Tema 1: Máquinas de Turing. Funciones y Lenguajes Calculables (Parte 2)

Serafín Moral

Universidad de Granada

Febrero, 2025

Contenido

- Máquinas de Turing
- Lenguajes recursivamente enumerables. Lenguajes recursivos
- Técnicas de construcción de Máquinas de Turing: memoria adicional, pistas múltiples, subrutinas
- Extensiones del concepto de MT
  - Pasos estáticos
  - Máquinas multicinta
  - Máquinas no deterministas
- Limitaciones de las MT
  - Máquinas con cintas semiilimitadas
- Problemas y Programas
  - Tipos de problemas
  - Programas
  - Problemas de decisión y Lenguajes
  - Palabras y Números
- Calculabilidad
  - El lenguaje diagonal
  - El lenguaje universal
  - Problemas indecidibles
  - Teorema de Rice
  - El problema de las correspondencias de Post

Problemas

Definición

Un problema PROBLEMA(x) ó PROBLEMA consta de:

- Un conjunto X de entradas. Un elemento $x \in X$ se llama una entrada.
- Un conjunto Y de solución. Un elemento $y \in Y$ se llama una solución.
- Una aplicación $F : X \to 2^Y$ que asigna a cada entrada $x \in X$ un conjunto $A \subseteq Y$ de soluciones posibles.

Problemas

Definición

Un problema PROBLEMA(x) ó PROBLEMA consta de:

- Un conjunto X de entradas. Un elemento $x \in X$ se llama una entrada.
- Un conjunto Y de solución. Un elemento $y \in Y$ se llama una solución.
- Una aplicación $F : X \to 2^Y$ que asigna a cada entrada $x \in X$ un conjunto $A \subseteq Y$ de soluciones posibles.

Ejemplo

Búsqueda de caminos en grafos dirigidos:

- Entradas X: conjunto formado por las tripletas (G, ns, nl), donde G es un grafo dirigido, ns es un nodo de salida, nl es un nodo de llegada.
- Conjunto Y: lista de nodos $(n_1, \ldots, n_k)$
- $F(G, ns, nl)$ es el conjunto de las listas de nodos $(n_1, \ldots, n_k)$ tales que $n_1 = ns, n_k = nl$ y todas las parejas (n_i, n_{i+1}) sean arcos de G.

Resolución de Problemas

- Los problemas se resuelven mediante algoritmos.
- ¿Qué es un algoritmo? En esta asignatura vamos a considerar dos conceptos que son equivalentes:
  - Un programa en Phyton (Ph) bien escrito sintácticamente y con, al menos, una función definida, la primera de las cuales es la función principal (más adecuados para una resolución efectiva de problemas).
  - Una Máquina de Turing (MT) que definiremos en el siguiente tema (más adecuadas para el razonamientos teórico-matemáticos).
- Un algoritmo ALG resuelve un problema $\text{PROBLEMA}(x)$ cuando el argumento de dicho algoritmo es un elemento $x \in X$ y $ALG(x)$ es un $y \in F(x)$ o dice "No hay Solución" si $F(x) = \emptyset$.

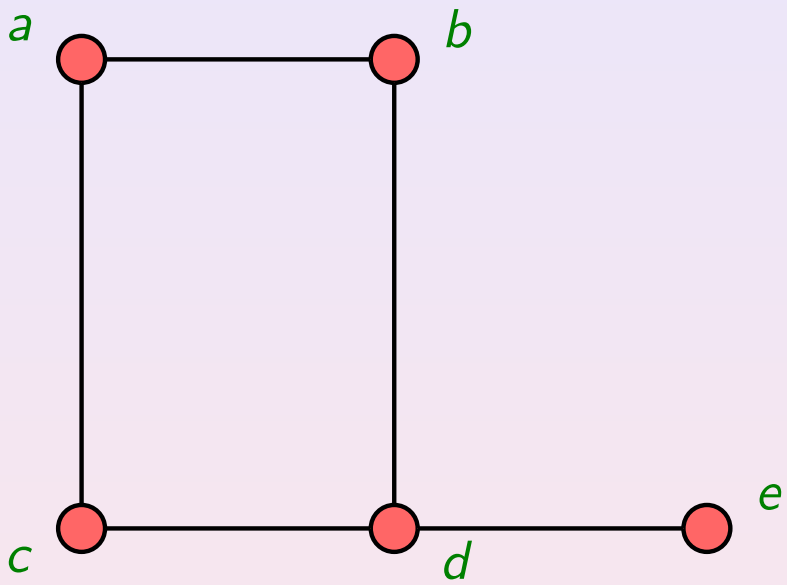
Problemas y su codificación

- Si queremos resolver un problema en un ordenador habrá que codificar las entradas y las soluciones en unos datos que pueda entender el ordenador.
- Nosotros vamos a suponer que las entradas y soluciones están siempre codificadas como palabras o cadenas de caracteres.
- Esto no supone ninguna restricción ya que cualquier tipo de entrada a un programa se puede representar mediante una cadena de caracteres, así como cualquier tipo de salida.
- También supondremos que hay sólo una palabra de entrada y una palabra de salida. Esto tampoco supone restricción ya que un conjunto de palabras se pueden codificar como una sola palabra en la que aparecen las palabras originales encadenadas con un separador para indicar el final de una y el comienzo de la siguiente.
- A partir de ahora, en términos generales hablaremos sólo sobre problemas en las que las entradas y las salidas son palabras sobre un alfabeto (Problema Computacional), aunque al hablar de un problema concreto hablaremos de grafos, números u otros elementos.

Problemas y su Codificación

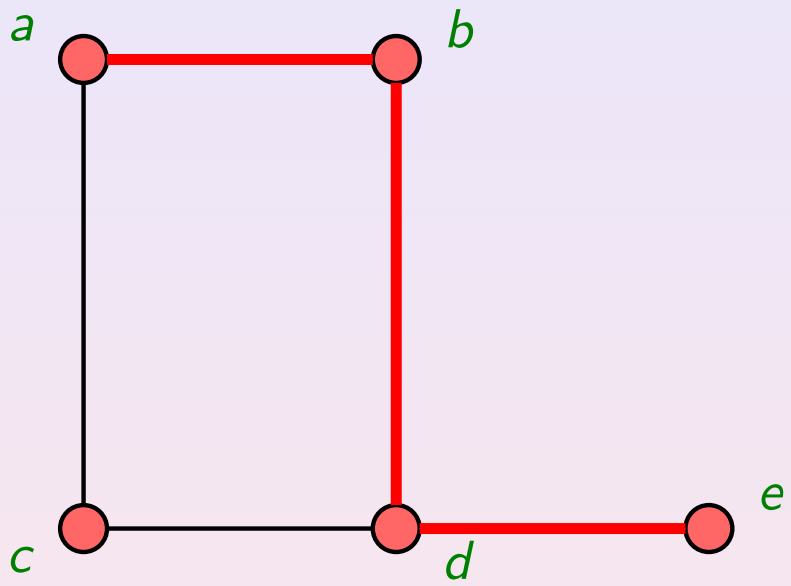
- Cualquier problema se transforma en un problema computacional con un sistema de codificación de las entradas y salidas.
- Puede haber palabras que no sean una codificación correcta de una entrada del problema original. En ese caso, supondremos que a esa entrada le corresponde una salida especial: 'NO'.
- En general, nuestro estudio teórico se hará sobre problemas computacionales, pero su resultado se aplicará también a problemas en general, ya que estos resultados no dependerán de la codificación elegida (siempre que ésta sea razonable).
- Si tenemos un problema y un sistema de codificación, para cada entrada x, la codificación de x, se denotará como $< x >$ (el elemento entre ángulos).

