de la segunda mitad.*

4. *No puede darse el caso que* v *esté formada por* a*'s de la primera mitad y* x *por* a*'s de la segunda mitad, pues entonces* $\mid vwx \mid > k$*, ya que en* w *hay, como mínimo,* k *símbolos* b*'s.*

$$\underbrace{aa}_{u}\underbrace{\overset{(k)}{\ldots}aabb}_{v}\underbrace{\overset{(k)}{\ldots}bbaa}_{w}\underbrace{\overset{(k)}{\ldots}aabb}_{x}\underbrace{\overset{(k)}{\ldots}bb}_{y}$$

Algo similar sucedería si se escogieran b*'s de la primera mitad y de la segunda mitad.*

5. *No pueden mezclarse símbolos en* v *ni en* x*, pues, si se hiciese, entonces las cadenas* $uv^iwx^iy,\ i \geq 2$ *no pertenecerían al lenguaje (habría símbolos descolocados respecto al formato de las cadenas del lenguaje)*

Como no hay más opciones posibles para la asignación de símbolos a las cadenas v *y* x*, este lenguaje no cumple el lema de bombeo y, por lo tanto, el lenguaje* $\{a^nb^ma^nb^m \mid n, m \geq 1\}$ *no es un lenguaje de contexto libre. Tampoco lo será el lenguaje* $L = \{xx \mid x \in (a+b)^*\}$.

5.3. Problemas Propuestos.

1. Demostrar que si L es un LCL entonces también lo es L^*.

2. Demostrar a qué clase de lenguajes (regular o de contexto libre) pertenecen cada uno de los siguientes lenguajes:

   a) $L_1 = \{(01)^*0^n(01)^*1^n \mid n \geq 0\}$
   b) $L_2 = \{(0+1)^*0^n(0+1)^*1^n \mid n \geq 0\}$
   c) $L_3 = L_1 \cup L_2$
   d) $L_4 = L_1 \cap L_2$

3. Identificar si los siguientes lenguajes son regulares, de contexto libre o no de contexto libre. Si un lenguaje es regular hay que construir el AF o la GR correspondiente; si es de contexto libre, hay que construir el AP o la GCL correspondiente:

   a) $L_1 = \{0^p1^q \mid p\texttt{MOD}4 = q\}$,
   b) $L_2 = \{0^p1^q0^p \mid p\texttt{MOD}4 = q\}$.

4. *a)* Escribir una GCL que genere el lenguaje $L = \{a^ib^jc^k \mid i \neq j \vee j \neq k\}$
   b) Demostrar que el complementario de L no es un LCL.

5. Determinar si los siguientes lenguajes son lenguajes regulares, lenguajes de contexto libre o no son lenguajes de contexto libre:

   a) $L = \{x \in (a+b+c)^* \mid S(x,c) = S(x,a) - S(x,b)\}$.
   b) $(\bigcup_{n\geq 1}((a^*b)^n \cap (ab^*)^n)) \cap a^*b^*$.

6. Determinar si los siguientes lenguajes son o no son LCL

   a) $L_1 = \{a^nb^{n\times n} \mid n \geq 0\}$.
   b) $L_2 = \{x \in (a+b+c)^* \mid S(a,x) = S(b,x) = S(c,x)\}$.
   c) $L_3 = \{a^i \mid$ i es un número primo$\}$.
   d) $L_4 = \{a^nb^mc^{n-m} \mid n \geq m\}$.

7. Identificar si cada uno de los siguientes lenguajes son o no son lenguajes de contexto libre

   a) $\{a^nb^mc^p \mid n = m \wedge m = 2 \times p\}$.
   b) $(a+b+c)^* - \{a^nb^mc^n \mid n, m > 0\}$.

8. Identificar la clase más restrictiva a la que pertenecen los siguientes lenguajes:

   a) $L_1 = \{(a+b)^n(a+b)^n \mid n \geq 1\}$.
   b) $L_2 = \{xyx^{-1}y^{-1} \mid x, y \in (a+b)^*\}$.
   c) $L_3 = \bigcup_{n\geq 0} L^n \mid L = \{xyy^{-1}x^{-1} \mid x, y \in (a+b)^*\}$.
   d) $L = \{xw \mid x, w \in (0+1)^* \wedge S(0,x) = n \wedge S(1,x) = m * n\}$

9. Identificar la clase más restrictiva a la que pertenecen los siguientes lenguajes:

   a) $L_1 = \{a^{2n}b^{2m}c^{2n} \mid m \leq 100 \wedge n \geq 0\}$.

   b) $L_2 = \{a^{2n}b^{2m}c^{2n} \mid m > 100 + n \wedge n \geq 0\}$.

10. Identificar a qué tipo pertenece el siguiente lenguaje:

$$L = \{a^n b^m c^p a^q b^n \mid q = p + m \wedge n, m \geq 1 \wedge p \geq 0\}.$$

11. Considérese el lenguaje siguiente:

$$L = \{a^n b^n c^m \mid m > n\}\ .$$

   a) ¿Es o no es un lenguaje regular? (justificar)

   b) ¿Es o no es un lenguaje de contexto libre? (justificar)

12. *a*) Considérese el lenguaje siguiente: $L = \{0^i 1^j \mid i \neq j \wedge i, j \geq 0\}$. ¿Es o no es un lenguaje regular? (justificar)

    b) El lenguaje del apartado anterior, ¿es o no es un lenguaje de contexto libre? (justificar)

Capítulo 6

Máquinas de Turing

Índice General

6.1. Modelo de Máquina de Turing.

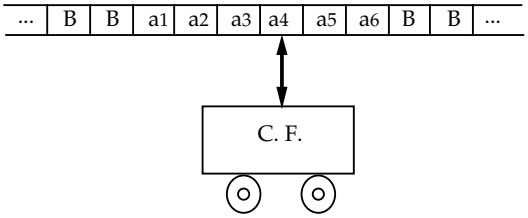
La Máquina de Turing es el último modelo de máquina abstracta que se estudiará. Es un autómata que, a pesar de su simplicidad, es capaz de realizar cualquier cálculo que pueda ser realizado por un computador y no existe ningún otro modelo con mayor poder computacional.

Es por ello que se considera como modelo formal del concepto de Algoritmo, si bien es cierto que existen otros modelos que definen la misma clase de problemas, pudiéndose establecer equivalencias entre todos ellos.

Hay dos puntos de vista para estudiarla:

1. La clase de lenguajes que define: es un *reconocedor de lenguajes de tipo 0*, según la jerarquía de Chomsky.
2. La clase de funciones que computa: es el *solucionador de problemas más potente que hay*.

Como autómata, la Máquina de Turing responde al siguiente modelo mecánico:



1. Tiene una cinta *infinita*. Sus celdas siempre están llenas: o bien por caracteres formando un secuencia de entrada/salida o bien por el carácter especial blanco (`B`).
2. Tiene un cabezal de *lectura/escritura*, que puede desplazarse tanto a la derecha como a la izquierda.
3. Su funcionamiento está basado en un paso elemental, *transición*, que se compone siempre de tres acciones:
   a) Cambio de estado.
   b) Escritura de un símbolo en la celda de la cinta que examina, reemplazando al que hubiera antes.
   c) Desplazamiento a izquierda (L) o derecha (R) una posición.
4. En el *control finito* se controla el funcionamiento: cuál es el estado actual de la máquina y cuáles son las posibles transiciones. El número de estados siempre es *finito*.

La capacidad de escribir es su principal diferencia con otros autómatas, como el autómata finito o el autómata de pilas, y es la que la dota de potencia para reconocer los lenguajes de tipo 0 o para calcular funciones.

Ejemplo 1:

Diseñar una Máquina de Turing para calcular la suma de dos números naturales.

Una forma simple de representar los números naturales es utilizando tantos símbolos como indique su valor numérico. Es decir, si hay que sumar 2 y 3,

se podrían representar, respectivamente, como 11 y 111. Para separar ambos sumandos se podría utilizar un 0. Así, la cinta de entrada del autómata podría tener el siguiente aspecto,

0011011100000BBBBBBBBBB...

Puesto que 11+111 = 11111, una posible forma de realizar el cálculo si el cabezal estuviera sobre la primera celda sería: primero, localizar el primer sumando (es decir, localizar el primer 1) y, segundo, eliminar ese primer 1 del primer sumando con un 0 y avanzar hasta encontrar el 0 de la separación y sustituirlo por un 1. Se obtendría como cadena de salida,

0001111100000BBBBBBBBBB...

Para ello, el autómata debe obedecer a la función de transición:

	0	*1*
q_0	$(q_0,\ 0,\ R)$	$(q_1,\ 0,\ R)$
q_1	$(q_2,\ 1,\ R)$	$(q_1,\ 1,\ R)$
q_2		

En esta tabla, hay tantas columnas como símbolos pueda leer el autómata y tantas filas como estados. Para interpretarla, se asume que cada entrada es la transición asociada al estado indicado por la fila cuando se lee el símbolo asociado a la columna. La tabla anterior es equivalente a la función

$$\begin{aligned}
\mathtt{f}(\mathtt{q}_0, \mathtt{0}) &= (\mathtt{q}_0, \mathtt{0}, \mathtt{R}) \\
\mathtt{f}(\mathtt{q}_0, \mathtt{1}) &= (\mathtt{q}_1, \mathtt{0}, \mathtt{R}) \\
\mathtt{f}(\mathtt{q}_1, \mathtt{1}) &= (\mathtt{q}_1, \mathtt{1}, \mathtt{R}) \\
\mathtt{f}(\mathtt{q}_1, \mathtt{0}) &= (\mathtt{q}_2, \mathtt{1}, \mathtt{R})
\end{aligned}$$

La primera transición, por ejemplo, se lee: "en el estado q_0, cuando se lee un 0, se transita a q_0, se escribe 0 y se mueve el cabezal a la derecha". El estado q_2 es un estado final y no se le asocia ninguna transición.

Ejemplo 2:

Diseñar una Máquina de Turing para determinar si dos números naturales son iguales.

En este caso, se puede representar el primer número mediante 0's y el segundo mediante 1's. El problema se reduciría a comprobar que haya tantos ceros como unos. La cinta de entrada podría tener este aspecto:

0000011111BBBBBB...

Se asume que el cabezal está ajustado a la izquierda del primer número. A partir de esta posición, moviéndose hacia la derecha, es fácil localizar el primer 1. Para poder comprobar que hay tantos ceros como unos, se deberían marcar de alguna forma las parejas que ya se hayan estudiado. Para ello se usará el símbolo X para marcar los ceros y el símbolo Y para marcar los unos. Así, el comportamiento del autómata, en etapas sucesivas, debería producir el siguiente contenido de la cinta:

Inicial:	`0000011111BBBBBB...`
Marcar primera pareja:	`X0000Y1111BBBBBB...`
Marcar segunda pareja:	`XX000YY111BBBBBB...`
Marcar tercera pareja:	`XXX00YYY11BBBBBB...`
Marcar cuarta pareja:	`XXXX0YYYY1BBBBBB...`
Marcar quinta pareja:	`XXXXXYYYYYBBBBBB...`

De esta forma, si al acabar de marcar parejas, sólo quedan X's e Y's en la cinta, seguro que los números eran iguales. Este comportamiento queda descrito por la siguiente función de transición:

	0	*1*	*X*	*Y*	`B`
q_0	(q_1, X, R)			(q_3, Y, R)	
q_1	$(q_1, 0, R)$	(q_2, Y, L)		(q_1, Y, R)	
q_2	$(q_2, 0, L)$		(q_0, X, R)	(q_2, Y, L)	
q_3				(q_3, Y, R)	$(q_4, \mathtt{B}, L)$
q_4					

Como en el ejemplo anterior, no hay transición asociada al estado q_4, que es el final (es al que se llega cuando efectivamente se ha comprobado que ambos números son iguales). Nótese que en la tabla hay entradas, correspondientes a estados no finales, sin transición asociada. En el caso de llegar a alguno de estos huecos, la máquina parará, pero al no parar en el estado q_4, se debe concluir que los números no eran iguales.

Este último ejemplo, además, permite comprender el doble punto de vista del estudio de las Máquinas de Turing. Al fin y al cabo, el segundo ejemplo es una Máquina de Turing que reconoce el lenguaje 0^n1^n.

Definición 6.1 *Una Máquina de Turing es una séptupla,*

$$M = \langle \Sigma,\ Q,\ \Gamma,\ f,\ q_0,\ \mathtt{B},\ F \rangle$$

donde,

Σ *es el alfabeto de entrada,*

Q *es el conjunto de estados. Es finito y no vacío. Se ubica en el Control Finito.*

Γ *es el alfabeto de la cinta,* $\Sigma \subseteq \Gamma - \{B\}$.

f *es la función de transición,*

$$f : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,\ R\}.$$

Partiendo de un estado y de un símbolo, indica la transición a otro estado, el símbolo a escribir en la cinta y el movimiento del cabezal.

q_0 *es el estado inicial,* $q_0 \in Q$.

B *es un símbolo especial que se denomina BLANCO,* $\mathtt{B} \in \Gamma$.

