**Práctica 2 - Jerarquía de la Computabilidad**

**Ejercicio 1. Responder brevemente los siguientes incisos:**

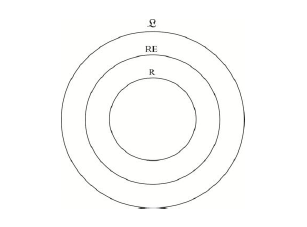
**1. ¿En qué difiere un lenguaje recursivo de un lenguaje recursivamente numerable no recursivo?**

Un lenguaje es recursivamente numerable si y sólo si existe una MT que lo reconoce.

Es decir, si es L el conjunto de todos los lenguajes (cada uno integrado por cadenas finitas de símbolos pertenecientes a un alfabeto universal Ʃ), sólo los lenguajes recursivamente numerables de L son reconocibles por una MT (por esto es que a los problemas de decisión asociados se los conoce como computables). La clase de los lenguajes recursivamente numerables se denomina RE (por recursively enumerable languages). El nombre se debe a que las cadenas de estos lenguajes se pueden enumerar. De esta manera, dado L ∈ RE, si M es una MT tal que L(M) = L, se cumple para toda cadena w de Ʃ\* que:

* Si w ∈ L, entonces M a partir de w se detiene en su estado qA.
* Si w ∉ L, entonces M a partir de w se detiene en su estado qR o no se detiene.

Se define que un lenguaje es recursivo si y sólo si existe una MT M que lo reconoce y que se detiene cualquiera sea su entrada (no queda en loop). La clase de los lenguajes recursivos se denomina R.

****

**¿En qué difiere un lenguaje recursivamente numerable de uno que no lo es?**

Los lenguajes no recursivamente numerables no son reconocidos por una MT. Un lenguaje que no está dentro de RE no es computable.

*Un lenguaje es recursivamente enumerable es un lenguaje formal para el cual existe una MT que acepta y se detiene con cualquier* [*cadena*](https://es.wikipedia.org/wiki/Cadena_de_caracteres) *que pertenece al lenguaje y para aquellos strings que no pertenecen al lenguaje puede rechazar o loopear. Es decir, que L(M)=L.. A estos lenguajes se les asocian los problemas computables. También es posible enumerar todas las cadenas que componen el lenguaje. Un lenguaje no es recursivamente numerable si no existe una MT que lo acepte. Esta frontera divide los lenguajes computables de los no computables.*

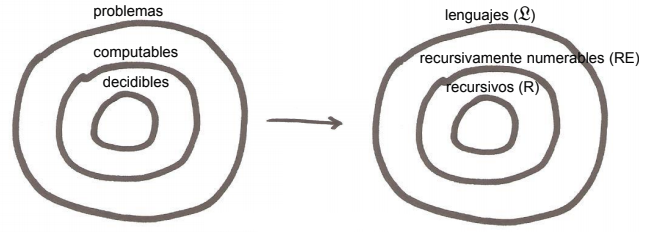
**2. Probar que R ⊆ RE ⊆** L**.**

Se prueba por definición. En R se encuentran todos aquellos lenguajes para los cuales se puede construir una MT que acepte el lenguaje y pare siempre. De esta forma R puede verse como un subconjunto particular de RE (recordemos que un lenguaje está en RE si se puede construir una MT que acepte el lenguaje).

Por su parte también sabemos que L representa el conjunto de todos los lenguajes que se pueden formar a partir de **Σ (** L **= P(** **Σ\* ) ).** Es decir que todo lenguaje que esté en RE pertenece a esta clase.

*Es importante saber que R ⊂ RE ⊂ L. no se puede probar por definición, sino que se tiene que hacer la demostración.*

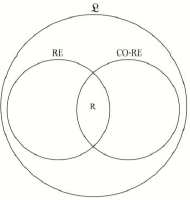
|  |  |
| --- | --- |
| A ⊆ B | Todo elemento de A está en B (puede ocurrir que A = B) |
| A ⊂ B | Todo elemento de A está en B ( asegura que ) |



**3. ¿Cuándo un lenguaje está en la clase CO-RE? ¿Puede un lenguaje estar al mismo tiempo en la clase RE y en la clase CO-RE? ¿Para todo lenguaje de la clase CO-RE existe una MT que lo acepta?**

Sea CO-RE la clase de los lenguajes complemento, con respecto a Ʃ\*, de los lenguajes

recursivamente numerables. Formalmente: CO-RE = {L | L ∈ L ^ L∁ ∈ RE}. Considerando CO-RE, la siguiente figura muestra una versión más detallada de la jerarquía de la computabilidad:



De la figura se desprende que un lenguaje L es recursivo si y sólo si tanto L como L∁ ∈ RE.