Codificando Grafos



Grafo: `a,b b,d c,d a,c d,e`

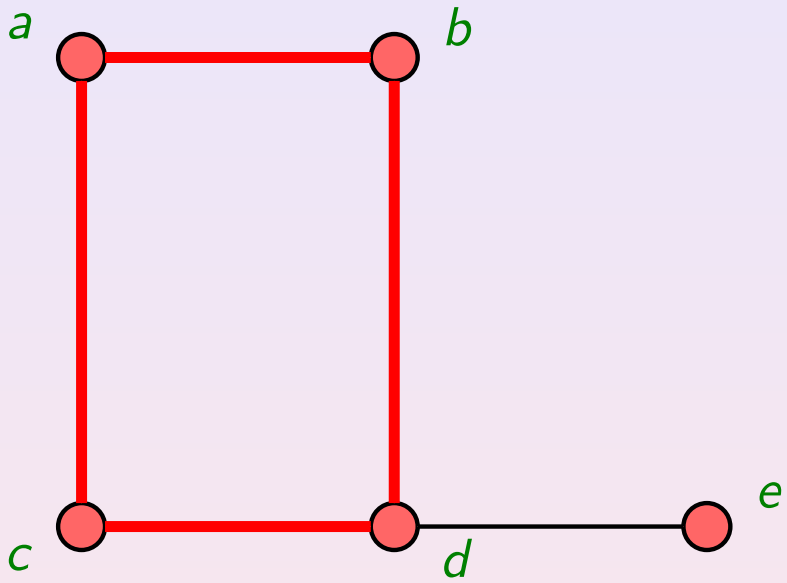
Codificando Grafos



Grafo: `a,b b,d c,d a,c d,e`

Camino: `a,b,d,e`

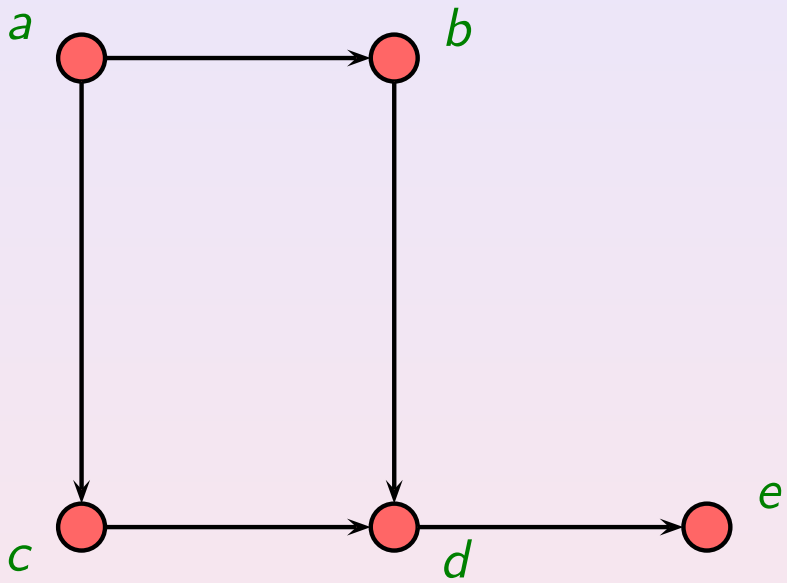
Codificando Grafos



Grafo: `a,b b,d c,d a,c d,e`

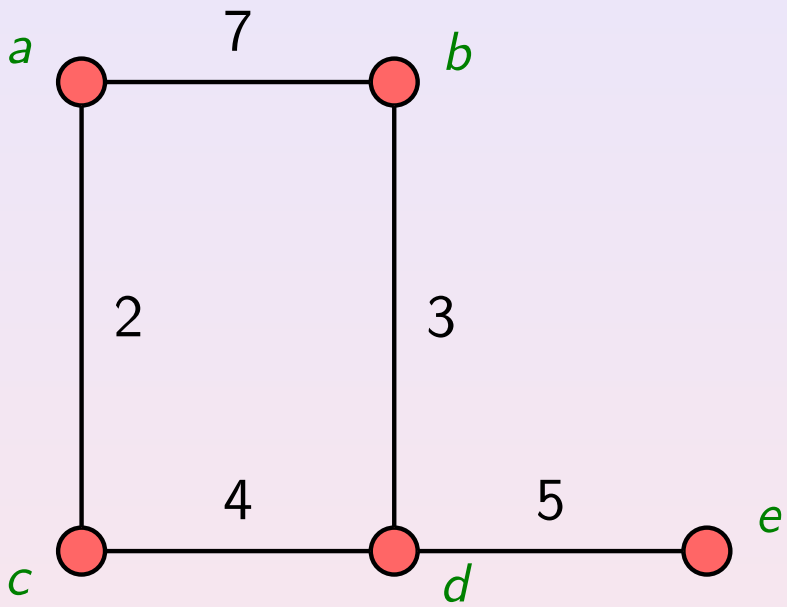
Ciclo: `a,b,d,c`

Codificando Grafos Dirigidos



Grafo dirigido: `a,b b,d c,d a,c d,e`

Codificando Grafos Con Pesos



Grafo con pesos: `a,b,7 b,d,3 c,d,4 a,c,2 d,e,5`

Tipos de Problemas

- Problemas de Búsqueda: Son los problemas genéricos cuando $F(x)$ puede ser vacío o contener varios elementos: Para una entrada x el problema consiste en encontrar una solución y que cumpla una relación con x cuando este exista y decir 'NO' cuando $F(x) = \emptyset$. Ejemplo: dado un grafo no dirigido encontrar un circuito hamiltoniano.
- Problemas de Decisión: Son aquellos en los que las soluciones son $Y = \{SI, NO\}$ y cada entrada x tiene una única solución. Ejemplo: dado un grafo determinar si tiene un circuito hamiltoniano.
- Problemas de Optimización: La solución optimiza (minimiza o maximiza) una función definida sobre un conjunto de soluciones factibles asociadas a la entrada. Ejemplo: el problema del viajante de comercio.
- Problemas de función: Cada entrada x tiene siembre una y sólo una solución: $F(x)$ tiene un solo elemento. Por ejemplo, dado un número n calcular su cuadrado n^2. A la solución que

Versión Decisión de los problemas

Los probemas de decisión son especialmente importantes ya que son simples y es fácil razonar sobre ellos y además cualquier otro problema tiene asociado un problema de decisión:

- Problemas de umbral para problemas de optimización: Los mismos datos de un problema de optimización más un umbral K y ahora se pregunta si existe una solución de valor mayor o menor que el valor K según sea un problema de máximo o mínimo. Ejemplo: dado un caso del problema del viajante de comercio y un valor K determinar si existe un circuito de coste menor o igual que K.
- Problemas de existencia para problemas de búsqueda: Dado un x, determinar si existe un y tal que sea una solución de x.
- Problemas de comprobación para problemas de función y búsqueda: Dado x y una posible solución y determinar si y es una solución de x.