F *es el conjunto de estados finales (aceptadores),* $F \subseteq Q$.

En el primer ejemplo se tendría:

$\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \mathtt{B}\}$, $Q = \{q_0, q_1, q_2\}$, $F = \{q_2\}$ *y f, función de transición, según queda descrita en la tabla,*

y, en el segundo,

$\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, X, Y, \mathtt{B}\}$, $Q = \{q_0, q_1, q_2, q_3, q_4\}$, $F = \{q_4\}$ *y f, función de transición, según queda descrita en la tabla.*

Definición 6.2 (Descripción Instantánea) *Una descripción instantánea es una cadena que pertenece a* $\Gamma^* Q \Gamma^*$.

Las descripciones instantáneas, DI, se utilizan para describir el estado de la máquina. En el contexto del modelo de Máquina de Turing una DI representa la cadena que en un momento dado se encuentra en la cinta y un identificador de estado que, además de indicar en qué estado se halla la máquina, indica la posición del cabezal de L/E: el cabezal siempre está sobre el símbolo inmediatamente a la derecha del identificador de estado. Como la cinta es infinita, sólo se representarán los símbolos significativos.

Convenios para interpretar las DI:

1. Si la transición es del tipo $f(q, x_i) = (p, Y, L)$, la evolución se representa

$$x_1 x_2 \cdots x_{i-1} \mathbf{q} x_i x_{i+1} \cdots x_n \vdash x_1 x_2 \cdots x_{i-2} \mathbf{p} x_{i-1} Y x_{i+1} \cdots x_n$$

2. Si la transición es del tipo $f(q, x_i) = (p, Y, R)$, la evolución se representa

$$x_1 x_2 \cdots x_{i-1} \mathbf{q} x_i x_{i+1} \cdots x_n \vdash x_1 x_2 \cdots x_{i-2} x_{i-1} Y \mathbf{p} x_{i+1} \cdots x_n$$

En el primer ejemplo, la evolución de la MT diseñada (con la entrada indicada) queda descrita por la siguiente secuencia de DIs,

$$q_0 00110111000000\mathtt{B} \vdash 0 q_0 0110111000000\mathtt{B} \vdash 00 q_0 110111000000\mathtt{B} \vdash$$
$$000 q_1 10111000000\mathtt{B} \vdash 0001 q_1 0111000000\mathtt{B} \vdash 00011 q_2 1110000 0\mathtt{B}$$

Definición 6.3 *Dos Descripciones Instantáneas de la Máquina de Turing M, I_1 e I_2, están en relación si desde I_1 se puede alcanzar I_2 en un sólo paso ($I_1 \vdash I_2$).*

El cierre reflexivo-transitivo de una relación se representa como $I_1 \overset{*}{\vdash} I_2$; el cierre transitivo, como $I_1 \overset{+}{\vdash} I_2$.

Las definiciones anteriores permiten definir formalmente qué lenguaje reconoce una Máquina de Turing:

Definición 6.4 *Dada la Máquina de Turing,*

$$M = \langle \Sigma,\ Q,\ \Gamma,\ f,\ q_0,\ \mathtt{B},\ F \rangle$$

el lenguaje asociado a esta máquina, al que se denomina $L(M)$ se define como,

$$L(M) \ = \ \{x \in \Sigma^* \mid q_0 x \ \overset{*}{\vdash} \ \alpha_1 p \alpha_2,\ p \in F,\ \alpha_1, \alpha_2 \in \Gamma^*\}$$

Es decir, $L(M)$ es el conjunto de cadenas de entrada que llevan a la máquina a un estado final, independientemente de la posición que ocupe el cabezal.

6.2. Técnicas para la Construcción de Máquinas de Turing.

El objetivo de esta sección es estudiar técnicas que facilitan la construcción de Máquinas de Turing, pero que no afectan a la potencia computacional del modelo ya que siempre se puede simular la solución obtenida mediante el modelo formal.

6.2.1. Almacenamiento en el Control Finito.

Consiste en asociar a determinados estados (o a todos) campos finitos de información adicional sobre las transiciones en el control finito.

Por lo tanto, un estado no sólo queda definido por su identificador, sino también por un número finito de campos de información (normalmente asociado a símbolos del alfabeto de la cinta). La idea es frenar un crecimiento excesivo de identificadores de estado, que pueda dificultar el entendimiento del comportamiento de la máquina. Suele ser útil en aquellos casos en los que se debe "recordar" qué símbolo se ha leído (y en los que la alternativa a usar esta técnica es realizar transiciones a distintos estados, dependiendo de que se lean distintos símbolos).

Ejemplo 3: Construir una Máquina de Turing capaz de reconocer el lenguaje

$$L = \{x \in \Sigma^+ \mid x = ay,\ a \notin y,\ a \in \Sigma\}$$

es decir, un lenguaje en el que el primer símbolo de la cadena no puede volver a aparecer en el resto. Supóngase que Σ = {0,1}*. Una posible idea es diseñar una MT tal que del estado inicial al leer un* 0 *pase al estado* q_1 *y a leer un* 1 *pase a* $q_2 \ldots$*, pero lo único que motiva esta diferencia entre los estados es el símbolo leído, es decir, la distinción de estados está motivada por la necesidad de poder distinguir de alguna forma entre leer primero un* 1 *o un* 0 *para saber que posteriormente no se va a poder repetir. Entonces, ¿por qué no realizar una transición a un estado que sirva para "recordar" cuál fue el símbolo leído en primer lugar? Por ejemplo,*

$$\begin{aligned}
\mathtt{f}([\mathtt{q}_0, \mathtt{B}], \mathtt{0}) &= ([\mathtt{q}_1, \mathtt{0}], \mathtt{0}, \mathtt{R}) \\
\mathtt{f}([\mathtt{q}_0, \mathtt{B}], \mathtt{1}) &= ([\mathtt{q}_1, \mathtt{1}], \mathtt{1}, \mathtt{R}) \\
\mathtt{f}([\mathtt{q}_1, \mathtt{0}], \mathtt{1}) &= ([\mathtt{q}_1, \mathtt{0}], \mathtt{1}, \mathtt{R}) \\
\mathtt{f}([\mathtt{q}_1, \mathtt{1}], \mathtt{0}) &= ([\mathtt{q}_1, \mathtt{1}], \mathtt{0}, \mathtt{R}) \\
\mathtt{f}([\mathtt{q}_1, \mathtt{0}], \mathtt{B}) &= ([\mathtt{q}_1, \mathtt{B}], \mathtt{B}, \mathtt{R}) \\
\mathtt{f}([\mathtt{q}_1, \mathtt{1}], \mathtt{B}) &= ([\mathtt{q}_1, \mathtt{B}], \mathtt{B}, \mathtt{R})
\end{aligned}$$

Con esto se tendría que Q = {$[\mathtt{q}_0, \mathtt{B}]$, $[\mathtt{q}_1, \mathtt{0}]$, $[\mathtt{q}_1, \mathtt{1}]$, $[\mathtt{q}_1, \mathtt{B}]$}*; el estado inicial es* $[\mathtt{q}_0, \mathtt{B}]$ *y el estado final es* $[\mathtt{q}_1, \mathtt{B}]$.

Este ejemplo puede parecer muy simple; para apreciar realmente la utilidad de esta técnica, se aconseja resolver el mismo problema cuando Σ *coincide con el alfabeto latino, por ejemplo.*

En un Máquina de Turing así diseñada, el conjunto de estados Q se puede ver como el resultado de realizar el producto cartesiano entre un conjunto de identificadores de estados y el conjunto de símbolos que sean significativos a la hora de realizar transiciones. Si se guarda un único campo de información se obtendrían pares [id estado, inf]; si se guardan dos campos de información se obtendrían triplas [id estado, inf1, inf2], y así sucesivamente.

Si se quiere obtener una Máquina de Turing estándar bastará con renombrar los pares (o triplas, etc.) obtenidos al realizar el diseño mediante la técnica descrita.

En el ejemplo anterior, de $Q = \{[\mathtt{q}_0, \mathtt{B}], [\mathtt{q}_1, \mathtt{0}], [\mathtt{q}_1, \mathtt{1}], [\mathtt{q}_1, \mathtt{B}]\}$ *se puede obtener el conjunto* $Q' = \{\mathtt{q}'_0, \mathtt{q}'_1, \mathtt{q}'_2, \mathtt{q}'_3\}$ *en el que el estado inicial es* $\mathtt{q}'_0$ *y el estado final es* $\mathtt{q}'_3$.

Es importante tener en cuenta que la información que se almacena en el control finito siempre será **finita**.

6.2.2. Cintas con Sectores Múltiples y Marcaje de símbolos.

En el diseño de algunas Máquinas de Turing puede resultar interesante considerar que la cinta está dividida en varios sectores; nótese que se tiene una cinta, pero es como si cada celda tuviera distintos "compartimentos". En este caso se asume que el cabezal tiene capacidad para leer/escribir todos los sectores a la vez: cada sector puede tener un símbolo y el cabezal los interpreta todos juntos (sería similar a interpretar que `000000101` es `5`). La ventaja de esta técnica es disminuir el número de símbolos del alfabeto de la cinta: con sólo dos símbolos y k sectores se podría representar lo mismo que con 2^k símbolos, por ejemplo. Dependiendo del sector, puede variar la interpretación.

En general, para describir las Máquinas de Turing así diseñadas se realizan tantos productos cartesianos del alfabeto como sectores aparezcan. El cabezal lee una k_tupla, por lo que el alfabeto de la cinta toma la forma,

$$\Gamma \ = \ \{[\alpha_1, \alpha_2, \ldots, \alpha_k] \mid [\alpha_1, \alpha_2, \ldots, \alpha_k] \in \Gamma_1 \times \Gamma_2 \times \ldots \times \Gamma_k\}$$

lo que lleva a que escribir las transiciones como

$$f(q_i, [\alpha_1, \alpha_2, \ldots, \alpha_k] = (q_j, [\alpha'_1, \alpha'_2, \ldots, \alpha'_k], L/R).$$

Evidentemente, para obtener una Máquina de Turing estándar, basta con renombrar cada una de las k_tuplas, eligiendo el número adecuado de símbolos.

Hay un caso particular de esta técnica, que suele utilizarse muy frecuentemente: la Máquina de Turing tiene dos sectores, de los cuales uno se utiliza para almacenar la cadena de entrada (y obtener la cadena de salida, si es el caso) y el segundo sólo admite una marca especial ($\surd$) ó `B`. La técnica permite marcar símbolos sin borrarlos.

$\surd$	B	$\surd$	B	B	$\cdots$
0	0	1	0	1	$\cdots$

Al utilizar esta técnica hay que redefinir el alfabeto de entrada y el de cinta, como el producto cartesiano entre el alfabeto original y el conjunto $\{\mathtt{B}, \surd\}$.

Como ejemplo de aplicación, considérese el problema propuesto en el segundo ejemplo, pero, en este caso, se marcarán los símbolos en lugar de sobreescribirlos con X e Y. De acuerdo a lo visto, se tendría

$$\Sigma = \{[\mathtt{B}, d] \mid d \in \{0,1\}\},\ \Gamma = \{[X, d] \mid X \in \{\mathtt{B}, \surd\},\ d \in \{0,1,\mathtt{B}\}\}$$

	[B,0]	[B,1]	[$\surd$,0]	[$\surd$,1]	[B,B]
q_0	(q_1, [$\surd$,0], R)			(q_3, [$\surd$,1], R)	
q_1	(q_1, [B,0], R)	(q_2, [$\surd$,1], L)		(q_1, [$\surd$,1], R)	
q_2	(q_2, [B,0], L)		(q_0, [$\surd$,0], R)	(q_2, [$\surd$,1], L)	
q_3				(q_3, [$\surd$,1], R)	(q_4, [B,B], L)
q_4					

6.2.3. Uso de Subrutinas.

La idea es la misma que cuando se trabaja en un lenguaje de alto nivel: aprovechar las ventajas del diseño modular para facilitar el diseño de la Máquina de Turing.

La base será la descomposición de la tarea a realizar en tareas más simples; cada una de estas subtareas se describirá en una tabla de transición propia (la "subrutina"). En la tabla de transición que describe a la Máquina de Turing que resuelve el problema completo habrá estados de "llamada a subrutina", q_{ll}, caracterizados por que suponen la transición al estado inicial de una "subrutina". El estado final de ésta será realmente un "estado de salida" que permite transitar hacia un estado de "return" en la Máquina de Turing "principal".

"Subrutina":

	α	β	$\cdots$
q_i	(q_j,...)		
$\cdots$			$\cdots$
q_f		(q_r,...)	$\cdots$

MT "Principal":

	γ	ω	δ	$\cdots$
q_r	(q_s,...)			$\cdots$
$\cdots$				$\cdots$
q_{ll}		(q_j,...)	(q_i,...)	$\cdots$

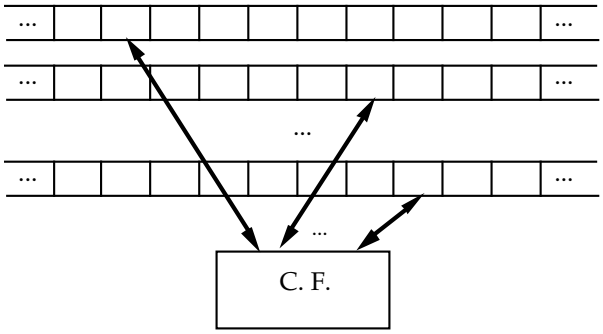
Para obtener una Máquina de Turing estándar, bastaría con reescribir todas las funciones de transición como una única función de transición más grande.