De la figura se desprende que un lenguaje L es recursivo si y sólo si tanto L como L^C son recursivamente numerables. Los lenguajes R son los que están al mismo tiempo en la clase RE y la clase CO-RE.

* R es un conjunto de lenguajes que son decidibles por MT. Dice "si" o "no" (para siempre).
* RE es un conjunto de lenguajes que ante una entrada que acepta dice que "si", pero ante una entrada que no acepta puede loopear o decir "no".
* Entonces, podemos afirmar que CO-RE es un conjunto de lenguajes que ante una entrada puede responder "No" (es decir, el input NO es aceptado), o puede loopear o decir "si".

**Nota: L^C = Complemento de L.**

### 4. Justificar por qué los lenguajes Ʃ\* y ∅ son recursivos.

Volviendo a la definición de lenguaje recursivo:

Un lenguaje L es recursivo (L ∈ R) si y sólo si existe una MT M que lo acepta y para siempre (también se puede decir directamente que lo decide). Por lo tanto, para toda cadena w de Ʃ\*:

* Si w ∈ L, entonces ML a partir de w para en su estado qA
* Si w ∉ L, entonces ML a partir de w para en su estado qR

Si construimos una MT que acepte el lenguaje Ʃ\*, entonces ante cualquier cadena que se le ingrese, siempre resolverá aceptando dicha cadena. De forma inversa, se podría asumir lo mismo para un MT que acepta el lenguaje ∅ (rechaza todo input).

**5. Si L ⊆ Ʃ\*, ¿se cumple que L ∈ R?**

Parte de esta respuesta puede justificarse con lo explicado en la primera parte de la respuesta al punto anterior. Se cumple L ∈ R si existe una MT que lo acepta y ésta se detiene siempre. Este hecho es totalmente independiente de que L ⊆ Ʃ\*.

* Ʃ\* es el conjunto de todas las cadenas finitas formadas a partir del alfabeto. Es infinito. Hay lenguajes que las máquinas construidas para aceptarlos no paran.
* Se cumple L ∈ R si existe una MT que lo acepta y ésta se detiene siempre. L es un problema decidible.
* Si L ⊆ Ʃ\*, sabiendo que Ʃ\* es recursivo, no podemos asegurar que L pertenece a R.

*La computabilidad de un lenguaje o problema no tiene nada que ver con su tamaño o densidad, es decir con la cantidad de cadenas que contiene, como sí veremos que sucede en la complejidad computacional temporal. La computabilidad se relaciona con la definibilidad, con el contorno de los conjuntos involucrados.*

Por ende, no importa el tamaño de los lenguajes, ni la cantidad de strings que cumplen con estos. Lo que importa es la estructura particular de esos strings.

**6. Justificar por qué un lenguaje finito es recursivo.**

Dado un lenguaje finito, sabemos que hay un número finito de cadenas y que estas cadenas son de longitud finita: hay infinitamente muchas cadenas posibles de, como máximo, n símbolos. Entonces, es fácil decidir si una palabra dada pertenece a este lenguaje o no.

Porque existe una MT capaz de reconocerlo y ésta, o bien lo acepta, o bien se detiene. Es decir, podría construirse una MT con dos cintas: una con el lenguaje input y otra con el lenguaje finito. La idea es ir comparando símbolo a símbolo y, si en algún momento no coincide, rechace; caso contrario, que acepte.

### **7. Justificar por qué si L1 ∈ CO-RE y L2 ∈ CO-RE, entonces (L1 ⋂ L2) ∈ CO-RE.**

Si L1 está en CO-RE, quiere decir que L1^C está en RE. Lo mismo con el complemento de L2.

Por propiedad de clausura respecto a la intersección, L1 ⋂ L2 está en RE. Entonces, podemos afirmar que el complemento de L1 ⋂ L2 está en CO-RE.

Dado que CO-RE es cerrado en cuanto a intersección, entonces si L1⋂L2 se da que (L1⋂L2)∈CO-RE. Esto se demuestra construyendo una MTU que en la que somete al input a la M1 y luego, si acepta, a la M2. Si acepta, la MTU acepta; si rechaza en M1 o en M2, la MTU rechaza.

**Ejercicio 2.**

**Sean L1 y L2 dos lenguajes recursivamente numerables de números naturales representados en notación unaria (por ejemplo, el número 5 se representa con 11111). Probar que también es recursivamente numerable el lenguaje L = {x | x es un número natural representado en notación unaria, y existen y, z, tales que y + z = x, con y ∈ L1, z ∈ L2}. Ayuda: la prueba es similar a la de la propiedad de clausura de la clase RE con respecto a la operación de concatenación.**

Al usar notación unaria para representar números, al realizar una suma no hacemos más que agregar más “1”s a la cadena. Por lo cual el problema se puede probar de manera idéntica a como probamos la propiedad de clausura de RE para la concatenación.