Ejemplo

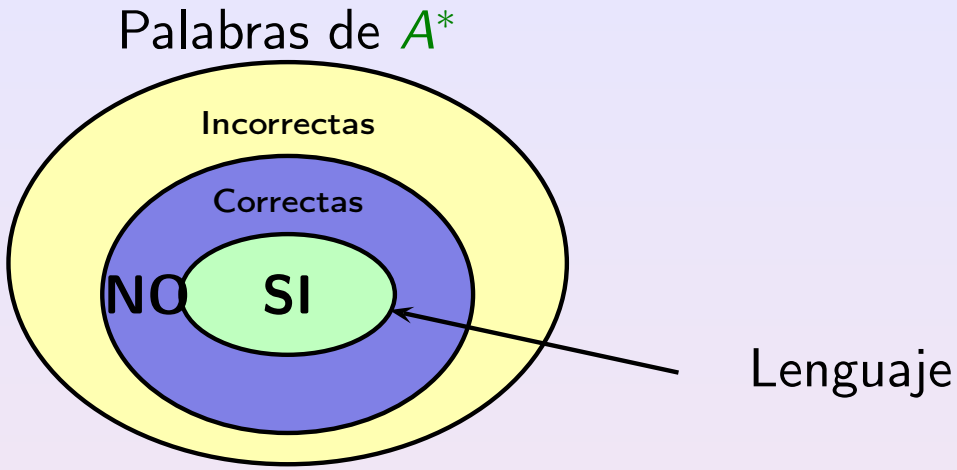
- **Camino mínimo:** Dado un grafo y dos nodos, encontrar un camino de longitud mínima entre estos nodos si este existe (problema de optimización).
- **Búsqueda de caminos:** Dado un grafo y dos nodos, encontrar un camino entre ambos nodos, en caso de que exista, decir 'NO' en caso contrario.
- **Existencia de Caminos:** Dado un grafo y dos nodos, determinar si existe un camino entre ellos (problema de existencia).
- **Umbral del camino mínimo:** Dado un grafo, dos nodos, y un umbral K determinar si existe un camino entre estos nodos de longitud menor o igual a K (problema de umbral).
- **Problema de comprobación:** Dado un grafo, dos nodos, y un camino, determinar si es un camino que une esos dos nodos (problema de comprobación).

Problemas de Decisión y Lenguajes

Un problema computacional de decisión en el que las entradas se codifican como palabras del alfabeto A con el lenguaje de las entradas que tienen respuesta 'SI':
$L(\text{PROBLEMA(x)}) = \{x \in A^* : \text{PROBLEMA}(x) =' SI'\}$.

- Es decir los casos con respuesta afirmativa.
- Un lenguaje L sobre un alfabeto A, siempre define también un problema de decisión: dada $x \in A^*$ determinar si $x \in L$.

Problemas y Lenguajes



- En muchos casos, la teoría se desarrolla hablando de lenguajes.
- Cuando comenzamos con un problema genérico de decisión y lo transformamos en un problema computacional mediante una codificación de las entradas, las palabras que no son una codificación correcta de un ejemplo del problema se meten en el mismo 'saco' que los casos en los que la respuesta es 'NO'.
- La identificación de las codificaciones correctas (palabras que corresponden realmente a un ejemplo del problema) se considera que no es importante desde el punto de vista computacional y no se tendrá en cuenta.

Problemas y Lenguajes

Un problema de decisión + Una codificación $\equiv$ Problema Computacional 'SI' 'NO'
$\equiv$ Problema Computacional 'SI' 'NO' $\equiv$ Lenguaje

- Las definiciones sobre lenguajes se pueden transformar en definiciones sobre los problemas asociados. Los problemas cuyos lenguajes son recursivos se llaman decidibles o calculable y los que no lo son indecidibles o no calculables. Esta terminología se aplica también a los lenguajes.
- Los problemas cuyos lenguajes son r.e. se llaman semidecidibles o parcialmente calculables.
- La identificación de las codificaciones correctas (palabras que corresponden realmente a un ejemplo del problema) se considera que no es importante desde el punto de vista computacional (en todos los ejemplos el reconocimiento de una entrada correcta se puede realizar de forma eficiente).

Problema Contrario

Dado un problema de decisión PROBLEMA, el problema contrario de PROBLEMA, es el problema $\mathrm{C}-\mathrm{PROBLEMA}$ que intercambia las salidas 'SI' y 'N0' o de forma más precisa $\mathrm{PROBLEMA}(x) =' SI' \Leftrightarrow \mathrm{C}-\mathrm{PROBLEMA}(x) =' NO'$.

Problemas contrarios y lenguajes

El lenguaje asociado al problema contrario de PROBLEMA es el lenguaje complementario del lenguaje del lenguaje asociado a PROBLEMA:

$$L(\mathrm{C}-\mathrm{PROBLEMA}) = \overline{L(\mathrm{PROBLEMA})}$$

Problema Contrario

Dado un problema de decisión PROBLEMA, el problema contrario de PROBLEMA, es el problema $C-PROBLEMA$ que intercambia las salidas 'SI' y 'N0' o de forma más precisa $PROBLEMA(x) =' SI' \Leftrightarrow C-PROBLEMA(x) =' NO'$.

Problemas contrarios y lenguajes

El lenguaje asociado al problema contrario de PROBLEMA es el lenguaje complementario del lenguaje del lenguaje asociado a PROBLEMA:

$$L(C-PROBLEMA) = \overline{L(PROBLEMA)}$$

Nota: Cuando hablamos de problemas genéricos, esto no es del todo exacto: ya que las codificaciones incorrectas estarán englobadas con el 'SI' en el lenguaje complementario o problema contrario, pero nosotros consideraremos que esta propiedad también se aplica a los problemas genéricos sin tener esto en cuenta: en realidad no importa como se traten las codificaciones incorrectas, ya que estas son siempre fáciles de detectar.