6.3. Máquinas de Turing Modificadas.

La Máquina de Turing es un autómata de definición y funcionamiento muy sencillo. El objetivo de esta sección es estudiar si es posible aumentar su poder computacional añadiéndole nuevos elementos o si esto simplemente reportará ventajas desde el punto de vista de una mayor rapidez en los cálculos o una mayor facilidad para realizarlos.

6.3.1. Máquina de Turing Multicinta.

En este modelo, la máquina de Turing tiene k cintas y k cabezales de L/E,



Sólo hay una entrada de información, en la primera cinta. Los tres pasos asociados a cada transición son ahora:

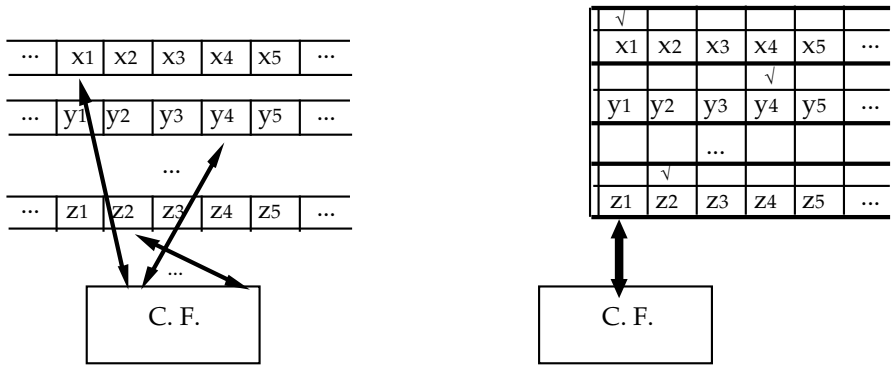
1. transición de estado,
2. escribir un símbolo en cada una de las celdas sobre las que están los cabezales de L/E,
3. el movimiento de cada cabezal es independiente y será R, L ó NADA (Z).

Teorema 6.1 *El lenguaje L es reconocido por una Máquina de Turing Multicinta ⇔ L es reconocido por una Máquina de Turing de una sola cinta.*
(Ambos modelos tienen el mismo poder computacional)

Demostración:(Idea Intuitiva)

1. *Si L es reconocido por una Máquina de Turing de una sola cinta, L es reconocido por una Máquina de Turing Multicinta*. Basta con hacer funcionar una sola cinta de la Máquina de Turing multicinta, la que se utiliza para realizar la entrada de información.
2. *Si L es reconocido por una Máquina de Turing Multicinta, L es reconocido por una Máquina de Turing de una sola cinta*. La idea básica será pasar de un Máquina de Turing multicinta con k cintas a una Máquina de Turing de una sola cinta pero con

2k sectores. Así, por cada cinta de la Máquina de Turing multicinta se tendrá en un sector, la información, y en el otro, una marca que indique la posición del cabezal de L/E de la máquina original.



M1 M2

La máquina M_2 simula el funcionamiento de la máquina M_1 por barridos. Se necesitará un contador de valor inicial k y, suponiendo que el cabezal de M_2 está ajustado sobre la marca situada más a la izquierda, se comienza un movimiento hacia la derecha. Cada vez que se detecte una marca de cabezal de L/E, se decrementa el contador y se almacena el símbolo asociado en el control finito. El barrido acaba cuando el contador valga 0. Habrá k símbolos almacenados en el CF de M_2. Se realizaría la misma transición que se realizaría en M_1. Para poder llevar a cabo los movimientos, habrá que realizar un nuevo barrido hacia la izquierda, trasladando las marcas hacia donde haya que trasladarlas (L, R ó Z) y sobreescribiendo los símbolos que haya que reescribir; una vez completado el barrido, se cambia de estado.

♣

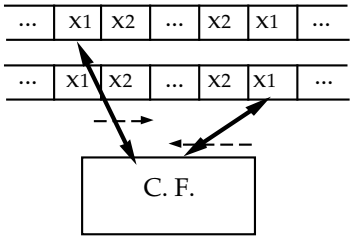
Nótese que un movimiento de M_1 equivale a varios movimientos de M_2. Este modelo tiene el mismo poder computacional, pero suele ser más eficiente que el modelo con una sola cinta.

Ejemplo:

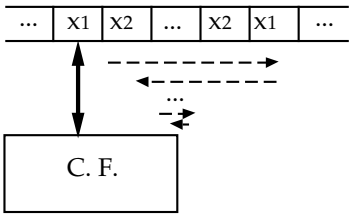
Si se calcula el número de movimientos necesarios para reconocer una cadena del lenguaje

$$L = \{ww^{-1} \mid w \in \Sigma^*\}$$

con respecto a su longitud, en una máquina con 2 cintas el resultado sería lineal y en una máquina con una cinta, cuadrático.



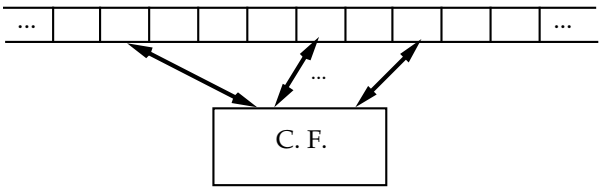
M1: lineal



M2: cuadrático

6.3.2. Máquina de Turing con Múltiples Cabezales.

Tiene k cabezales de L/E, como la multicinta, pero con una sola cinta. Los cabezales operan todos de forma independiente. Como en las Máquinas de Turing multicinta, se admiten movimientos L, R ó Z.



Teorema 6.2 *Un lenguaje L es reconocido por una Máquina de Turing de múltiples cabezales ⇔ L es reconocido por una Máquina de Turing de un cabezal.*
(Ambos modelos tienen el mismo poder computacional)

Demostración:(Idea Intuitiva)

1. *Si L es reconocido por una Máquina de Turing unicabezal, L es reconocido por una Máquina de Turing multicabezal*. Basta con trabajar con un sólo cabezal en la de múltiples cabezales.

2. *Si L es reconocido por una Máquina de Turing multicabezal, L es reconocido por una Máquina de Turing unicabezal*. La simulación es similar a la realizada para las Máquinas de Turing multicinta: se utiliza una cinta con k+1 sectores, k para marcar las posiciones de los distintos cabezales y uno para la información de la cinta.

♣

6.3.3. Máquina de Turing No Determinista.

Es una Máquina de Turing con cinta limitada a la izquierda, que se caracteriza por tener asociada más de una transición desde algún estado con el mismo símbolo,

$$\begin{aligned}
&\mathtt{f}(\mathtt{q_i},\mathtt{a}) = \{(\mathtt{q_i},\mathtt{b},\mathtt{L}),(\mathtt{q_i},\mathtt{a},\mathtt{R}),(\mathtt{q_j},\mathtt{a},\mathtt{R}),(\mathtt{q_j},\mathtt{b},\mathtt{R})\}\\
&\mathtt{f}(\mathtt{q_i},\mathtt{b}) = \{(\mathtt{q_i},\mathtt{b},\mathtt{R})\}\\
&\mathtt{f}(\mathtt{q_j},\mathtt{a}) = \{(\mathtt{q_j},\mathtt{b},\mathtt{L}),(\mathtt{q_j},\mathtt{a},\mathtt{R})\}\\
&\vdots
\end{aligned}$$

El número de transiciones asociado a cada par estado/símbolo *siempre es finito*.

Teorema 6.3 *El lenguaje L es reconocido por una Máquina de Turing No Determinista ⇔ L es reconocido por una Máquina de Turing Determinista.*
(Ambos modelos tienen el mismo poder computacional)

Demostración:(Idea Intuitiva)

1. *Si L es reconocido por una Máquina de Turing Determinista, L es reconocido por una Máquina de Turing No Determinista*. Las Máquinas de Turing deterministas son Máquina de Turing no deterministas en las que sólo hay una transición por cada par estado/símbolo.

2. *Si L es reconocido por una Máquina de Turing No Determinista, L es reconocido por una Máquina de Turing Determinista*. La demostración consiste en determinar cómo una Máquina de Turing determinista puede simular el comportamiento de una Máquina de Turing no determinista. Para ello, en primer lugar, y ya que el número de transiciones asociadas a cada par estado/símbolo es finito, se determina **r**, el número máximo de opciones asociadas a las transiciones (en el ejemplo anterior, r=4). Además, se necesita disponer de una Máquina de Turing determinista con 3 cintas, limitadas a la izquierda.

   La **primera cinta**, recoge la información de entrada, la cadena a reconocer. La **segunda cinta** sirve para llevar la cuenta de qué opciones se van tomando. Para ello, sobre esa cinta hay que ir generando cadenas del alfabeto {1, 2, .., r} por orden numérico:

   - primero, todas las cadenas de longitud 1,

     1, 2, 3, ..., r

   - segundo, todas las cadenas de longitud 2,

     11, 12, 13, ..., 1r, 21, 22, 23, ..., 2r, ..., r1, r2, r3, ..., rr

   - tercero, todas las cadenas de longitud 3,

     111, 112, 113, ..., 11r, 121, 122, 123 ..., 12r, ..., 1r1, 1r2, ..., 1rr, ..., rr1, rr2, rr3, ..., rrr

- y, en general, en el paso i-ésimo se generarían todas las cadenas de longitud i,

11..1, 11..2, ..., 11..r, 11..21, 11..22, ..., 11..r1, .. etc.

Sobre la **tercera cinta** se desarrolla la simulación propiamente dicha. Cada vez que se genera una secuencia en la cinta 2, se copia la cadena de entrada en la cinta 3. La secuencia de la cinta 2 indica qué transición concreta se elige cada vez. Si, por ejemplo, en la cinta 2 está la secuencia

11231..

quiere decir que la primera vez que aplique la función de transición se aplica la primera transición de entre las posibles. La segunda también se aplica la primera; la tercera, se debe aplicar la segunda, la cuarta se debe aplicar la tercera, la quinta se debe aplicar la primera[1]...

Con este "chivato", la máquina operaría sobre la cinta 3. Cada vez que se prueba una secuencia y no se llega a un estado de aceptación, se genera la siguiente y se vuelve a comenzar la simulación. Cuando se encuentra una secuencia que permite aceptar la cadena, la máquina para y acepta.

Si la cadena es aceptada por la Máquina de Turing no determinista, es porque existe una secuencia de aplicaciones de la función de transición que conducen a un estado final. Como en la cinta 2 se van generando todas las posibles secuencias, esta nueva Máquina de Turing determinista alguna vez tendrá que encontrar la correcta y parar aceptando.

6.3.4. Máquinas de Turing Restringidas.

El objetivo de las subsecciones anteriores era poner de relieve que diferentes mejoras físicas sobre el modelo de Máquina de Turing, no se traducen en un mayor poder computacional, sino en una mayor eficiencia en el cálculo o en una mayor simplicidad de diseño.

De la misma forma que podría creerse que introduciendo mejoras físicas en el modelo tendría que aumentar su poder computacional, puede parecer que si se considera un modelo más restringido, éste debe disminuir. Sin embargo, es posible establecer restricciones sobre el modelo sin que esto ocurra. Por ejemplo, el modelo original propuesto por Turing usaba una cinta infinita pero limitada a la izquierda. Y se puede demostrar que esta limitación no supone cambiar el poder computacional del modelo.

[1] Nótese que se producirán secuencias imposibles. Se debe tener en cuenta que r es un máximo y no todas las transiciones tendrán tantas opciones. Al generar todas las secuencias del alfabeto {1,2,..,r}, es posible que una secuencia obligue, por ej., a tomar la tercera opción en un par estado/símbolo que sólo tiene dos. Entonces, se desecha la secuencia y se pasa a la siguiente.

Pero no es la única, también se puede restringir el número de estados o el número de símbolos del alfabeto de la Máquina de Turing, sin variar el poder computacional[2].

En concreto, una Máquina de Turing sin restricciones sobre el alfabeto, con una cinta y sólo tres estados puede simular el comportamiento de cualquier Máquina de Turing. Si lo que se restringe es el número de símbolos, se puede probar el siguiente resultado:

Teorema 6.4 *Si un lenguaje L, $L \subseteq (0+1)^*$, es reconocido por una Máquina de Turing, entonces L es aceptado por una Máquina de Turing con una sola cinta y alfabeto de cinta restringido a los símbolos {0, 1,* B*}.*

Demostración:(Idea intuitiva)

Sea $L = L(M_1)$, con $M_1 = \langle\{0,1\}, Q, \Gamma, f, q_0, B, F\rangle$. Bajo el supuesto de que Γ tiene entre $2^{k-1}+1$ y 2^k símbolos, son suficientes *k* bits para codificar cualquier símbolo[3] de Γ. La Máquina de Turing M_2, con alfabeto de cinta restringido a {0,1,B}, debe simular el comportamiento de M_1.