Por enunciado sabemos que existen los lenguajes L1 y L2. Ambos pertenecen a la clase RE. Además sabemos que existe el lenguaje L que de acuerdo a su definición puede expresarse como (L1 . L2). Se nos pide probar que L1 . L2 pertenece a RE.

Para probar esto usaremos una MT.

Idea General:

Se usarán 6 cintas. La primera cinta tendrá el input recibido. La segunda tendrá un contador para la cantidad de pasos máximos que la máquina considerará en cada iteración. La cintas 3 y 4 servirán como los contadores “i” y “k”. La cinta 5 “simulará” una MT (a la que llamaremos M1) que acepta L1 (Sabemos que existe porque L1 ∈ RE). La cinta 6 “simulará” de la misma forma una MT (a la que llamaremos M2) que acepta L2.

Máquina:

Pasos:

1. En la cinta 2 se inicializa el contador en 1
2. En la cinta 3 se inicializa el contador i con el valor 0
3. En la cinta 4, el contador k con el valor n (siendo n la longitud del input).
4. En la cinta 5 se copia el contenido de las primeras i celdas del input de la cinta 1.
5. En la cinta 6 se copian las últimas n.
6. Se simula “en paralelo” la ejecución de M1 y M2. En cada una se harán a lo sumo. Si M1 y M2 aceptan (llegan al estado Qa) entonces la máquina acepta.
7. Si i=n, en la cinta 2 incrementa en uno el valor, borra los contenidos de la cinta 3, 4, 5 y 6 y vuelve al paso 2.
8. Hace i= i + 1, borra el contenido de las cintas 5 y 6 y vuelve al paso 4.

Pruebas:

Prueba de que M se detiene en qA o loopea

* 1. w ∈ L→ con entrada w, M **se detiene** en qA luego de que se determina que M1 y M2  aceptan al recibir w.
  2. w ∉ L→ con entrada w, M **se** **queda en loop**.

Prueba de que L(M) = L

* 1. w ∈ L(M)↔con entrada w M se detiene en qA↔ con una parte de w M1 se detiene en qA y con el resto de w M2 se detiene en qA
  2. w ∉ L(M)↔con entrada w M lopea↔ con una parte de w M1 se detiene en qR o loopea y con el resto de w M2 se detiene en qR o loopea.

**Ejercicio 3.**

**Dada una MT M1 con Ʃ = {0, 1}:**

**1. Construir una MT M2 que determine si L(M1) tiene al menos una cadena.**

Se tiene que tener especial atención a que la MT M2 no se quede loopeando. Es fácil detectar cuando una máquina queda en este estado al ver que ya ha pasado por todos los estados posibles definidos.

Se entiende como “paso” a un cambio de estado (escribir algo, y hacer algún tipo de movimiento).

Para evitar caer en loop se podría ejecutar la maquina de a pasos. Esto quiere decir ejecutar primero la máquina con 1 paso, luego intentar con 2, y así sucesivamente. En la primera instancia (solo 1 paso) se detecta y la máquina M1 acepta una cadena de longitud 1 (acepta 0 o 1), si lo hace entonces ya cumple, en caso contrario se deberá intentar con cadenas de longitud 2. Este procedimiento debe repetirse hasta encontrar una cadena w de longitud k.

Con encontrar una única cadena que cumpla ya se puede parar.

**2. ¿Se puede construir además una MT M3 para determinar si L(M1) tiene a lo sumo una cadena? Justificar.**

Resulta imposible probar que solo 1 cumple, puesto a que el alfabeto que utiliza MT M1 genera un conjunto infinito de cadenas (nunca se podría parar de probar).

**Ayuda para la parte (1): Si L(M1) tiene al menos una cadena, entonces existe al menos una cadena w de unos y ceros, de tamaño n, tal que M1 a partir de w acepta en k pasos. Teniendo en cuenta esto, pensar cómo M2 podría simular M1 considerando todas las cadenas de unos y ceros hasta encontrar eventualmente una que M1 acepte (¡cuidándose de los casos en que M1 entre en loop!).**

**Ejercicio 4.**

**Considerando el Lema 2 estudiado en la Clase Teórica 2 (propiedad de clausura de la clase R con respecto a la operación de intersección):**

**1. Indicar cómo se implementaría copiar el input w en la cinta 2 de la MT M construida.**

mientras (no llegue a un B en la cinta 1)

escribo el simbolo que lee la cinta 1 en la cinta 2 y me muevo a la derecha en las 2

**2. Indicar cómo se implementaría borrar el contenido de la cinta 2 de M.**

mientras (no llegue a un B en la cinta 2)

escribo B en la cinta 2 y se mueve a la izquierda

escribir blanco y mover a la derecha para dejar el cabezal posicionado al inicio.