Ejemplo: Isomorfismo de subgrafos

- X: Datos
  Dos grafos $G_1 = (V_1, E_1)$ y $G_2 = (V_2, E_2)$
- Y: 'SI','NO' (Problema de Decisión)
  Relación: Respuesta 'SI' corresponde a ¿Contiene G_1 un subgrafo isomorfo a G_2?
  Es decir, existe un subconjunto $V' \subseteq V_1$ y un subconjunto de aristas $E' \subseteq E_1$ tal que $E' \subseteq V' \times V'$ y existe una aplicación biyectiva $f : V_2 \to V'$ de tal manera que se verifica

$$(u, v) \in E_2 \Leftrightarrow (f(u), f(v)) \in E'$$

Palabras y Números

Supongamos un alfabeto $A = \{a_1, \ldots, a_n\}$. Podemos establecer una correspondencia biyectiva entre las palabras sobre este alfabeto y los números naturales. Supongamos $w = a_{i_k} \ldots, a_{i_1}$, entonces el número de w que denotaremos como $N(w)$ es $\sum_{j=1}^{k} i_j . n^{j-1}$, siendo $N(\varepsilon) = 0$.

Ejemplo

Si $A = \{0,1,2\}$, $N(\varepsilon) = 0, N(0) = 1, N(1) = 2, N(2) = 3, N(202) = 3 + 1 \times 3 + 3 \times 3^2$

Dado un alfabeto $A = \{a_1, \ldots, a_n\}$, Si m es un número natural, siempre se puede encontrar una cadena, que denotaremos como $C(m)$ o como w_m cuya codificación sea m. Esto se puede conseguir de la siguiente forma,

- Si $m = 0$, $C(m) = \varepsilon$
- Si $m > 0$, sea

$$i = \begin{cases} R(m,n) & \text{si } n \text{ no divide a } m \\ n & \text{si } n \text{ divide a } m \end{cases}$$

$$p = \begin{cases} [m/n] & \text{si } n \text{ no divide a } m \\ [m/n] - 1 & \text{si } n \text{ divide a } m \end{cases}$$

donde $R(m,n)$ es el resto de la división entera de m entre n y $[m/n]$ es la división entera de m entre n.
Entonces $C(m) = C(p)a_i$.

Distintos alfabetos

Si tenemos dos alfabetos A y B podemos establecer una aplicación biyectiva entre las palabras de ambos alfabetos:

- Sea N_A la aplicación que codifica las palabras de A como números naturales: $N_A : A^* \to \mathbb{N}$.
- Sea C_B la aplicación que transforma números naturales en palabras sobre B: $C_B : \mathbb{N} \to B^*$.
- La composición $C_B \circ N_A$ es una aplicación biyectiva de A^* en B^* (primero se calcula el código numérico de una palabra de B y entonces la palabra del alfabeto de A que corresponde a ese código.
- Esta función es calculable en una MT.

Distintos alfabetos

Si tenemos dos alfabetos A y B podemos establecer una aplicación biyectiva entre las palabras de ambos alfabetos:

- Sea N_A la aplicación que codifica las palabras de A como números naturales: $N_A : A^* \to \mathbb{N}$.
- Sea C_B la aplicación que transforma números naturales en palabras sobre B: $C_B : \mathbb{N} \to B^*$.
- La composición $C_B \circ N_A$ es una aplicación biyectiva de A^* en B^* (primero se calcula el código numérico de una palabra de B y entonces la palabra del alfabeto de A que corresponde a ese código.
- Esta función es calculable en una MT.

Es suficiente trabajar sobre un sólo alfabeto. Usualmente, lo haremos sobre el alfabeto $\{0, 1\}$ (para desarrollar la teoría) aunque lo haremos sobre alfabetos más amplios (ASCII) para un caso práctico.

Orden Total en las Palabras de un Alfabeto

- Si $A = \{a_1, \ldots, a_n\}$ nosotros siempre vamos a considerar el siguiente orden total en sus palabras:
  - $u_1 \leq u_2$ si y solo si se da una de las siguientes condiciones:
    1. $u_1 = u_2$
    2. $|u_1| < |u_2|$
    3. $|u_1| = |u_2|$ y u_1 precede a u_2 en orden alfabético, teniendo en cuenta que $a_1 < a_2 < \cdots < a_n$. Es decir si $u_1 = a_{r_1} \ldots a_{r_i} a_{s_1} \ldots a_{s_m}$ y $u_2 = a_{r_1} \ldots a_{r_i} a_{l_1} \ldots a_{l_m}$ y $s_1 < l_1$ (el primer símbolo en el que son distintas las palabras es menor en u_1 que en u_2.
- Esto es equivalente a $u_1 \leq u_2$ si y solo si $N(u_1) \leq N(u_2)$
- El orden alfabético como tal no es operativo ya que no podemos establecer un ciclo infinito que recorra todas las palabras: hay infinitas palabras que empiezan por a_1 antes de las palabras que empiezan por a_2 y nunca se llegaría a ellas.

Codificando Máquinas de Turing

Se le puede asignar a cada MT sobre un alfabeto $\{0,1\}$ una cadena y un número natural. Para ello, suponemos

- Los estados son $\{q_1,\ldots,q_k\}$. El estado inicial es q_1 y hay un único estado final q_2 (esto siempre se puede conseguir).
- Los símbolos de B son $\{a_1,a_2,\ldots,a_m\}$ donde a_1 es 0, a_2 es 1 y a_3 es el símbolo blanco.
- Al movimiento izquierda le asignamos un 1 y al de la derecha un 2. Este número será $u(M)$.

La codificación de la MT se realiza de la siguiente forma:

- Cada transición $\delta(q_i,a_j)=(q_k,a_l,M)$ se codifica como $0^i10^j10^k10^l10^{u(M)}$.
- Todas las transiciones se van añadiendo a la codificación separadas por 11.

Una vez calculada la cadena $w=<M>$, podemos calcular su número $N(w)$ con el alfabeto $\{0,1\}$, según el procedimiento que hemos visto para asignar números a palabras. Este número también se denotará como $N(M)$.
Cada número natural corresponderá a una MT, o corresponderá a una cadena sin sentido alguno. Sea $T(n)$ la MT correspondiente al número n o *'Nula'* (que rechaza todas las palabras) si no hay MT asociada al número n. También denotaremos como $T(w)$ la MT cuyo código es w: $<M>=w$.

Codificando Máquinas de Turing y Entradas

- Para codificar una MT M y una palabra w sobre el alfabeto $\{0,1\}$, podemos codificar M como hemos visto y después añadir 111 seguido de w, dando lugar a la cadena $< M, w >$

El lenguaje de diagonalización

Vamos a definir un lenguaje L_d sobre $\{0,1\}$ que no es r.e. Este lenguaje se conoce como lenguaje de diagonalización.

Lenguaje de Diagonalización

Sea $w \in \{0,1\}^*$, $w \in L_d$ si y solo si la MT cuya codificación es w ($T(w)$) no acepta w. Se interpreta que si w no es un MT correcta, entonces representa una MT con un sólo estado y ninguna transición (siempre rechaza).