Para ello, la cinta de M_2 consistirá en una secuencia de códigos representando los símbolos de M_1. El control finito de M_2 debe recordar tanto el estado de M_1 como la posición de su cabezal de L/E, módulo *k*, de forma que sepa cuando se encuentra al principio de un código de un símbolo de M_1.

Al comenzar la simulación de un movimiento de M_1, el cabezal de M_2 debe estar situado al comienzo del código binario de un símbolo de M_1. Moviéndose hacia la derecha, M_2 lee los *k-1* símbolos siguientes para determinar la transición que debe realizar M_1.

Una vez que se sabe qué símbolo escribiría M_1, M_2 reemplaza los *k* símbolos leídos para reemplazarlos por el código del nuevo símbolo, moviéndose hacia la izquierda; se colocaría sobre el comienzo del siguiente código binario a interpretar (dependiendo de que el movimiento de M_1 sea L ó R) y realizaría la misma transición de estado. Si el estado es final, M_2 acepta y, si no, pasa a simular el siguiente movimiento de M_1.

Hay un caso especial en esta simulación, que sería el caso en el que M_1 alcanzara un blanco en la cinta (por ejemplo, llegar a la posición donde se acaba la cadena). La máquina M_2 también alcanzaría una celda en blanco, pero para realizar fielmente la simulación, debe escribir en esa celda y en las *k-1* siguientes el código binario correspondiente al símbolo B de M_1.

Es preciso puntualizar, además, que la entrada de M_2 y M_1 no pueden ser la misma cadena. Es decir, la cadena *w*, $w \in (0+1)^*$, que es la entrada de M_1, debe codificarse también para ser una cadena de entrada que M_2 acepte. Por lo tanto, antes de la simulación

[2]No simultáneamente: si se restringe el alfabeto de cinta, el número de cintas y el número de estados simultáneamente, el resultado sería un número finito de posibles MT que, evidentemente, no pueden tener el mismo poder que el modelo general, con un número infinito de posibles MT.

[3]Hay que tener en cuenta que $\Sigma \subset \Gamma$; es decir, sobre 0, 1 y B también se realizará la codificación.

se debe proceder a reescribir *w*, codificando cada uno de sus símbolos en *k* bits.

♣

Puesto que la misma técnica de codificación se puede aplicar sobre cualquier alfabeto, se establece el siguiente corolario:

Corolario 6.1 *Si un lenguaje L, sobre cualquier alfabeto, es reconocido por una Máquina de Turing, entonces L es aceptado por una Máquina de Turing con alfabeto de cinta restringido a los símbolos* $\{0, 1, \mathtt{B}\}$.

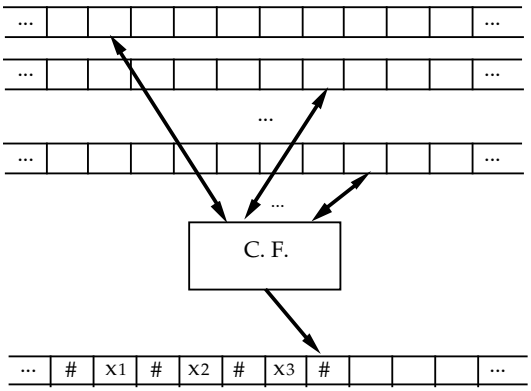
6.4. La Máquina de Turing como Generador.

Las Máquinas de Turing no sólo tienen capacidad para reconocer las cadenas de un lenguaje, sino que también tienen capacidad para generar lenguajes.

La definición formal de la máquina es la misma, con dos salvedades. En un generador el alfabeto Σ debe entenderse como el alfabeto sobre el que se formarán las cadenas del lenguaje. Y, al hablar de las Máquinas de Turing como generadoras, se suele considerar una máquina multicinta con una cinta especial, la *Cinta de Salida* (lo cual es lógico: en una máquina *reconocedora* cobra especial importancia la *entrada*, la cadena a reconocer; en una máquina *generadora*, lo importante es la *salida*, las cadenas que genera).

En la cinta de salida, la máquina irá produciendo todas las cadenas que pertenecen al lenguaje que tiene que generar. Sobre esta cinta sólo se puede escribir. Y, una vez escrito un símbolo, *no se puede reescribir*. El cabezal asociado tiene restringido el movimiento en un sentido, de izquierda a derecha. La imagen más gráfica de esta cinta sería una impresora.

Como ya se ha dicho, las cadenas se formarán a partir del alfabeto Σ, y aparecen en la cinta de salida separadas por un símbolo especial que no pertenezca a dicho alfabeto (normalmente, se utilizará el símbolo #).



Sea M la Máquina de Turing que genera el lenguaje L, L = $\{x_1, x_2, x_3, ...\}$. A M se le denomina *Generador* de L y se utiliza la notación *L = G(M)* para representar a dicho lenguaje.

Cuando se estudia la Máquina de Turing como generador las siguientes afirmaciones son ciertas:

- Si M para, *G(M)* es *finito*. El recíproco no es cierto, en general.
- Las cadenas pueden aparecer repetidas varias veces en la cinta de salida.
- No se asume ningún tipo de orden en la generación de cadenas.
- Una cadena pertenece a G(M) si antes o después aparece en la cinta de salida entre dos símbolos $\#$.

6.4.1. Dos Máquinas de Turing Generadoras Básicas.

Se describen, a continuación, dos generadores básicos para la asignatura y que intervienen en varias construcciones y demostraciones.

El Generador Canónico Se llama Generador Canónico del alfabeto Σ, $G_c(\Sigma^*)$, a la Máquina de Turing capaz de generar todas las cadenas de Σ^*, siguiendo el *Orden Canónico*, que se define de la siguiente forma[4]:

1. Se generan las cadenas por orden creciente de tamaño,
2. Las cadenas del mismo tamaño se generan en orden numérico creciente; para ello, si
$$\Sigma = \{a_0, a_1, a_2, ..., a_{k-1}\},$$
se supone que el símbolo a_i es el *i-ésimo* dígito en base *k*. Por lo tanto, las cadenas de longitud n son los números del 0 al $k^n - 1$.

   Ejemplo:
   Si Σ = {0, 1}, entonces Σ^* se irá generando en el siguiente orden: λ, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, 0000, ... etc.
   Nótese que 0, 00, 000, 0000, ... son distintas cadenas (igual que lo serían a, aa, aaa, aaaa, ... si Σ = {a, b}).

En el orden canónico, cada cadena ocupa una posición determinada, pudiéndose **caracterizar cada cadena por dicha posición**.

[4]Este sería el generador necesario para poder realizar la simulación de una Máquina de Turing No Determinista, tal y como se vio en el teorema 6.3.

El Generador de Pares Se llama Generador de Pares a la Máquina de Turing capaz de generar *todos* los pares $(i, j) \in N \times N$,

$$G(M) = \{(i, j) \mid i, j \in N\}.$$

Si hay que generar todos los pares, la política de generación no puede ser del tipo

```
for i in range(1, ∞):
    for j in range(1, ∞):
        generar (i,j)
```

ya que nunca se llegaría a generar el par (2,1). Para asegurar que todos los pares se generan en tiempo finito, se recurre al orden que proporciona el Triángulo de Tartaglia:

$$\begin{array}{c} (1,1) \\ (1,2)(2,1) \\ (1,3)(2,2)(3,1) \\ (1,4)(2,3)(3,2)(4,1) \\ (1,5)(2,4)(3,3)(4,2)(5,1) \\ \ldots \end{array}$$

En este triángulo, los pares *(i,j)* se generan por orden creciente de sumas, y se podría utilizar la secuencia algorítmica,

```
for s in range(2, ∞):
    i = 1
    j =  s - i
    while (i<s):
         generar (i,j)
         i++
         j--
```

De esta forma, se puede asegurar que el número de pares generado antes de generar el par *(i,j)* es

$$(\sum_{k=1}^{i+j-2} k) + i = \frac{(i+j-2)(i+j-1)}{2} + i\,,$$

por lo tanto, finito. Por lo tanto, el número de pasos necesario para calcular cualquier par *(i,j)* es *finito*.

6.5. Problemas Propuestos.

1. Sea la Máquina de Turing M, $M = \langle \Sigma,\ Q,\ \Gamma,\ f,\ q_1,\ \mathtt{B},\ F \rangle$, con cinta infinita en ambos sentidos y en la que $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \mathtt{B}\}$, Q = $\{q_1, q_2, q_3, q_4, q_5, q_6\}$, F = $\{q_5\}$ y f es la función de transición según esta tabla,

	a	b	B
q_1	(q_1, a, R)	(q_2, b, L)	$(q_5, \mathtt{B}, L)$
q_2	(q_3, a, L)		$(q_6, \mathtt{B}, R)$
q_3	(q_2, a, L)		$(q_4, \mathtt{B}, L)$
q_4			$(q_4, \mathtt{B}, L)$
q_5			
q_6			

a) Describir el funcionamiento de M, realizando secuencias de descripciones instantáneas con las cadenas: `aababbb`, `aaabaabb`, `aaa`.

b) ¿Cuál es el lenguaje $L(M)$?

c) Si llamamos L_P al conjunto de cadenas que hacen que una Máquina de Turing pare ¿cuál es el lenguaje $L_P(M)$?

d) ¿De qué tipo es $L(M)$, LR ó LRE? ¿Por qué?

2. Sea la Máquina de Turing M, $M = \langle \Sigma,\ Q,\ \Gamma,\ f,\ q_1,\ \mathtt{B},\ F \rangle$, que es no determinista y en la que $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \mathtt{B}\}$, Q = $\{q_1, q_2\}$, F = $\{q_2\}$ y f es la función de transición según esta tabla,

	a	b	B
q_1	$\{(q_1, b, Z), (q_1, a, R)\}$	$\{(q_1, a, Z), (q_1, b, R)\}$	$(q_2, \mathtt{B}, R)$
q_2			

a) Presentar *completamente* al menos 4 de las posibles secuencias de descripciones instantáneas que se pueden obtener cuando la entrada es la cadena `aba`. Indicar qué hace la máquina.

b) De acuerdo al apartado anterior, justificar cuál o cuáles serán las secuencias con menor complejidad temporal y cuál o cuáles las de mayor complejidad temporal.

3. Diseñar una MT capaz de calcular la resta propia de dos enteros, que se define como

$$m \dot{-} n = \begin{cases} m - n & si\ m > n \\ 0 & si\ m \leq n \end{cases}$$

4. Diseñar una MT capaz de reconocer el lenguaje $0^n 1^n 2^n,\ n \geq 1$.

5. Diseñar una MT capaz de calcular el producto de dos enteros, $m \times n$.

6. Diseñar una MT capaz de realizar la división entera de dos enteros, $m \,\%\, n$ y $m \,/\, n$.

7. Diseñar una MT con capacidad para reconocer el lenguaje $\{ww^{-1} \mid w \in \Sigma^*\}$.

8. Diseñar una MT con capacidad para reconocer el lenguaje $\{w\$w \mid w \in \Sigma^*\}$.

9. Diseñar una MT con capacidad para reconocer el lenguaje $\{ww \mid w \in \Sigma^*\}$.

10. Diseñar una MT capaz de calcular el mínimo de *k* números enteros, $min(m_1, m_2, \ldots, m_k)$.

11. Diseñar una MT capaz de calcular el máximo de *k* números enteros, $max(m_1, m_2, \ldots, m_k)$.

12. Diseñar una MT capaz de determinar si *k* números enteros son o no son iguales.

13. Diseñar una MT que, dado un número en binario, calcule su complemento a 2.

14. Diseñar una MT que reconozca las cadenas del lenguaje

$$L = \{a^i b^j a^i b^j \mid i, j > 0\}.$$

15. El ADN cromosómico es *bicatenario*: se compone de dos cadenas de nucleótidos enlazados entre sí y dispuestos en forma de hélice. Estos nucleótidos se denominan *Adenina* (A), *Guanina* (G), *Timina* (T) y *Citosina* (C) y en la hélice del ADN sólo se enlazan *Adenina* con *Guanina* y *Timina* con *Citosina*.

    Dado el alfabeto $\{A, G, T, C\}$, diseñar una MT que, dadas dos cadenas sobre dicho alfabeto, determine si es posible o no formar con ellas una cadena de ADN.

    Por ejemplo, la cadenas GATTACA y AGCCGTG podrían formar una hélice.

16. Sea $\Sigma = \{0, 1\}$. Construir una MT que dado un entero *k* construya la *k-ésima* cadena de la serie definida como

    a) La primera cadena es '0',

    b) La cadena *i-ésima* se obtiene a partir de la *i-1* de acuerdo al siguiente procedimiento: si al recorrer la cadena *i-1* se encuentra el símbolo '0', se substituye por el símbolo '1'y si se encuentra el símbolo '1', se añade al final de la cadena un '0'.

    ¿Cuál es la longitud de la cadena obtenida?