**3. Probar la correctitud de la construcción:**

**(a) M para siempre.**

**L3(M L3)= L3 Y L1(M1 L1) = L1 , L2(M2 L2) = L2 Y** L1 ⋂ L2 = L3

/\*nuestra\*/

**si w**  ∉ l1 entonces ML a partir de w para en su estado qr

si w ∈ l1 y ∈ l2 entonces ML a partir de w para en su estado qa

si w ∈ l1 y ∉ l2 entonces ML a partir de w para en su estado qr

**(b) L(M) = L1 ⋂ L2.**

**w** ∈ M si y solo si w e l1 y pertenece a l2,

w ∉ M si Y solo si w ∈ l1 y no pertenece a l2 o w no pertenece a l1.

/\*otra más\*/

La MT para siempre porque está compuesta por dos MT que paran siempre; se ejecuta el input en MT1, ésta dirá qR en cuyo caso ya es suficiente para que la MT pueda rechazar; y si dice qA será necesario ejecutar el mismo input en la MT2 que se comporta de manera similar; es decir, si MT2 dice qR MT también dirá qR; en otro caso dira qA y aceptará el lenguaje. Bajo ninguna circunstancia la máquina loopeara, ya que copiar y borrar el input terminarán cuando procesen los n elementos del input

### Probar las otras propiedades de clausura de R mencionadas.

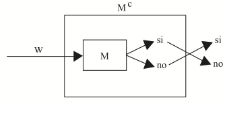
*La resolución se encuentra en el libro, entre las páginas 28 y 31 inclusive*.

Considerando las operaciones de complemento, intersección, unión y concatenación de lenguajes, se cumple que la clase R es cerrada con respecto a todas ellas.

#### *Probamos propiedad de clausura de R con respecto al complemento*:

Dado un lenguaje L, su lenguaje complemento es L^C = {w | w ∈ Ʃ\* ∧ w ∉ L}. Demostramos a continuación que si L ∈ R, entonces también L^C ∈ R.

* Idea general: Dado un lenguaje recursivo L, sea M una MT que lo acepta y se detiene siempre, es decir a partir de cualquier entrada. Se va a construir una MT M^C que acepta L^C y se detiene siempre, de la siguiente manera: dada una entrada w, si M se detiene en qA, entonces M^C se detiene en qR, y viceversa. La figura siguiente ilustra esta idea:



* Construcción de la MT M^C.

Si M = (Q, Ʃ, Γ, δ, q0, qA, qR), entonces M^C = (Q, Ʃ, Γ, δ’, q0, qA, qR), con δ y δ’ idénticas salvo en la aceptación y rechazo. Para todo estado q de Q, todo par de símbolos xi y xk de Γ, y todo movimiento d del conjunto {L, R, S}, se define:

1. Si δ(q, xi) = (qA, xk, d), entonces δ’(q, xi) = (qR, xk, d)
2. Si δ(q, xi) = (qR, xk, d), entonces δ’(q, xi) = (qA, xk, d)

* Prueba de que M^C se detiene siempre.

1. w ∈ L^C → w ∉ L → con entrada w, M se detiene en qR → con entrada w, M^C se detiene en qA.
2. w ∉ L^C → w ∈ L → con entrada w, M se detiene en qA → con entrada w, M^C se detiene en qR.

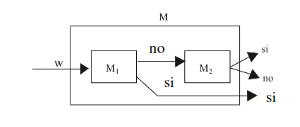
* Prueba de L(M^C) = L^C.

1. w ∈ L(M^C) ↔ con entrada w, M^C se detiene en qA ↔ con entrada w, M se detiene en qR ↔ w ∉ L ↔ w ∈ L^C.

Esta es una típica prueba por construcción de que un lenguaje L es recursivo. Se construye una MT M, y se prueba que M se detiene siempre y acepta L. La prueba por construcción de que un lenguaje L es recursivamente numerable, en cambio, requiere solamente la construcción de una MT M y la prueba de que M acepta L.

#### *Probamos propiedad de clausura de R con respecto a la unión*:

* Idea general: Sean M1 una MT que acepta L1 y se detiene siempre, y M2 una MT que acepta L2 y se detiene siempre, se va a construir una MT M que acepta L1 U L2 = L3 y se detiene siempre, con las siguientes características: M simula primero M1 y luego M2. Dada una entrada w si M1 se detiene en su estado qA entonces M se detiene en qA. En cambio, si M1 se detiene en qR entonces con la misma entrada w se simula en M M2 y si esta se detiene en qA entonces M se detiene en qA, caso contrario M para en qR.