Si $w_0, w_1, w_2, \ldots$ son todas las palabras de $\{0,1\}^*$ ordenadas, entonces

Máquinas \ Palabras	w_0	w_1	w_2	w_3	$\ldots$
w_0	0	0	0	0	$\ldots$
w_1	1	0	1	0	$\ldots$
w_2	1	1	0	0	$\ldots$
w_3	0	1	0	1	$\ldots$
.	.	.	.	.	$\ldots$
.	.	.	.	.	$\ldots$

$$L_d = \{ < M > \mid M \text{ no acepta } < M > \text{ como entrada } \}$$

El lenguaje de diagonalización

Vamos a definir un lenguaje L_d sobre $\{0,1\}$ que no es r.e. Este lenguaje se conoce como lenguaje de diagonalización.

Lenguaje de Diagonalización

Sea $w \in \{0,1\}^*$, $w \in L_d$ si y solo si la MT cuya codificación es w ($T(w)$) no acepta w. Se interpreta que si w no es un MT correcta, entonces representa una MT con un sólo estado y ninguna transición (siempre rechaza).

Si $w_0, w_1, w_2, \ldots$ son todas las palabras de $\{0,1\}^*$ ordenadas, entonces

Máquinas \ Palabras	w_0	w_1	w_2	w_3	$\ldots$
w_0	0	0	0	0	$\ldots$
w_1	1	0	1	0	$\ldots$
w_2	1	1	0	0	$\ldots$
w_3	0	1	0	1	$\ldots$
.	.	.	.	.	$\ldots$
.	.	.	.	.	$\ldots$
L_d	1	1	1	0	$\ldots$

$$L_d = \{< M > \mid M \text{ no acepta } < M > \text{ como entrada } \}$$

L_d no es r.e.

Teorema

L_d no es r.e.

Supongamos una MT M_d que acepta L_d. Dicha MT estará definida sobre el alfabeto $\{0,1\}$. Dicha máquina acepta palabras w tales que la máquina de Turing con codificación w no acepta w.
Sea $< M_d >$ la codificación de la MT M_d.

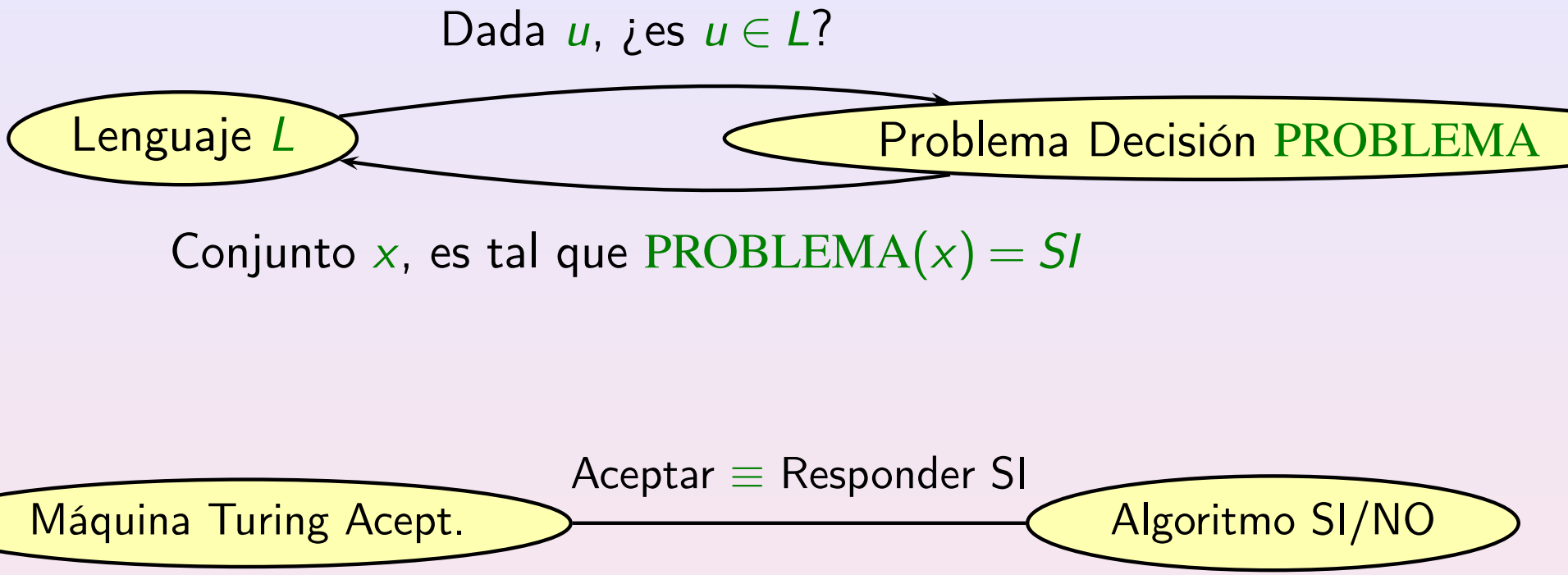
- Si $< M_d > \in L_d$ entonces M_d acepta $< M_d >$ (ya que M_d acepta L_d), como consecuencia y por la definición de L_d, $< M_d > \notin L_d$.
- Si $< M_d > \notin L_d$ entonces M_d no acepta $< M_d >$ (ya que M_d acepta L_d) y, por la definición de L_d, $< M_d > \in L_d$.

Con lo que la existencia de una MT M_d que acepte L_d nos lleva a una contradicción.

Problema de Diagonalización

A la versión problema de decisión del lenguaje de diagonalización le llamaremos Problema de Diagonalización y lo notaremos como DIAGONAL(M).
Consiste en determinar si dada una máquina de Turing M, entonces M no acepta cuando se lee a ella misma.

Recordatorio: Problemas y Lenguajes



Recordatorio: Problemas y Lenguajes

- Lenguaje recursivamente enumerable $\equiv$ Problema semidecidible
  - Existe una MT que acepta las palabras del lenguaje. Para las palabras del lenguaje la MT puede rechazar o ciclar.
  - Existe un algoritmo que responde correctamente las entradas cuya salida es 'SI', para las entradas de 'NO' el algoritmo puede decir 'NO' o ciclar
- Lenguaje recursivo $\equiv$ Problema decidible
  - Existe una MT que acepta las palabras del lenguaje y rechaza las palabras que no son del lenguaje (nunca cicla).
  - Existe un algoritmo que responde correctamente las entradas cuya salida es 'SI' y aquellas cuya salida es 'NO' (nunca cicla).

Complementarios de los lenguajes recursivos y r.e.

Si $L \subseteq A^*$, el complementario de L respecto a A^* se denotará como $\overline{L}$.

Teorema

Si L es un lenguaje recursivo, entonces $\overline{L}$ también lo es.