17. Dada M, una MT con dos cintas; sea $\Gamma = \{0, 1, \mathtt{B}\}$ su alfabeto de cinta y sea una de sus transiciones de la forma,

$$f(q_i, 0, 1) = (q_j, \{1, L\}, \{1, R\})$$

    Dar las ideas básicas sobre cómo simular la aplicación de dicha transición en una MT de una sola cinta.

18. Supóngase que se dispone de un conjunto de *k* Máquinas de Turing, de forma que cada una de ellas tiene su propio cabezal de lectura/escritura, su propio conjunto de estados y su propia función de transición, pero que trabajan en paralelo compartiendo una única cinta de entrada, así como los alfabetos de entrada y de cinta. ¿Aumentaría este nuevo modelo el poder computacional de las Máquinas de Turing? Justificar la respuesta.

19. Se tiene la idea de realizar una MT con dos cintas. Sobre la primera el cabezal sólo puede leer y en la segunda el cabezal sólo puede escribir. Esta nueva MT ¿tendría menor, mayor o igual poder computacional que el modelo estándar?¿Por qué?.

20. Para cada uno de los lenguajes propuestos, justificar qué modelo modificado proporcionaría más ventajas con respecto al estándar en una "implementación":

    a) $L = \{x \in (0 + 1)^* \mid x = x^{-1}\}$

b) $L = \{x \in (0+1)^* \mid x = zyzy,\ y, z \in (0+1)^*\}$

21. *a*) Diseñar una Máquina de Turing "clásica" que reconozca las cadenas del lenguaje
$$L = \{0^N 10^M 10^{N+M} \mid N, M \geq 1\}.$$
b) De los distintos modelos modificados de Máquinas de Turing, ¿cuál crees que permitiría una "implementación" más rápida de ese lenguaje? ¿Por qué?.

22. *a*) Diseñar una Máquina de Turing "clásica" que reconozca las cadenas del lenguaje
$$L = \{v\#w \mid v, w \in \{a+b\}^* \wedge S(a,w) = 2 \times S(a,v)\}.$$
Es decir, en L están cadenas de la forma $v\#w$ sabiendo que v y w están formadas por a's y b's y que en w hay el doble de a's que en v. Por ejemplo: `abba#baaaba`, `aabb#bbabbbbbbbabaa`, `aaa#aaaaaa`, `b#bbbbbbbb`, `aba#aaaa`, ...
b) De los distintos modelos modificados de Máquinas de Turing, ¿cuál crees que permitiría una "implementación" más rápida de ese lenguaje? ¿Por qué?.

23. *a*) Diseñar una Máquina de Turing "clásica" que ante la siguiente entrada:
`...BBxByBB...`
en la que *x* e *y* son cadenas de $(0+1)^*$, obtenga como salida
`...BBxyBB...`
b) Hemos estudiado distintos modelos modificados de Máquinas de Turing, ¿cuál crees que permitiría una "implementación" más rápida de ese lenguaje? ¿Por qué?.

24. *a*) Diseñar una Máquina de Turing "clásica" que ante la siguiente entrada:
`...BBxByBB...`
en la que *x* e *y* son cadenas de $(0+1)^+$, obtenga como salida
`...BByBxBB...`
b) Hemos estudiado distintos modelos modificados de Máquinas de Turing, ¿cuál crees que permitiría una "implementación" más rápida de ese lenguaje? ¿Por qué?.

25. *a*) Diseñar una Máquina de Turing "clásica" que, dada una cadena de $(0+1)^*$, haga la siguiente operación: la cadena estará formada por bloques de `0`'s separados por `1`'s (vamos, una entrada que sigue el convenio habitual sobre los argumentos de una función de enteros). La máquina debe indicar cuántos bloques de `0`'s hay. Por ejemplo, ante la siguiente entrada:
`...B0001001001000100B...`
debe obtener como salida
`...B00000B...`

ya que se le han pasado 5 argumentos en la cadena de entrada.

b) ¿Qué modelo modificado elegirías para una "implementación" más rápida? ¿Por qué?.

26. *a*) Diseñar una Máquina de Turing "clásica" que, dada una cadena de $(0+1)^*$, haga la siguiente operación: la cadena estará formada por bloques de `0`'s separados por `1`'s (vamos, una entrada que sigue el convenio habitual sobre los argumentos de una función de enteros). La máquina debe indicar si los bloque de ceros están ordenados por orden creciente. Por ejemplo, debe aceptar la siguiente entrada:

`...B00100100010000001000000000B...`

y rechazar la siguiente:

`...B000001000000100B...`

ya que el segundo bloque tiene 6 ceros y el tercero tiene menos, sólo 2 ceros.

b) ¿Qué modelo modificado elegirías para una "implementación" más rápida? ¿Por qué?.

27. Diseñar una Máquina de Turing "clásica" que, dada una cadena de $(0 + 1)^*$, que interpretaremos como un número en binario, le añada un bit de paridad impar; esto es, si el número de bits 1 es par, el de paridad será 1 y si el número de bits 1 es impar, el de paridad será 0 (en ambos casos, el número final de 1's siempre es impar). Ojo, que el bit de paridad siempre es el bit *más significativo*.

Por ejemplo, la MT debe transformar la cadena `...BB00101011B...` en la cadena `...B`**`1`**`00101011B...`.

28. Diseñar una Máquina de Turing que genere las cadenas del lenguaje

$$L = \{a^n b^{2n} a^n \mid n \geq 0\}.$$

29. Construir una MT M, tal que $G(M) = \{0^n 1^n \mid n \geq 0\}$.

30. Construir el generador canónico de Σ^* , siendo $\Sigma = \{0, 1\}$.

31. Diseñar una Máquina de Turing que genere las cadenas del lenguaje

$$L = \{a^n b^{2n} a^n \mid n \geq 0\}.$$

32. Teniendo en cuenta la construcción del Generador Canónico de $(0 + 1)^*$, dar la ideas básicas para la construcción del Generador Canónico de $(a + b + c)^*$.

33. Modificar $G_c(\Sigma^*)$, de forma que ante una entrada *i*, devuelva w_i, *i-ésima* cadena en orden canónico.

34. Supuesto que se conoce la función de transición de una MT M, construir M′ de forma que se comporte como M pero sólo durante *j* pasos.

Capítulo 7

Computabilidad

Índice General

7.1. Lenguajes Recursivos y Funciones Computables.

7.1.1. Introducción.

En 1900, el matemático David Hilbert propuso el llamado *Problema de la Decisión* (*Entscheidungsproblem*), formulado como *"Descubrir un método general para determinar si una fórmula de lógica formal puede o no satisfacerse"*. El interés que podía tener tal problema se debía al trabajo desarrollado desde 1879 por Gottlob Frege, para reducir los enunciados matemáticos a enunciados de la lógica formal: si los enunciados matemáticos se reducen a fórmulas lógicas y hay un método general para determinar si una fórmula se puede satisfacer o no, entonces habría un método general para determinar si un enunciado (teorema) es cierto o no. Enunciados como la Conjetura de Goldbach o la Conjetura de los Números Perfectos, dejarían de ser conjeturas: al aplicarles este método, pasarían a ser teoremas. Bastaría con aplicar tal método "mecánicamente" para determinar si un enunciado

es o no es un teorema. Ese fue el inicio del *"formalismo"*, escuela matemática que pretendía convertir todas las matemáticas en un gran *sistema formal*, un sistema formado por *axiomas* y *reglas*, en el que las expresiones se representan mediante símbolos y sólo se puede operar usando las reglas del sistema.

Se abrió así una vía de investigación que, 36 años más tarde, llevó a la demostración de que tal problema, la determinación de ese método "general", era imposible de resolver. Sin embargo, los resultados obtenidos hasta llegar a esta conclusión son de gran importancia.

El primer resultado de interés surge en 1901, cuando Bertrand Russell descubrió una paradoja irrefutable en la teoría elemental de los conjuntos, que se puede formular como *"El conjunto de los conjuntos que no pertenecen a sí mismos, ¿pertenece a sí mismo?"*. Esta paradoja consiguió anular una de las reglas básicas instituidas por Frege para reducir los enunciados matemáticos a enunciados lógicos. Russell era un gran admirador del trabajo de Frege, y consiguió salvar su trabajo cuando publicó, junto a Whitehead, su obra *"Principia Mathematica"* en la que introduce una *teoría de conjuntos axiomática* que elimina su paradoja (introduce restricciones en cuanto a cómo definir un conjunto, en concreto a conjuntos que se definan a sí mismos).

En 1928, Hilbert y Ackermann definieron el cálculo de predicados de primer orden y en 1931, Kurt Gödel demostró en su tesis doctoral el llamado *Teorema de la Completitud*: el cálculo de predicados de primer orden es completo, es decir, cada predicado válido o es cierto o es falso. Pero, como consecuencia de su trabajo, también pudo demostrar que *un sistema axiomático no es completo* ya que no puede contener todos los enunciados verdaderos de la teoría que pretende formalizar[1].

> *Cualquier sistema consistente de lógica formal lo bastante potente para abarcar los enunciados de toda la aritmética ordinaria tiene que contener enunciados verdaderos que no pueden ser demostrados dentro del sistema.*

La consecuencia inmediata de este teorema fue que la propuesta de Hilbert se eliminó de un plumazo: no puede existir un método que permita decidir si un enunciado arbitrariamente dado es cierto o falso, ya que si lo hubiera demostraría todos los enunciados ciertos y tal demostración es imposible en un sistema incompleto.

A partir de ahí, la investigación continuó por un problema menos ambicioso, el *Problema de la Demostrabilidad*. No es posible encontrar un método que permita decidir si un enunciado es cierto o no, pero *¿existe un único método que permita demostrar, a partir de un sistema de axiomas lógicos, los enunciados matemáticos demostrables?*.

Esta línea de trabajo llevó a Alonzo Church, junto con Kleene y Rosser a definir el $\lambda - calculus$, un lenguaje formal que permite expresar funciones matemáticas (y en el que se basa el lenguaje LISP). Kleene demostró que todas las funciones utilizadas por Gödel, las *funciones recursivas*, pueden expresarse en $\lambda - calculus$. Esto llevó a Church a enunciar que si una función matemática es computable (se puede calcular su valor para todo

[1] Exponer, y entender, el trabajo de Gödel no es sencillo. Una idea aproximada de sus resultados sería saber que llegó en el sistema a construir una cadena que era la formulación equivalente a "*este teorema no se puede demostrar*". Pero Gödel pudo demostrarlo saliendo *fuera* del sistema.

número perteneciente a su dominio de definición), se puede expresar en $\lambda - calculus$. Esta suposición se conoce como la *Tesis de Church* y se suele enunciar como

> *Toda función efectivamente calculable es recursiva.*

Church, además, demostró que en el caso de que hubiera una función matemática expresable en $\lambda - calculus$ pero no computable, entonces no habría método alguno para determinar si un enunciado matemático dado es o no demostrable (y, si no sabemos si un enunciado es o no demostrable, ¿sobre qué se aplicaría el método único de demostración de los enunciados demostrables? ;-). En 1936 publicó una fórmula con esas características, poniendo punto final a la propuesta de Hilbert.

Estos resultados se produjeron de forma paralela al trabajo desarrollado por Alan Turing. Familiarizado con el Problema de la Decisión, lo atacó desde otro punto de vista, intentando formalizar el concepto de *método*. Su definición de método coincide con el concepto actual de *algoritmo*, un *procedimiento que puede ser ejecutado mecánicamente sin intervención creativa alguna*.

De esta idea surgió el modelo de computación que se conoce como Máquina de Turing, que permite la descomposición de cualquier algoritmo en una secuencia de pasos muy simples. Del modelo se desprende la definición formal de computabilidad dada por Turing,

> *Una función es computable si existe una Máquina de Turing que la calcule en un número finito de pasos.*

Trabajando de forma independiente a otros autores, Turing comprendió que había una relación entre el problema de Hilbert y el hecho de asegurar que una función es computable. Gracias a su modelo, Turing pudo llegar a las mismas conclusiones que Church, de una forma más directa. Para ello se basó en los resultados de Georg Cantor sobre conjuntos contables. Cantor había definido un *conjunto contable*, como un conjunto infinito en el que cada objeto se puede asociar, de forma biunívoca, a un elemento del conjunto de los enteros positivos.

Abstrayendo el modelo mecánico, la Máquina de Turing se puede identificar con la función que calcula, lo que permite definirla como una *función de enteros sobre enteros*. Una Máquina de Turing acepta un *conjunto finito* de caracteres de entrada, tiene un *número finito* de estados y, *si la función es computable*, acaba sus cálculos en un *número finito* de pasos. Por lo tanto, cualquier Máquina de Turing se puede describir mediante una *cadena finita de caracteres*, lo que lleva a la conclusión de que el número de máquinas de Turing (por lo tanto, el número de funciones computables) es infinito pero *contable*.

Pero del trabajo de Cantor también se deduce que hay infinitos conjuntos no contables (el de los números reales, por ejemplo). Entre estos, está el conjunto de todas las funciones de enteros sobre enteros. Por lo tanto, si el número de funciones de enteros sobre enteros es no contable y el número de máquinas de Turing es contable, es evidente que es imposible que existan suficientes máquinas de Turing para calcularlas todas. Y sólo son funciones computables las que se pueden calcular mediante una Máquina de Turing.