* Construcción de la MT M:

M tiene 2 cintas. Dada una entrada w en la cinta 1, M:

1. Copia w en la cinta 2.
2. Simula M1 a partir de w en la cinta 2. Si M1 se detiene en su estado qA entonces M se detiene en su estado qA.
3. Borra el contenido de la cinta 2.
4. Copia w en la cinta 2.
5. Simula M2 a partir de w en la cinta. M se detendrá en el estado en que lo haga M2.

* Prueba de que M se detiene siempre.

1. w ∈ L3 → con entrada w, M se detiene en qA luego de que acepten M1 o M2
2. w ∉ L3 → con entrada w, M se detiene en qR luego de que rechaze M1 y rechace M2

* Prueba de que L(M) = L1 U L2 = L3

1. w ∈ L3 ↔ con entrada w M se detiene en qA ↔ con entrada w M1 se detiene en qA o M2 se detiene en qA

#### *Probamos propiedad de clausura de R con respecto a la concatenación*:

* Idea general: Sean M1 una MT que acepta L1 y se detiene siempre, y M2 una MT que acepta L2 y se detiene siempre. Se va a construir una MT M que acepta L1 . L2 y se detiene siempre, con las siguientes características. Dada una entrada w, con |w| = n, M simula M1 a partir de los primeros 0 símbolos de w, y M2 a partir de los últimos n símbolos de w, y si en ambos casos hay aceptación, entonces M acepta w. En caso contrario, M hace lo mismo pero ahora con el primer símbolo de w y los últimos n – 1 símbolos de w. Mientras no se detenga por aceptación, M repite el proceso con 2 y n – 2 símbolos de w, 3 y n – 3 símbolos, y así siguiendo hasta llegar a los n y 0 símbolos, en cuyo caso M rechaza w.
* Construcción de la MT M.

M tiene cinco cintas. A partir de una entrada w en su cinta 1, tal que |w| = n, hace:

1. Escribe el número 0 en la cinta 2. Sea i dicho número.
2. Escribe el número n en la cinta 3. Sea k dicho número.
3. Escribe los primeros i símbolos de w en la cinta 4.
4. Escribe los últimos k símbolos de w en la cinta 5.
5. Simula M1 en la cinta 4 a partir del contenido de dicha cinta, y simula M2 en la cinta 5 a partir del contenido de dicha cinta. Si ambas simulaciones se detienen en qA, entonces M se detiene en qA.
6. Si i = n, se detiene en qR.
7. Hace i := i + 1 en la cinta 2, k := k – 1 en la cinta 3, borra los contenidos de las cintas 4 y 5, y vuelve al paso 3.

* Prueba de que M se detiene siempre.

1. w ∈ L3 → con entrada w, M se detiene en qA luego de que acepten M1 y M2 cada una una porción de w
2. w ∉ L3 → con entrada w, M se detiene en qR luego de haber hecho todas las particiones posibles de w y que M1 y M2 sigan rechazando.

* Prueba de que L(M) = L1 . L2 = L3

1. w ∈ L3 ↔ con entrada w M se detiene en qA↔ con una porcion de w M1 se detiene en qA y con el resto de w M2 se detiene en qA

**Considerando el Lema 3 estudiado en la Clase Teórica 2 (propiedad de clausura de la clase RE con respecto a la operación de unión):**

**1. Indicar cómo se implementaría la suma de 1 al contador i en la MT M construida.**

Si utilizamos un contador unario, y el valor de i = n, entonces será necesario que desde la celda inicial nos desplazamos n lugares hacia la derecha utilizando algún carácter especial que sirva de corte, de cambio de estado. De esta forma incrementar el contador i consistiría en escribir n+1 caracteres en celdas sucesivas.

*/\*otra opción\*/*

*Se elige un carácter, por ejemplo el número 1, y se copian tantos caracteres como el número que quiero representar. Por ejemplo, si está en blanco, el contador es 0, si está en “1”, vale 1, si es “11” vale 2, si es “111” vale 3, y así sucesivamente.*

**2. Indicar cómo se implementaría ejecutar en M, i pasos de las MT M1 y M2.**

Se utiliza un contador el cual inicialmente tiene un valor igual a 0. De esta forma se logra tener diferentes iteraciones cuyo punto de finalización es cuando el contador alcanza el valor i. En cada una de estas iteraciones la máquina ejecutará como máximo tantos pasos como indique el contador. Cada iteración termina cuando la máquina acepta o cuando llega al número de pasos antes mencionados.