Demostración

Si L es recursivo, entonces es aceptado por una MT que siempre para. Se construye $\overline{M}$ con las siguientes características:

- Los estados de aceptación de M se convierten en estados de rechazo de $\overline{M}$ donde no se realizan nuevas transiciones.
- $\overline{M}$ tiene un nuevo estado r que es de aceptación y que no tiene transiciones definidas.
- Para cada estado p de M que no sea de aceptación y para cada símbolo de la cinta $a \in B$ para el que no haya definida una transición, se añade $\delta(p, a) = (r, a, D)$.

Está claro que $\overline{M}$ acepta el lenguaje $\overline{L}$ y siempre termina, ya que M lo hace.

Complementarios de los lenguajes recursivos y r.e.

Teorema

Si L y $\overline{L}$ son ambos r.e., entonces L es recursivo.

Demostración

Sea M_1 la MT que acepta L y M_2 la MT que acepte $\overline{L}$. Vamos a construir una MT M con dos cintas: en una funciona como M_1 y en la otra como M_2.
M es una máquina que funciona como M_1 y M_2 a la vez (como en el autómata producto).
El conjunto de estados de M es el producto $Q_1 \times Q_2$ donde Q_1 es el conjunto de estados de M_1 y Q_2 el de M_2. En cada paso, se pasa al estado que correspone según M_1 y al estado que corresponde según M_2.
La MT termina cuando una de las dos máquinas termina.
Los estados de aceptación de M son las parejas, en las que el primer elemento es un estado de aceptación de M_1.
Está claro que M acepta L y siempre termina, ya que toda palabra $u \in A^*$ está en L o en $\overline{L}$. En el primer caso, M_1 termina y en el segundo lo hace M_2. Por lo tanto M siempre termina.

El lenguaje L_d no es r.e. Su complementario podría ser r.e., pero nunca recursivo. De hecho, es r.e.

El Lenguaje Universal

Definición

El lenguaje universal L_u es el conjunto de todas v las cadenas del alfabeto $\{0,1\}$ que codifican parejas (M,w) (es decir $v =< M,w >$) tales que la MT M acepta la cadena w, donde $w \in \{0,1\}^*$ y el alfabeto de entrada de M es $\{0,1\}^*$.

Problema Universal

A la versión problema de decisión del lenguaje universal le llamaremos Problema Universal y lo notaremos como UNIVERSAL(M.w).
Consiste en determinar si dada una máquina de Turing M y una entrada w, entonces M acepta w.

Teorema

El lenguaje L_u es r.e.

Demostración

La idea básica es construir una MT M_u que lea la codificación $< M, w >$ simule la MT M sobre la entrada w y termine cuando termina M, aceptando si lo hace M. Esta MT se llama Máquina de Turing Universal. Está claro que si hace la simulación, aceptará cuando M acepta w. M_u contiene varias cintas:

- En la primera contiene la codificación de M y w
- En la segunda contiene lo que sería la cinta de M para la entrada w. Un símbolo $a_i \in B$ se representa como 0^i y los distintos símbolos se separan por un 1.
- En la tercera cinta representa el estado de M. El estado q_i se representa mediante 0^i
- La cuarta cinta se utiliza para cálculos auxiliares

Demostración (cont.)

1. Primero M_u examina M y w para asegurarse de que la entrada es correcta.
2. Inicializa la segunda cinta con w. El 0 se codifica como 01. El 1 se codifica como 001. Cada vez que introduzca un blanco lo tendrá que hacer como 0001. Todo de acuerdo con los códigos vistos.
3. Inicializa la tercera cinta con 0 que corresponde con el estado inicial (suponemos que es q_1).
4. Simula los movimientos. Tendrá que localizar en la primera cinta $0^i10^j10^k10^l10^m$ donde q_i es el estado en el que se encuentra, 0^j1 es lo que se ve en la segunda cinta. Si no lo encuentra para. Entonces debe de ejecutar el movimiento correspondientes:
   1. Cambiar el la cinta 3, el estado a 0^k
   2. Sustituir 0^j en la cinta 2 por 0^l
   3. Hacer el movimiento en la cinta 2, según sea m ($m = 1$ a la izquierda, $m = 2$ a la derecha).
5. Si M pasa a un estado de aceptación (el estado q_2), entonces M_u para y acepta.

Indecibilidad del Lenguaje Universal

Teorema

El lenguaje universal L_u no es recursivo.

Demostración

Por reducción al absurdo.

- Si L_u fuese recursivo, entonces $\overline{L_u}$ también sería recursivo.
- Si M es una MT que aceptase $\overline{L_u}$, construiríamos la siguiente MT, M':
  - Si M' lee w entonces, convierte w en $w111w$ (como la codificación de (w, w)).
  - Entonces pasa a funcionar como M, aceptando si M acepta.
  - M' acepta w si y solo sí, M acepta $w111w$. Es decir $w111w \notin L_u$. Esto es la MT cuya codificación es w no acepta la palabra w. Esto es equivalente a que $w \in L_d$.
- Hemos construido una MT que acepta L_d, en contra de lo que sabemos: L_d no es r.e.

Demostración en términos de problemas y programas

Teorema

El problema universal UNIVERSAL(M,w) no es decidible.

Demostración

Por reducción al absurdo:

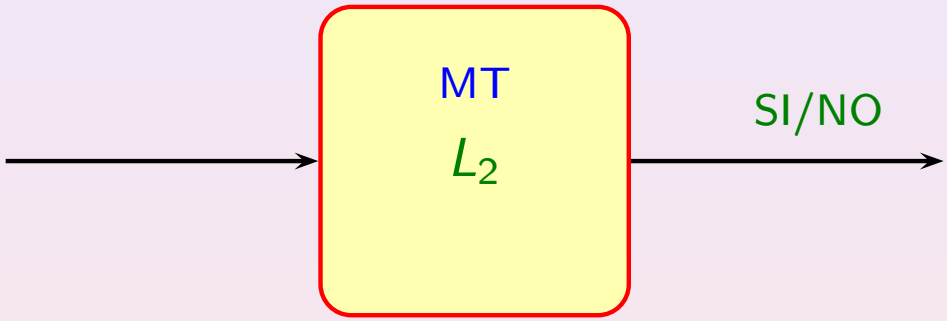
- Si UNIVERSAL(M, w) fuese decidible, entonces existiría un programa PROGRAMA(M, w) que siempre termina y lo resuelve.
- Construyamos ahora el siguiente programa:

```
PROGRAMAD(M)
        SAL ← PROGRAMA(M,<M>)
        Si SAL=='NO'
            Return 'SI'
        En otro Caso
            Return 'NO'
```

- Pero este programa tiene como entrada una MT M y responde 'SI' cuando esta máquina no acepta su propia codificación, luego implicaría que el problema de diagonalización es decidible, lo que sabemos que no es cierto.

Reducción (en términos de lenguajes)

Si $L_1 \subseteq A^*$ y $L_2 \subseteq B^*$ son lenguajes, el lenguaje L_1 se reduce al lenguaje L_2 si existe un algoritmo M (una MT) que siempre para y calcula una función $f : A^* \to B^*$ tal que para toda entrada $w \in A^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.