El objetivo de este tema es, precisamente, formalizar el concepto de función computable. Para ello, primero se definirán las clases de lenguajes (y de funciones) que define la Máquina de Turing según su comportamiento. A continuación, se enunciará la Tesis de Church (ó de Turing-Church), además de presentar otros modelos equivalentes tanto al de Máquina de Turing como al de función recursiva. Para finalizar, se verá la caracterización de estas clases mediante Máquinas de Turing Generadoras, además de ver algunas propiedades de clausura.

Lo que sigue son las definiciones básicas de las clases definidas por el modelo de Máquina de Turing, bien desde el punto de vista de reconocedor de lenguajes, bien desde el punto de vista de calculador de funciones.

7.1.2. La Máquina de Turing como Reconocedor de Lenguajes.

Se recuerda la definición de L(M), lenguaje reconocido por una Máquina de Turing, M:

Definición 7.1 *Dada la Máquina de Turing,*

$$M = \langle \Sigma,\ Q,\ \Gamma,\ f,\ q_0,\ \mathtt{B},\ F \rangle$$

el lenguaje asociado a esta máquina, al que llamaremos $L(M)$ se define como,

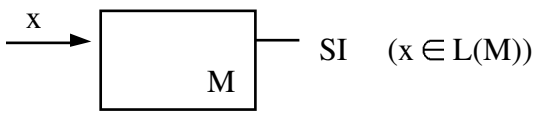
$$L(M) \ = \ \{x \in \Sigma^* \mid q_0 x \overset{*}{\vdash} \alpha_1 p \alpha_2,\ p \in F,\ \alpha_1, \alpha_2 \in \Gamma^*\}$$

(L(M) es el conjunto de cadenas de entrada que llevan a la máquina a un estado final, independientemente de la posición que ocupe el cabezal).

Si la cadena de entrada en una máquina M pertenece a *L(M)*, la máquina M siempre se detiene. Pero lo que ocurre cuando la cadena no pertenece al lenguaje da pie a la clasificación de los lenguajes en *Recursivos* y *Recursivamente Enumerables*.

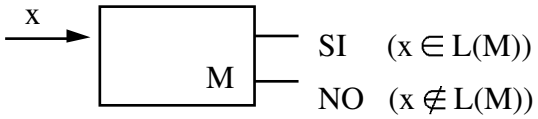
Definición 7.2 (Lenguaje Recursivamente Enumerable) *Se M una Máquina de Turing; se dice que L = L(M) es un Lenguaje Recursivamente Enumerable si*

- $\forall\, x \in L$, *M se detiene en* $q \in F$,
- $\forall\, x \notin L$, *M se detiene en* $q \notin F$ *ó bien M no se detiene.*

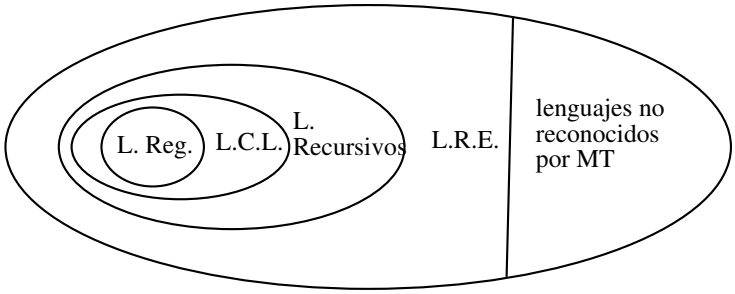


Definición 7.3 (Lenguaje Recursivo) *Se dice que L es un Lenguaje Recursivo si* existe al menos una *Máquina de Turing M, tal que L = L(M) y*

- $\forall\ x \in L$, *M se detiene en* $q \in F$,
- $\forall\ x \notin L$, *M se detiene en* $q \notin F$.



Con esta definición, el conjunto de todos los lenguajes queda de la siguiente forma



7.1.3. La Máquina de Turing como Calculador de Funciones.

Formalmente, la definición de la Máquina de Turing como calculador de funciones, toma la forma de una función k-aria *f*, con argumentos enteros y que devuelve un valor entero,

$$f : Z^k \longrightarrow Z,\ f(i_1, i_2, \ldots, i_k) = i.$$

Para normalizar su funcionamiento como calculador, se toma el convenio de representar los argumentos enteros como cadenas de 0's ($n \rightarrow 0^n$), utilizando el 1 como separador entre los argumentos:

$$f(i_1, i_2, \ldots, i_k) \hookrightarrow f(0^{i_1}10^{i_2}1 \ldots 0^{ik}).$$

El mismo convenio sirve para representar el resultado de la computación.

Definición 7.4 (Función Parcial) *Una función f,* $f : A \longrightarrow B$ *se dice que es una Función Parcial si* $\exists\ C \subseteq A,\ C \neq \emptyset$, *tal que* $\forall\ x \in C,\ \exists\ f(x)$ *(existe un subconjunto no vacío de A en el que todos los elementos tienen imagen calculable).*

Definición 7.5 (Función Total) *Una función f,* $f : A \longrightarrow B$ *se dice que es una Función Total si* $\forall\ x \in A,\ \exists\ f(x)$ *(se puede calcular la imagen para cualquier elemento de A).*

El modelo de Máquina de Turing define las *funciones recursivas parciales* ya que, es posible suministrarle argumentos con los cuales la Máquina de Turing no se detenga. De ahí que se establezca un paralelismo entre la definición de función parcial y la de lenguaje recursivamente enumerable: ambos establecen un funcionamiento de la Máquina de Turing que queda indeterminado, bien sea porque un argumento no tiene imagen, bien sea porque se proporciona a la máquina una cadena que no pertenece al lenguaje. .

Una Máquina de Turing que se detiene ante cualquier entrada define una *función recursiva total*, ya que para cualquier argumento hay una imagen. En términos de lenguajes, una función total se puede asimilar a un lenguaje recursivo.

Nota: todas las operaciones aritméticas habituales (+, *, /, n!, $\log(n)$, 2^n) son funciones recursivas totales.

7.2. La Tesis de Church.

Al estudiar en el capítulo 6 el modelo de Máquina de Turing y sus posibles extensiones, se demostró que cualquier posible mejora introducida no aumenta el poder computacional del modelo básico. Lo mismo se puede aplicar al estudio de otros autómatas que trabajan sobre el mismo principio: el desarrollo de un cálculo mediante la aplicación repetida de pasos finitos y completamente definidos.

De ahí que la Máquina de Turing sea un modelo formal de la noción de algoritmo: sólo se puede considerar como un algoritmo lo que sea posible realizar mediante una Máquina de Turing. Lo que Gödel demostró en 1931 es que en cualquier sistema de axiomas matemático hay enunciados que son ciertos pero cuya veracidad no se puede probar dentro del sistema. Church demostró en 1936 que no hay un método mecánico general para decidir si un enunciado es o no es demostrable. Ello hizo necesario disponer de un substituto matemático y exacto para la noción, intuitiva e informal, de computabilidad mecánica; Church utilizó su propio sistema, el $\lambda - calculus$. Turing llegó de forma independiente a los mismos resultados al mismo tiempo. Aún más: su explicación lógica del concepto de función computable en términos de una máquina abstracta, es superior y más simple que la de Church. Este modelo ha permitido probar los resultados sobre la incomputabilidad y desarrollar el teorema de incompletitud de Gödel, y sus consecuencias, en su totalidad.

El principio de que las Máquinas de Turing son versiones formales de algoritmos y que ningún procedimiento computacional se puede considerar un algoritmo salvo que sea posible presentarlo como una Máquina de Turing, es la versión informática de la Tesis de Church, o Tesis de Turing-Church, que ya se comentó en la Introducción:

La noción intuitiva de función computable puede identificarse con las funciones recursivas parciales. Las Máquinas de Turing tienen el poder suficiente para calcular cualquier función recursiva parcial.

En esta tesis, la noción de función computable no pone límites ni al número de pasos necesarios, ni a la cantidad de espacio de almacenamiento necesario, para desarrollar la computación. No hay que confundir *computable* con *efectivamente computable*.

Esta es una tesis, no un teorema ni un resultado matemático: sólo establece la correspondencia entre un concepto informal (la computabilidad) y un concepto matemático (las funciones recursivas parciales). Es, por lo tanto, teóricamente posible que la Tesis de Church pueda quedar obsoleta en el futuro, si se propone un modelo alternativo para la noción de computabilidad que pueda desarrollar cálculos que no pueden realizarse mediante la Máquina de Turing. Pero no se considera probable.

De hecho, hay una serie de modelos lógicos desarrollados ($\lambda - calculus$, Sistemas de Post, Máquinas de Turing,...) y todos definen la misma clase de funciones, las funciones recursivas parciales. De entre estos modelos, hay uno más cercano al punto de vista de los programadores, el modelo RAM.

7.2.1. El Modelo RAM.

El modelo RAM (*Random Access Machine*), consiste en un *número infinito* de palabras de memoria, *numeradas* (0, 1, 2, ...), cada una de las cuales puede almacenar cualquier número entero, y un *número finito de registros aritméticos* capaces de almacenar cualquier entero. Los enteros podrían entenderse como las instrucciones (codificadas en binario) de un computador.

Eligiendo correctamente el conjunto de instrucciones y el tamaño del máximo entero representable en memoria, el modelo RAM puede simular cualquier computador existente. Veremos a continuación, que la Máquina de Turing tiene el mismo poder computacional que el modelo RAM.

Teorema 7.1 *Una Máquina de Turing puede simular una RAM, supuesto que las instrucciones RAM pueden ser simuladas (las conocemos y sabemos interpretar cada una de ella) por una Máquina de Turing.*

Demostración: (Idea Intuitiva)

La Máquina de Turing necesaria para la simulación será una Máquina de Turing multicinta. En una cinta, se almacenan las palabras de memoria de la RAM que tienen valores. El formato para almacenarlas podría ser

$$\mathtt{B}\#0 * v_0\#1 * v_1\#10 * v_2\#...\#i * v_i\#...$$

donde con v_i se representa el contenido, en binario, de la palabra *i* de memoria de la RAM. Nótese que, en un momento dado, sólo se está usando un número finito de palabras de

memoria; por lo tanto, el número de celdas ocupadas en la Máquina de Turing también será finito.

Para almacenar los contenidos de los registros de la RAM, la Máquina de Turing utilizará tantas cintas como registros haya (el número de registros es finito) . Además, en otras dos cintas se almacenará el valor del Contador de Posición, que contiene el número de la palabra en la que está la próxima instrucción, y el valor del Registro de Direcciones de Memoria, en el cual puede colocarse el número de una palabra de memoria.

¿Cómo podría funcionar? Se asume, por ejemplo, que en cada palabra 10 bits indican una instrucción standard (`LOAD`, `STORE`, `ADD`,...) y los demás son la dirección de un operando.

En un momento dado, el contador de posición contiene el valor *i*, en binario. Habría que buscar, en la primera cinta, la secuencia $\#i*$. Si no la encuentra, no hay instrucción y la Máquina de Turing para (la RAM también pararía, en este caso).

Si la encuentra, se examina la información entre el símbolo * y el siguiente símbolo #. Supóngase que es el equivalente a "`ADD reg 2`" y un número *j* en binario (sumar al registro 2 el contenido de la posición j). Se copia el valor de *j* en la cinta usada como registro de direcciones de memoria. A continuación, se busca la secuencia $\#j*$ en la primera cinta. Si no se encuentra, se asume el valor 0; si se encuentra se suma el valor de v_j a los contenidos de la cinta que se usa como registro 2. Se incrementa el valor de la cinta usada como contador de posición. Y se pasaría a la siguiente instrucción.

♣

7.3. Caracterización de L.R. y L.R.E.

Si una función computable se identifica con una función recursiva parcial o, lo que es lo mismo, queda descrita por un lenguaje recursivamente enumerable, es sencillo comprender la importancia de poder clasificar a un lenguaje dentro de esta clase. Para demostrar que un lenguaje es recursivamente enumerable, basta con construir una Máquina de Turing capaz de reconoce sus cadenas; si, además, dicha Máquina tiene la parada garantizada, es decir, rechaza las cadenas que no pertenecen al lenguaje, se habrá demostrado que el lenguaje es recursivo.

Existe otra forma de caracterizar el carácter recursivo o recursivamente enumerable de un lenguaje, utilizando generadores. Al fin y al cabo, los lenguajes recursivamente enumerables reciben este nombre por influencia del idioma inglés (en este idioma, un generador se conoce como *enumerator*).

7.3.1. Caracterización de los L.R.E. mediante Generadores.

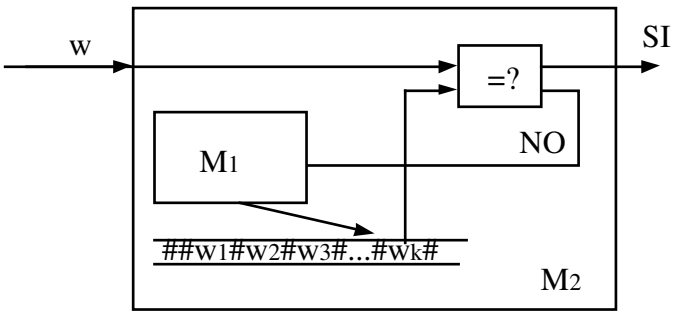
El objetivo de este apartado es llegar a demostrar que se puede afirmar que todo lenguaje que se pueda generar mediante una Máquina de Turing es un L.R.E.; además, cualquier L.R.E. podrá ser generado por una Máquina de Turing.