### 3. Probar la correctitud de la construcción: L(M) = L1 ⋃ L2.

L3(M L3) = L3 Y L1(M1 L1)=L1 , L2(M2 L2)= L2 L1 ⋃ L2 = L3

si w ∈ l1 y a l2 M se detiene en qa

si w ∈ l1 y ∉ l2 O w ∉ l1 y ∈ l2 M se detiene en qa

si w ∉ l1 y ∉ l2 M dice qr o loopea.

### 4. Probar las otras propiedades de clausura de RE mencionadas.

*La resolución se encuentra en el libro, entre las páginas 22 y 34 inclusive*.

Considerando las operaciones de intersección, unión y concatenación de lenguajes, se cumple que también la clase RE es cerrada con respecto a ellas. En cambio, a diferencia de la clase R, RE no es cerrada con respecto al complemento.

#### *Probamos propiedad de clausura de RE con respecto a la concatenación*:

Al igual que la clase R, RE es cerrada con respecto a la concatenación de lenguajes, es decir que si L1 ∈ RE y L2 ∈ RE, entonces también (L1 . L2) ∈ RE, lo que se prueba a continuación. Debe tenerse en cuenta que las MT consideradas pueden no detenerse en caso de rechazo.

* Idea general: Tal como se hizo con los lenguajes recursivos, se va a construir una MT M que reconozca L1 . L2 simulando M1 y M2 (las MT que reconocen L1 y L2, respectivamente), primero a partir de 0 y |w| símbolos de la entrada w, después a partir de 1 y |w – 1| símbolos, y así siguiendo hasta llegar a |w| y 0 símbolos, aceptando eventualmente. La diferencia con el caso de los lenguajes recursivos está en que ahora, teniendo en cuenta las posibles no detenciones de M1 y M2, M debe simularlas “en paralelo”. La MT M primero hace simulaciones de un paso de M1 y M2 con todas las posibles particiones de la entrada w (|w| + 1 posibilidades), luego hace simulaciones de a lo sumo dos pasos, y así siguiendo hasta eventualmente aceptar (éste es el caso en que al cabo de a lo sumo un determinado número de pasos, digamos k, M1 acepta los primeros i símbolos de w y M2 acepta los últimos |w| – i símbolos de w).
* Construcción de la MT M.

Sea la siguiente MT M con seis cintas, que a partir de una entrada w en su cinta 1, tal que |w| = n, hace:

1. Escribe el número 1 en la cinta 2. Sea h dicho número.
2. Escribe el número 0 en la cinta 3. Sea i dicho número.
3. Escribe el número n en la cinta 4. Sea k dicho número.
4. Escribe los primeros i símbolos de w en la cinta 5.
5. Escribe los últimos k símbolos de w en la cinta 6.
6. Simula a lo sumo h pasos de M1 en la cinta 5 a partir del contenido de dicha cinta, y simula a lo sumo h pasos de M2 en la cinta 6 a partir del contenido de dicha cinta. Si ambas simulaciones se detienen en qA, entonces M se detiene en qA.
7. Si i = n, hace h := h + 1 en la cinta 2, borra los contenidos de las cintas 3, 4, 5 y 6, y vuelve al paso 2.
8. Hace i := i + 1 en la cinta 3, k := k – 1 en la cinta 4, borra los contenidos de las cintas 5 y 6, y vuelve al paso 4.

* Prueba de que M se detiene en qA o loopea: (llamo L3 al resultado de L1 . L2)

1. w ∈ L3 → con entrada w, M se detiene en qA luego de h pasos en los que se determina acepta M1 al recibir una porcion de w y acepta M2 al recibir el resto de w.
2. w ∉ L3 → con entrada w, M se queda en loop mientras que M1 y M2 sigan recibiendo partes de w y no aceptando (no podemos asumir que se detienen y rechazan).

* Prueba de que L(M) = L1 . L2 = L3

1. w ∈ L3 ↔ con entrada w M se detiene en qA ↔ con una parte de w M1 se detiene en qA y con el resto de w M2 se detiene en qA.

#### *Probamos propiedad de clausura de RE con respecto a la intersección*:

* Idea general: Sea M1 una MT que acepta L1 y M2 una MT que acepta l2, se va a construir una MT M que reconozca L1 ⋂ L2 simulando M1 y M2 (las MT que reconocen L1 y L2, respectivamente). En este caso no hace falta la ejecución en paralelo, ya que si M1 no acepta L1, la máquina M tampoco aceptará (es decir, ambas deben aceptar para que M acepte). Tenemos entonces, si M1 loopea, M loopea. Si M1 ante w se detiene en su estado qA, entonces M simula M2 con w y si M2 también se detiene en su estado qA, entonces M acepta. Caso contrario, M loopea.
* Construcción de la MT M

M tiene dos cintas. Dada una entrada w en la cinta 1, M:

1. Copia w en la cinta 2.
2. Simula M1 a partir de w en la cinta 2. Si M1 loopea, entonces M loopea.
3. Borra el contenido de la cinta 2.
4. Copia w en la cinta 2.
5. Simula M2 a partir de w en la cinta. M se detiene si M2 acepta. Sino, puede loopear.