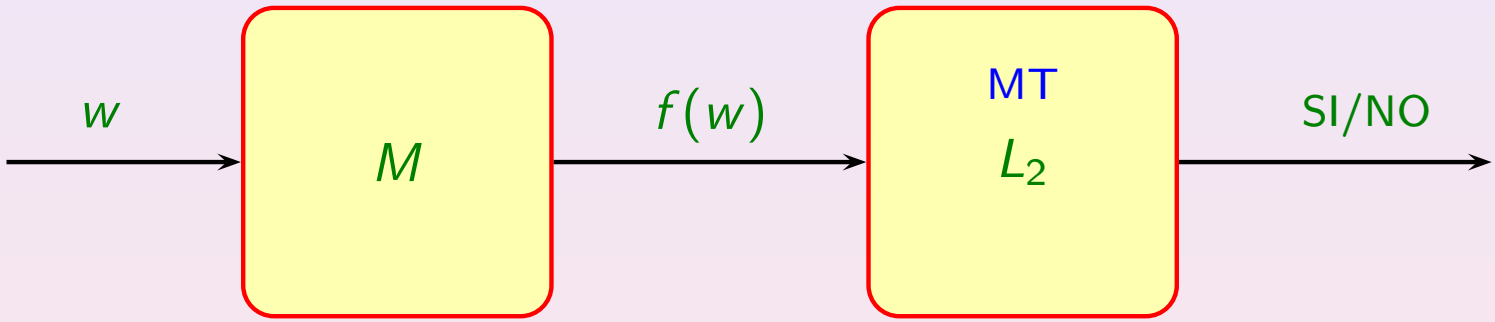


Teorema

Si L_1 se reduce a L_2, entonces si L_1 no es recursivo, tampoco lo es L_2 y si L_1 no es r.e., tampoco lo es L_2.

Reducción (en términos de lenguajes)

Si $L_1 \subseteq A^*$ y $L_2 \subseteq B^*$ son lenguajes, el lenguaje L_1 se reduce al lenguaje L_2 si existe un algoritmo M (una MT) que siempre para y calcula una función $f : A^* \to B^*$ tal que para toda entrada $w \in A^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.

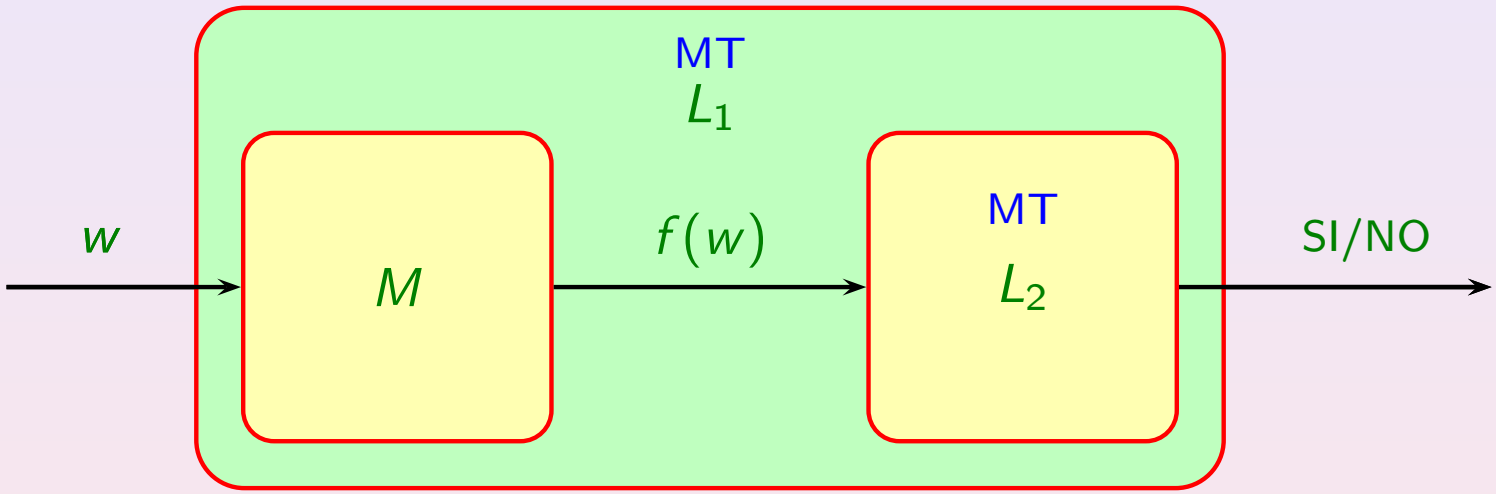


Teorema

Si L_1 se reduce a L_2, entonces si L_1 no es recursivo, tampoco lo es L_2 y si L_1 no es r.e., tampoco lo es L_2.

Reducción (en términos de lenguajes)

Si $L_1 \subseteq A^*$ y $L_2 \subseteq B^*$ son lenguajes, el lenguaje L_1 se reduce al lenguaje L_2 si existe un algoritmo M (una MT) que siempre para y calcula una función $f : A^* \to B^*$ tal que para toda entrada $w \in A^*$, $w \in L_1 \Leftrightarrow f(w) \in L_2$.



Teorema

Si L_1 se reduce a L_2, entonces si L_1 no es recursivo, tampoco lo es L_2 y si L_1 no es r.e., tampoco lo es L_2.

Reducción (en términos de problemas)

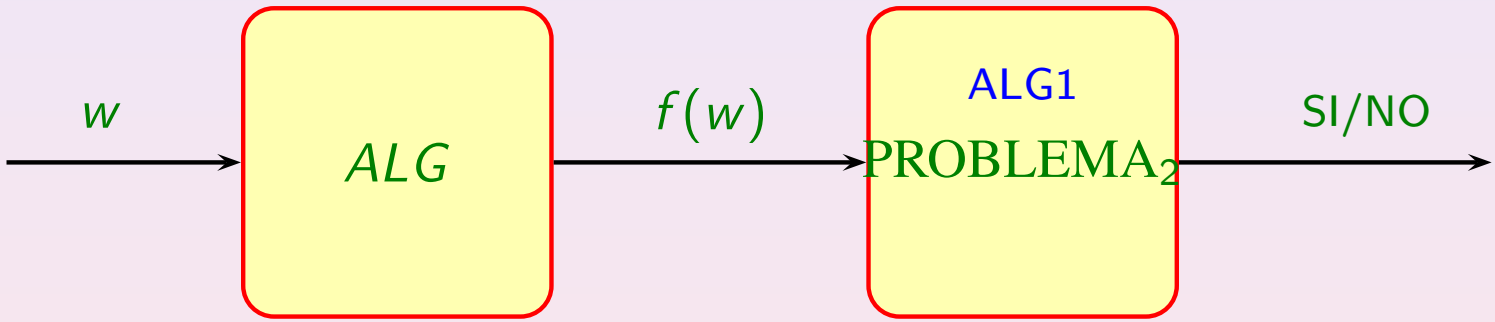
Sean PROBLEMA_1 y PROBLEMA_2 problemas de decisión, entonces decimos que PROBLEMA_1 se reduce a PROBLEMA_2 si existe un algoritmo $ALG(w)$ que siempre para y calcula una función $f(w)$ tal que para toda entrada w a PROBLEMA_1, tenemos que PROBLEMA_2 produce la misma respuesta para la entrada $f(w) = ALG(w)$.