Lema 7.1 *Si un lenguaje L es generado por una Máquina de Turing M_1, L = G(M_1), entonces L es un L.R.E. ($\exists\ M_2 \mid L = L(M_2)$).*

Demostración:

Idea básica: si se puede garantizar la construcción de M_2 que reconozca L, entonces L es un L.R.E.

Para demostrar que L es L.R.E., hay que construir una Máquina de Turing M_2 con una cinta más que M_1, que será una *cinta de entrada*. Para reconocer una cadena introducida en la cinta de entrada, el comportamiento de M_2 sería el siguiente: M_2 se construye para simular el comportamiento de M_1; además, llegado el momento en que M_1 imprime el símbolo # sobre la cinta de salida (es decir, después de producir una nueva cadena de L), se compara la cadena generada con la cadena de entrada. Si son iguales, M_2 acepta; si no, continúa la simulación, pasando a generar la siguiente cadena.



Esta construcción asegura que M_2 sólo acepta la cadena de entrada cuando es igual a una cadena generada por M_1, es decir, una cadena que pertenece a G(M_1); por lo tanto, se sigue que L(M_2) = G(M_1). Además, el lenguaje reconocido por M_2 es recursivamente enumerable, ya que sólo se puede asegurar que *la máquina acepta la cadena si pertenece al lenguaje*.

c.q.d.

También es posible demostrar el recíproco, es decir, si L es reconocido por alguna Máquina de Turing, entonces L se puede generar mediante una Máquina de Turing. La idea básica, en este caso, es la siguiente: conocido el alfabeto sobre el que se forman las cadenas de L, se podrían generar todas las cadenas sobre ese alfabeto. Puesto que se dispone de una Máquina de Turing que reconoce, entre todas, las que pertenecen al lenguaje, se puede generar éste completamente.

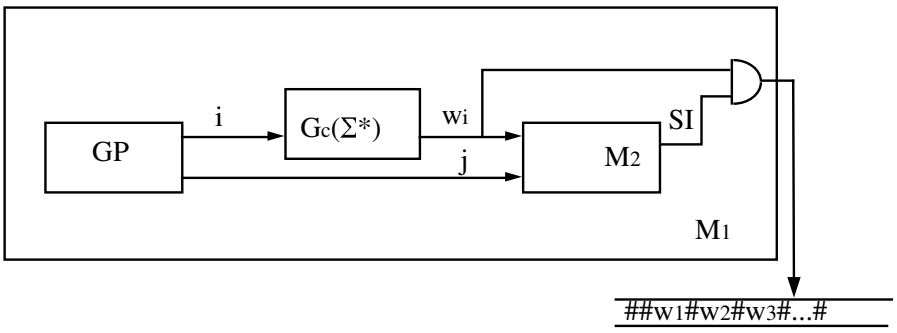
Para desarrollar esta idea, se utiliza el Generador Canónico, ya que, dado un alfabeto Σ, genera todas las cadenas que pertenecen a Σ^*. Pero hay que tener en cuenta que el lenguaje es recursivamente enumerable: por lo tanto, si w_i no perteneciera a L, es posible que la Máquina de Turing que reconoce L no pare nunca al trabajar sobre dicha cadena. Esto supondría que nunca se llegaría a trabajar sobre w_{i+1}, w_{i+2}, ... que puede que sí pertenezcan. Esta dificultad se puede solventar gracias al Generador de Pares, tal y como se verá a continuación.

Teorema 7.2 *Un lenguaje L es L.R.E.* ($\exists\ M_2 \mid L = L(M_2)$) $\Leftrightarrow$ *L se puede generar* ($\exists\ M_1 \mid L = G(M_1)$).

Demostración:

"⇐": Lema 7.1

"⇒": Por construcción de la Máquina de Turing M_1 de la siguiente forma:



Cada vez que el Generador de Pares produce un par *(i,j)*, el valor de *i* se introduce como entrada al Generador Canónico para que produzca w_i, la *i-ésima* cadena en orden canónico. Esta cadena se suministra a M_2, junto con el valor de *j*. Si M_2 reconoce la cadena w_i en exactamente *j* pasos, entonces M_1, genera la cadena w_i (la imprime en la cinta de salida).

Si una determinada cadena w pertenece a L, ocupará una determinada posición en orden canónico en el lenguaje Σ^* y será reconocida en un número determinado y finito de pasos; es decir, w se corresponderá con algún par *(i,j)*. Como cualquier par es generado en un número finito de pasos por el generador de pares, si $w \in L$ se generará en un número finito de pasos. Por lo tanto, cualquier cadena del lenguaje puede ser generada por M_1 en tiempo finito.

c.q.d.

Corolario 7.1 *Si L es un L.R.E., entonces hay un generador de L que produce cada cadena exactamente una vez (*M_1*, por ejemplo).*

7.3.2. Caracterización de los L.R. mediante Generadores.

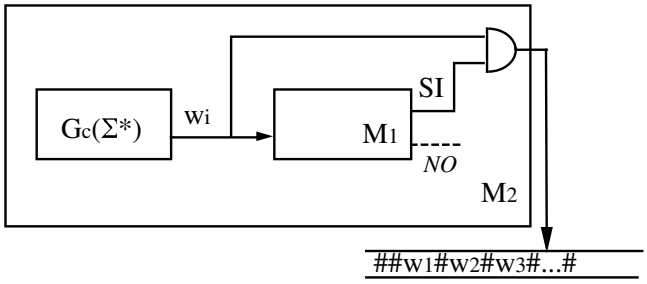
Tal y como se ha visto, cualquier lenguaje que se puede generar mediante una Máquina de Turing es un L.R.E. (y viceversa). A continuación se verá que si, además, se puede generar en orden canónico entonces es un L.R. (y viceversa: cualquier L.R. se puede generar en orden canónico).

Lema 7.2 *Si L es recursivo ($\exists\ M_1 \mid L = L(M_1)$ y M_1 siempre para), entonces existe un generador de L ($\exists\ M_2 \mid L = G(M_2)$) que imprime las cadenas de L en orden canónico.*

Demostración:

Idea básica: Construcción de M_2 a partir de la MT M_1.

El lenguaje L será un subconjunto de todas las cadenas que se puedan construir a partir de un cierto alfabeto, Σ. Se construye M_2, cuyo comportamiento queda descrito en el siguiente diagrama de bloques:



Se generan las cadenas de Σ^* en orden canónico. Después de generar cada cadena, M_2 simula el comportamiento de M_1 con la cadena generada. Si M_1 la acepta, M_2 la escribe en la cinta de salida. Ya que M_1 siempre para, M_2 podrá generar cada cadena en tiempo finito. Y puesto que las cadenas se producen en orden canónico, aparecerán también en orden canónico en la cinta de salida.

c.q.d.

Para demostrar el recíproco, es decir, "*dado un lenguaje L, tal que L es generado por una Máquina de Turing M_2 en orden canónico, L=G(M_2), entonces L es un L.R.*", se podría seguir la misma pauta que en las demostraciones anteriores: construir una Máquina de Turing M_1, basada en M_2, que acepte las cadenas de L y rechace las cadenas que no pertenezcan a dicho lenguaje. La construcción se podría basar en el esquema mostrado en la figura 7.1.

La máquina M_1 simularía el comportamiento de M_2: cada cadena que se genere en la cinta de salida de M_2, se compara con la cadena que hay en la cinta de entrada de M_1. Si ambas son iguales, la cadena es aceptada. Si son distintas, se comprueba que la última cadena generada no sea superior, en orden canónico, a la cadena de la cinta de entrada; en

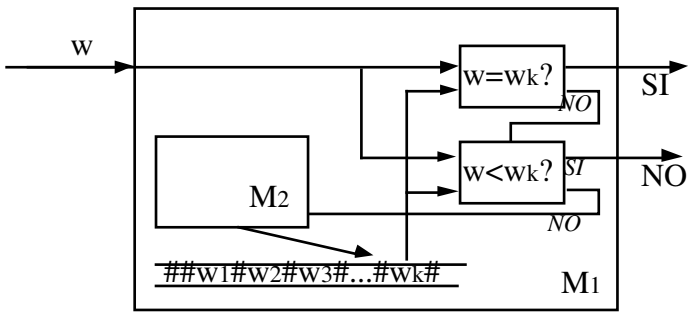


Figura 7.1: Máquina de Turing para reconocer un lenguaje generado en orden canónico.

este caso, dicha cadena pude ser rechazada, ya que se tiene la seguridad de que no pertenece al lenguaje puesto que, si así fuera, ya habría aparecido en la cinta de M_2.

Pero esta construcción presenta un problema

- Si L es un lenguaje infinito, está garantizado que M_1 para, tal y como se ha hecho la construcción.
- Pero si L es finito, no se sabe cuál será el comportamiento de M_2 cuando acabe de generar cadenas, puede parar o puede seguir funcionando. En este último caso, la construcción no permite garantizar que M_1 pare siempre.

Pero cualquier lenguaje finito tiene asociado una Máquina de Turing con parada asegurada (un Autómata Finito). Por lo tanto, se garantiza que, en cualquiera de los dos casos, tanto si L es finito como infinito, existe una Máquina de Turing que siempre para (aunque no se pueda dar una "receta" única para su construcción). Según esto, se puede enunciar el siguiente teorema,

Teorema 7.3 *Un lenguaje L es L.R. ($\exists\ M_1 \mid L = L(M_1)\ y\ M_1\ siempre\ para$) $\Leftrightarrow$ L se puede generar ($\exists\ M_2 \mid L = G(M_2)$) en orden canónico.*

Demostración:

"⇒": Lema 7.2

"⇐": Si L es infinito, entonces L es reconocido por la Máquina de Turing M_1 descrita anteriormente. Si L es finito, entonces hay algún autómata finito que acepta L; por lo tanto, en cualquiera de los dos casos, existe una Máquina de Turing que acepta L y siempre para.

c.q.d.

7.4. Propiedades de Lenguajes Recursivos y Lenguajes Recursivamente Enumerables.

El objetivo de esta sección es estudiar propiedades que pueden ayudar a determinar cuándo un lenguaje es o no es un lenguaje recursivo o recursivamente enumerable. El primer teorema muestra cómo estudiar una propiedad de clausura. Siguiendo el mismo modelo se pueden estudiar la intersección, la diferencia, la clausura transitiva, etc.

Teorema 7.4

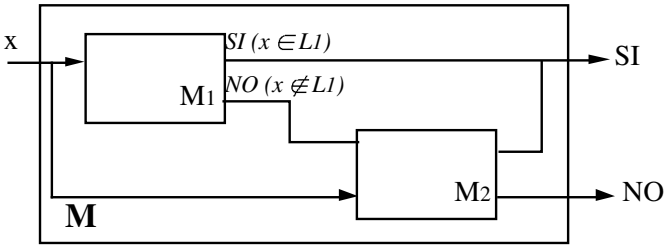
1. *La unión de dos lenguajes recursivos es un lenguaje recursivo.*
2. *La unión de dos lenguajes recursivamente enumerables es un lenguaje recursivamente enumerable.*

Demostración:

1. Sean L_1 y L_2, L.R. $\Rightarrow \exists\ M_1, M_2 \mid L_1 = L(M_1), L_2 = L(M_2)$ y M_1 y M_2 siempre paran:

   - si $x \in L_1$, M_1 acepta la cadena *x* y si $x \notin L_1$, M_1 rechaza la cadena *x*.
   - si $x \in L_2$, M_2 acepta la cadena *x* y si $x \notin L_2$, M_2 rechaza la cadena *x*.

   Sea $L = L_1 \cup L_2$, ¿$\exists\ M \mid L = L(M)$ y M siempre para? Sí, construyendo la máquina que se presenta en la figura:



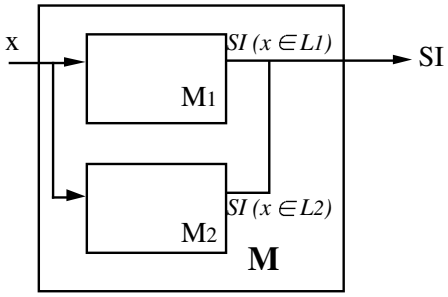
   La construcción es correcta puesto que

   - si $x \in L_1\ OR\ x \in L_2 \Rightarrow x \in L_1 \cup L_2 = L$ (M_1 ó M_2 aceptan $\Rightarrow$ M acepta)
   - si $x \notin L_1\ AND\ x \notin L_2 \Rightarrow x \notin L_1 \cup L_2 = L$ (M_1 y M_2 rechazan $\Rightarrow$ M rechaza).

   Por lo tanto, M siempre para y si $x \in L$, M acepta y si $x \notin L$, M rechaza.