* Prueba de que M se detiene en qA o loopea: (llamo L3 al resultado de L1 U L2)

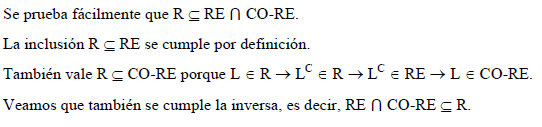
1. w ∈ L3 → con entrada w, M se detiene en qA si M1 se detuvo en qA y M2 se detuvo en qA.
2. w ∉ L3 → con entrada w, M se queda en loop mientras que M1 o M2 sigan recibiendo partes de w y no aceptando, o rechaza (se detiene en qR).

* Prueba de que L(M) = L1 U L2 = L3

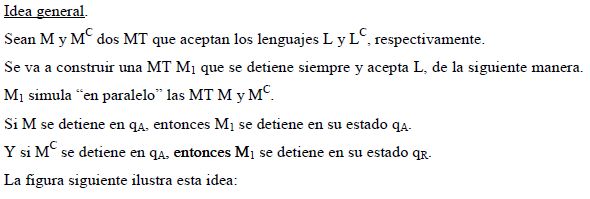
1. w ∈ L3 ↔ con entrada w M se detiene en qA ↔ con entrada w M1 se detiene en qA y M2 se detiene en qA.

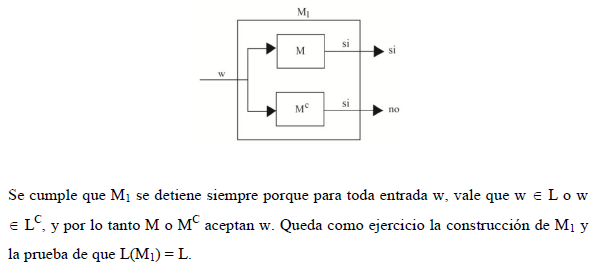
**Considerando el Lema 4 estudiado en la Clase Teórica 2 (R = RE ⋂ CO-RE):**

**1. Construir la MT M.**

****

Página 35 del libro en adelante.

****

****

Se cumple que M1 se detiene siempre porque para toda entrada w, vale que w ∈ L o w ∈ L^C, y por lo tanto M o M^C aceptan w.

### 2. Probar la correctitud de la construcción.

* Prueba de que M1 se detiene siempre.

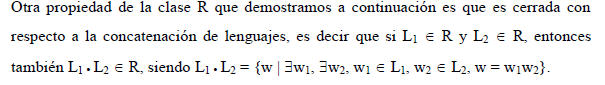
1. w ∈ L → con entrada w, M se detiene en qA luego de que acepte M.
2. w ∉ L → con entrada w, M se detiene en qR luego de que acepte M^c.

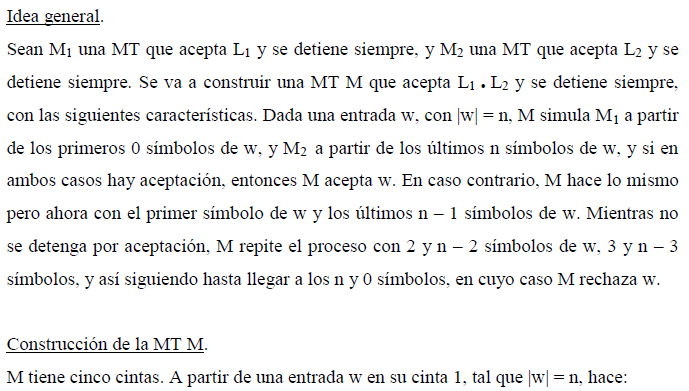
* Prueba de que L(M1) = L

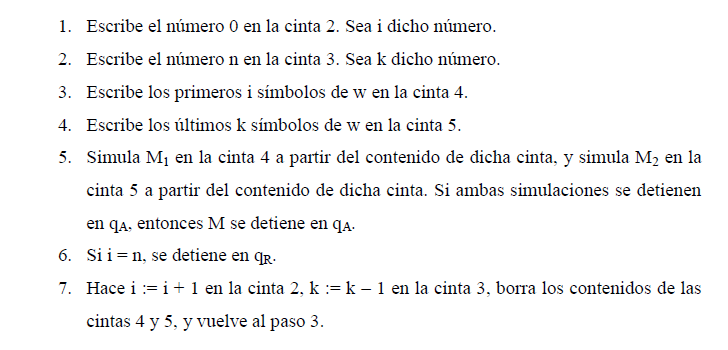
1. w ∈ L ↔ con entrada w M1 se detiene en qA ↔ con entrada w M se detiene en qA 0 M^C se detiene en qR.

**Verificar la correctitud de las construcciones de las MT efectuadas en la Clase Práctica 2, es decir las correspondientes a las pruebas de:**

**1. La propiedad de clausura de R con respecto a la operación de concatenación.**

****

****

****

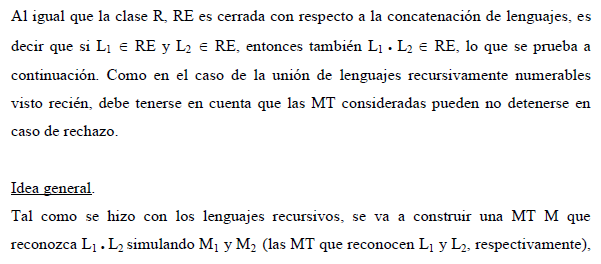
Prueba de que M se detiene siempre.

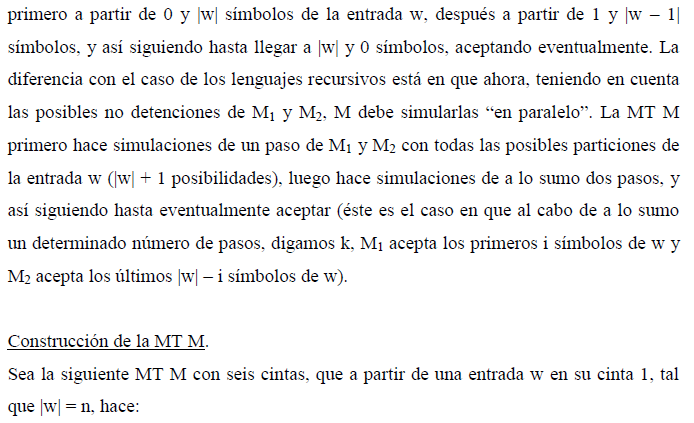
* 1. w ∈ L1 . L2 → con entrada w, M **se detiene** en qA luego de que acepten M1 y M2  cada una una porción de w
  2. w ∉ L1 . L2 → con entrada w, M **se** **detiene** en qR luego de haber hecho todas las particiones posibles de w y que M1 y M2 sigan rechazando.

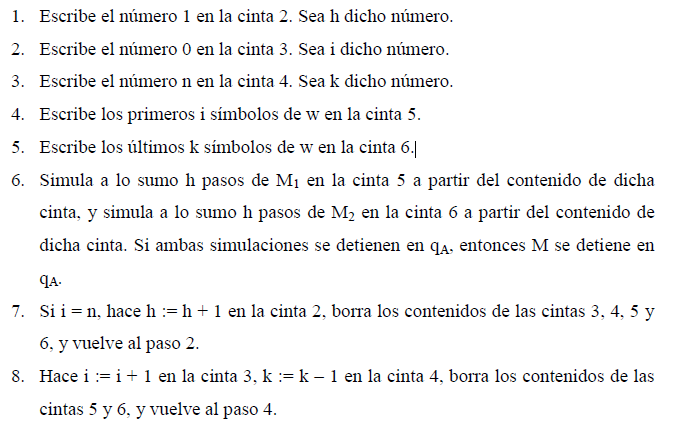
Prueba de que L(M) = L1.L2

* 1. w ∈ L1 . L2 ↔ con entrada w M se detiene en qA ↔ con una porción de w M1 se detiene en qA y con el resto de w M2 se detiene en qA

**2. La propiedad de clausura de RE con respecto a la operación de concatenación.**

****

****

****

Prueba de que M se detiene en qA o loopea:

* 1. w ∈ L1 . L2 → con entrada w, M **se detiene** en qA luego de que acepten M1 y M2  cada una una porción de w
  2. w ∉ L1 . L2 → con entrada w, M **se detiene en qR** **o se queda en loop** luego de haber hecho todas las particiones posibles de w y que M1 y M2 sigan rechazando.

Prueba de que L(M) = L1.L2

* 1. w ∈ L1 . L2 ↔ con entrada w M se detiene en qA ↔ con una porción de w M1 se detiene en qA y con el resto de w M2 se detiene en qA