2. Sean L_1 y L_2, L.R.E. $\Rightarrow \exists\ M_1, M_2 \mid L_1 = L(M_1), L_2 = L(M_2)$, es decir, si $x \in L_1$, M_1 acepta y si $x \in L_2$, M_2 acepta.

   Sea $L = L_1 \cup L_2$, ¿$\exists\ M \mid L = L(M)$? Sí, construyendo M de la siguiente forma:



La construcción es correcta puesto que si $x \in L_1 \; OR \; x \in L_2 \Rightarrow x \in L_1 \cup L_2 = L$ (M_1 ó M_2 aceptan $\Rightarrow$ M acepta) por lo tanto, si $x \in L$, M acepta. El comportamiento de M queda indeterminado si $x \notin L_1$ AND $x \notin L_2 \Rightarrow x \notin L_1 \cup L_2 = L$, pero se intenta caracterizar un lenguaje recursivamente enumerable.

c.q.d.

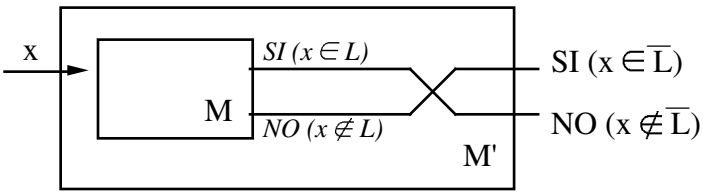
Los dos siguientes teoremas se refieren a la operación complemento:

Teorema 7.5 *El complemento de un lenguaje recursivo es un lenguaje recursivo.*

Demostración:

Sea L un L.R. $\Rightarrow \exists$ M | L = L(M) y M siempre para: si $x \in L$, M acepta la cadena x y si $x \notin L$, M rechaza la cadena x.

Sea $\bar{L}$ = Σ^* - L, ¿$\exists$ M′ | $\bar{L}$ = L(M′) y M′ siempre para? Sí, construyendo la máquina M′ de la siguiente forma:



La construcción es correcta puesto que

- si $x \in L \Rightarrow x \notin \bar{L}$ (M acepta $\Rightarrow$ M′ rechaza)
- si $x \notin L \Rightarrow x \in \bar{L}$ (M rechaza $\Rightarrow$ M′ acepta)

por lo tanto, M′ siempre para y si $x \in \bar{L}$, M′ acepta y si $x \notin \bar{L}$, M′ rechaza. Y dado que es posible construir M′, $\bar{L}$ es un L.R.

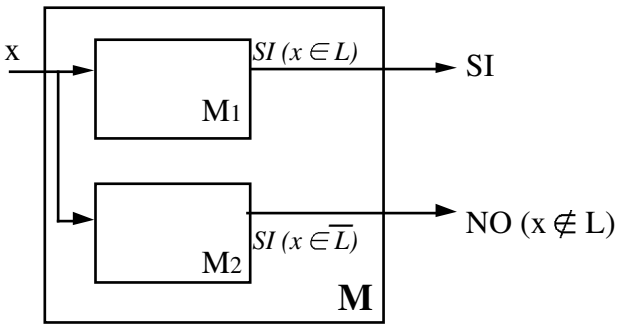
c.q.d.

Teorema 7.6 *Si un lenguaje L y su complementario $\bar{L}$ son L.R.E., entonces L y $\bar{L}$ son lenguajes recursivos.*

Demostración:

Sea L un L.R.E. $\Rightarrow \exists\ M_1 \mid$ L=L(M), es decir, si $x \in L$, M_1 acepta. Sea $\bar{L} = \Sigma^*$ - L, un L.R.E. $\Rightarrow \exists\ M_2 \mid \bar{L} = L(M)$, es decir, si $x \in \bar{L}$, M_2 acepta.

¿$\exists$ M | L = L(M) y M siempre para? Sí, construyendo la máquina M de la siguiente forma:



La construcción es correcta puesto que

- $x \in L \Rightarrow M_1$ acepta $\Rightarrow$ M acepta,
- $x \notin L \Rightarrow x \in \bar{L} \Rightarrow M_2$ acepta $\Rightarrow$ M rechaza.

por lo tanto, M siempre para y si $x \in L$, M acepta y si $x \notin L$, M rechaza. Por lo tanto, L es un lenguaje recursivo; y, por el teorema 7.5, $\bar{L}$ también es un lenguaje recursivo.

c.q.d.

Además, de los teoremas 7.5 y 7.6 se desprenden las siguientes relaciones: Dado el lenguaje L y su complementario $\bar{L}$, sólo se puede cumplir una de estas cuatro afirmaciones,

1. L y $\bar{L}$ son lenguajes recursivos,
2. L es recursivamente enumerable (no recursivo) y $\bar{L}$ NO es recursivamente enumerable,
3. L NO es recursivamente enumerable y $\bar{L}$ es recursivamente enumerable (no recursivo),
4. L y $\bar{L}$ NO son lenguajes recursivamente enumerables.

7.5. Problemas Propuestos.

1. El sistema MIU es un sistema sobre el alfabeto Σ = {M, I, U}. Las cadenas de este sistema se forman siguiendo las cuatro reglas siguientes, en las que *x* representa una subcadena genérica sobre Σ:

   a) Si *xI* es una cadena del sistema, también lo será *xIU*,

   b) Si *Mx* es una cadena del sistema, también lo será *Mxx*,

   c) En cualquier cadena, la subcadena *III* puede ser reemplazada por la subcadena *U*,

   d) En cualquier cadena, la subcadena *UU* puede ser reemplazada por la cadena λ.

   Se sabe que la cadena *MI* pertenece al sistema.

   a) Para cada una de las cadenas siguientes, *MIIU*, *MIIUU*, *MU*, indicar si se pueden derivar o no partiendo de la cadena *MI*. Justificar las respuestas.

   b) Relacionar la respuesta obtenida en el apartado anterior con algún resultado, bien conocido, estudiado en la asignatura.

2. Las siguientes afirmaciones, consideradas por separado, pueden ser, cada una de ellas, ciertas o falsas:

   - L es un lenguaje regular,
   - L es un lenguaje de contexto libre,
   - L es un lenguaje recursivo,
   - L es un lenguaje recursivamente enumerable.

   Sin embargo, si las 4 afirmaciones se refieren al mismo lenguaje, de las 2^4 combinaciones que se pueden formular a priori, no todas serán posibles. ¿Cuántas y cuáles serán válidas y por qué?

3. Sea L un lenguaje finito, ¿puede ser L un lenguaje recursivamente enumerable no recursivo?.

4. Sea M una MT, tal que *L(M) = L*, siendo L un lenguaje finito. ¿Puede ser que M no pare nunca ante entradas $x \notin L$?

5. Sea L un lenguaje recursivamente enumerable no recursivo, sobre el alfabeto Σ y sea M la MT que reconoce a L. Sea L_1,

   $$L_1 = \{x \in \Sigma^* \mid f(q_0, x) = q_i \; en \; menos \; de \; k \; pasos, \; q_i \in F\}$$

   ¿L_1 es recursivo o no?

6. Sea L un lenguaje recursivamente enumerable no recursivo sobre el alfabeto Σ y sea M la MT que reconoce a L. Sea L_k el lenguaje formado por aquellas cadenas de Σ^* tales que M no las ha reconocido en *k* pasos. ¿L_k es recursivo, recursivamente enumerable no recursivo o no recursivamente enumerable?

7. Sea L un lenguaje recursivamente enumerable no recursivo sobre el alfabeto Σ y sea M la MT que reconoce a L. Sea L_k el lenguaje formado por aquellas cadenas de L tales que M no las ha reconocido en *k* pasos. ¿L_k es recursivo, recursivamente enumerable no recursivo o no recursivamente enumerable?

8. Sea L_1 un lenguaje recursivamente enumerable no recursivo y sea L_2 un lenguaje recursivo. Indicar cómo serán (L.R., L.R.E. no recursivo, L. no R.E.) los lenguajes,

   a) $L_1 \cup L_2$,

   b) $L_1 \cap L_2$.

9. Se considera el lenguaje L, $L = L_1 - L_2$. Utilizando diagramas de bloques de Máquinas de Turing, deducir si L sería recursivo o recursivamente enumerable no recursivo en los supuestos,

   a) L_1 y L_2 son lenguajes recursivos.

   b) L_1 es recursivamente enumerable y L_2 es recursivo.

   c) L_1 es recursivo y L_2 es recursivamente enumerable.

10. Justificar si el conjunto de los lenguajes recursivamente enumerables es o no cerrado, con respecto a las siguientes propiedades:

    a) unión,

    b) intersección,

    c) complemento, y

    d) diferencia.

11. Se sabe que los lenguajes L_1 y L_2 son lenguajes recursivos. Se define la diferencia simétrica de dos lenguajes como

$$L = diffsim(L_1, L_2) = \{x \in L_1 \wedge x \notin L_2\} \cup \{x \notin L_1 \wedge x \in L_2\}$$

    a) Estudiar si es o no es una operación de clausura.

    b) ¿Qué ocurre si L_1 es un lenguaje recursivamente enumerable pero no recursivo? ¿y si L_2 es un lenguaje recursivamente enumerable no recursivo?

12. Se considera el lenguaje

$$L_{par} = \{x \in L \wedge \mid x \mid \ es\ par\}.$$

    Indicar si L_{par} es un lenguaje recursivo, recursivamente enumerable pero no recursivo o no recursivamente enumerable si:

    a) L es un lenguaje recursivo.

    b) L es un lenguaje recursivamente enumerable pero no recursivo.

13. Indicar para cada uno de los siguientes lenguajes si son recursivos, recursivamente enumerables no recursivos o no recursivamente enumerables, justificando el porqué:

a) Sabiendo que L_1 es un lenguaje recursivo,

$$L = \{x \in L_1 \mid x^{-1} \in L_1\}$$

b) Sabiendo que L_1 es un lenguaje recursivamente enumerable no recursivo,

$$L = \{x \in L_1 \mid x^{-1} \in L_1\}$$

c) Sabiendo que L_1 es un lenguaje recursivo y que L_2 es un lenguaje recursivamente enumerable no recursivo,

$$L = \{x \in \Sigma^* \mid x = x_1x_2 \wedge x_1 \in L_1 \wedge x_2 \in L_2\}$$

d) Sabiendo que L_1 es un lenguaje recursivamente enumerable no recursivo,

$$L = \{x \in \Sigma^* \mid x = yy^{-1} \wedge (y \in L_1 \Leftrightarrow y^{-1} \in L_1) \wedge x^{-1} \notin L\}$$

14. Considérese el lenguaje

$$L = \{0^I 10^J 10^{I+J} \mid \forall\, I, J \in \mathcal{N}\}.$$

a) Determinar si es un Lenguaje Recursivo o un Lenguaje Recursivamente Enumerable.

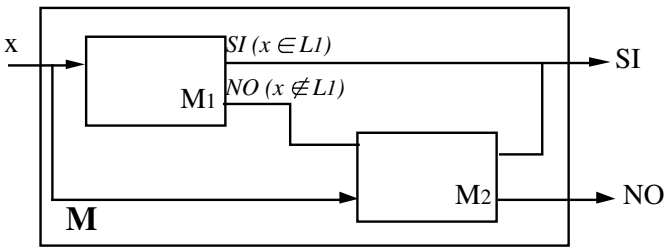
b) En cualquiera de los dos casos, L se puede generar. Dar las ideas básicas para la construcción de un generador, que, además, garantice que cualquier $x \in L$ se puede generar en un número finito de pasos.

15. *a*) Esquematizar mediante bloques la construcción de una MT capaz de **generar** el lenguaje $L = L_1 \cup L_2$, sabiendo que L_1 y L_2 son lenguajes recursivos. Explicar brevemente su funcionamiento.

b) Esquematizar mediante bloques la construcción de una MT capaz de **generar** el lenguaje $L = L_1 \cup L_2$, sabiendo que L_1 y L_2 son lenguajes recursivamente enumerables no recursivos. Explicar brevemente su funcionamiento.

c) Justificar, atendiendo al tipo y modo de funcionamiento de los generadores diseñados en los apartados (a) y (b), si los lenguajes correspondientes son recursivos o recursivamente enumerables no recursivos.

16. Dadas M_1 y M_2, indicar cómo habría que construir la MT M, que reconoce $L = L(M_1) \cup L(M_2)$:



17. Supóngase que los lenguajes L_1, L_2, ..., L_k forman una partición de Σ^*; es decir, la unión de todos ellos es igual a Σ^* y todos son disjuntos dos a dos,

$$\bigcup_{i=1}^{k} L_i = \Sigma^* \wedge L_i \cap L_j = \emptyset, \forall i, j \; i \neq j.$$

(nota: de ahí se desprende que para cualquier cadena *s* construida con símbolos de Σ, *s* sólo pertenecerá a un único L_i,

$$\forall s \in \Sigma^*, \; \exists! \; L_i \mid s \in L_i \; .)$$

Discutir la siguiente afirmación: *"Si cada L_i es un lenguaje recursivamente enumerable $\Rightarrow$ cada L_i es un lenguaje recursivo"*.