Una reducción de PROBLEMA_1 a PROBLEMA_2 es una algoritmo $ALG(x)$ de las entrandas de PROBLEMA_1 a las entrandas de PROBLEMA_2 de tal forma que

$$\text{PROBLEMA}_1(x) = \text{PROBLEMA}_2(ALG(x))$$

Reducción (en términos de problemas)

Sean PROBLEMA_1 y PROBLEMA_2 problemas de decisión, entonces decimos que PROBLEMA_1 se reduce a PROBLEMA_2 si existe un algoritmo $ALG(w)$ que siempre para y calcula una función $f(w)$ tal que para toda entrada w a PROBLEMA_1, tenemos que PROBLEMA_2 produce la misma respuesta para la entrada $f(w) = ALG(w)$.

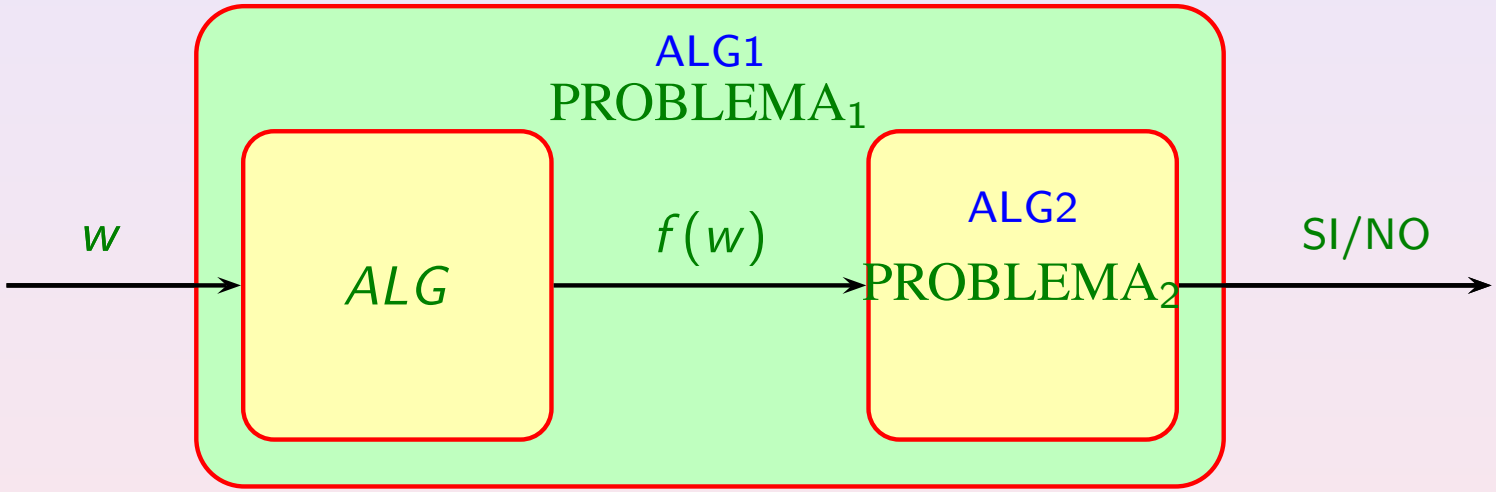


Una reducción de PROBLEMA_1 a PROBLEMA_2 es una algoritmo $ALG(x)$ de las entrandas de PROBLEMA_1 a las entrandas de PROBLEMA_2 de tal forma que

$$\text{PROBLEMA}_1(x) = \text{PROBLEMA}_2(ALG(x))$$

Reducción (en términos de problemas)

Sean PROBLEMA_1 y PROBLEMA_2 problemas de decisión, entonces decimos que PROBLEMA_1 se reduce a PROBLEMA_2 si existe un algoritmo $ALG(w)$ que siempre para y calcula una función $f(w)$ tal que para toda entrada w a PROBLEMA_1, tenemos que PROBLEMA_2 produce la misma respuesta para la entrada $f(w) = ALG(w)$.



Una reducción de PROBLEMA_1 a PROBLEMA_2 es una algoritmo $ALG(x)$ de las entrandas de PROBLEMA_1 a las entrandas de PROBLEMA_2 de tal forma que

$$\text{PROBLEMA}_1(x) = \text{PROBLEMA}_2(ALG(x))$$

Reducción (en términos de problemas)

Teorema

Si PROBLEMA_1 se reduce a PROBLEMA_2, entonces si PROBLEMA_1 es indecidible, también lo es PROBLEMA_2 y si PROBLEMA_1 no es semidecidible, tampoco lo es PROBLEMA_2.

Sea $ALG(w)$ el algoritmo de la reducción de PROBLEMA_1 a PROBLEMA_2. Supongamos que $ALG2(x)$ es un algoritmo que hace a PROBLEMA_2 semidecidible (decidible), entonces el algoritmo:

```
ALG1(x)
     w = ALG(x)
     Return (ALG2(w))
```

hará al problema PROBLEMA_1 semidecidible (decidible).
Por lo tanto si PROBLEMA_1 no es semidecidible (decidible), tampoco lo puede ser PROBLEMA_2.

MTs que aceptan el lenguaje vacío

Definimos los siguientes lenguajes sobre el alfabeto $\{0,1\}$:

- L_e conjunto de palabras $< M >$ tales que M es una MT sobre $\{0,1\}$ que no acepta ninguna palabra ($L(M) = \emptyset$).
- L_{ne} conjunto de palabras $< M >$ tales que M es una MT sobre $\{0,1\}$ que acepta alguna palabra ($L(M) = \emptyset$)($L(M) \neq \emptyset$).

Versión Problema

- VACIO(M) es la versión de problema de L_e: dada una MT M, determinar si acepta el lenguaje vacío.
- C-VACIO(M) es la versión de problema de L_{ne}: dada una MT M, determinar si acepta un lenguaje distinto del vacío.

Semidecidividad de C-VACIO(M)

Teorema

L_{ne} es r.e.

Demostración

Hay una MT no determinista M que acepta L_{ne}:

- M lee w que codifica una MT N, entonces con el algoritmo no determinista genera una palabra de entrada u.
- Después pone a simular N sobre u con la MT universal M_u.
- Si M_u para y acepta, entonces M acepta w.

C-VACIO(M) es semidecidible.

MTs que aceptan el lenguaje vacío

Teorema

L_{ne} no es recursivo (C-VACIO(M) no es decidible)

Demostración

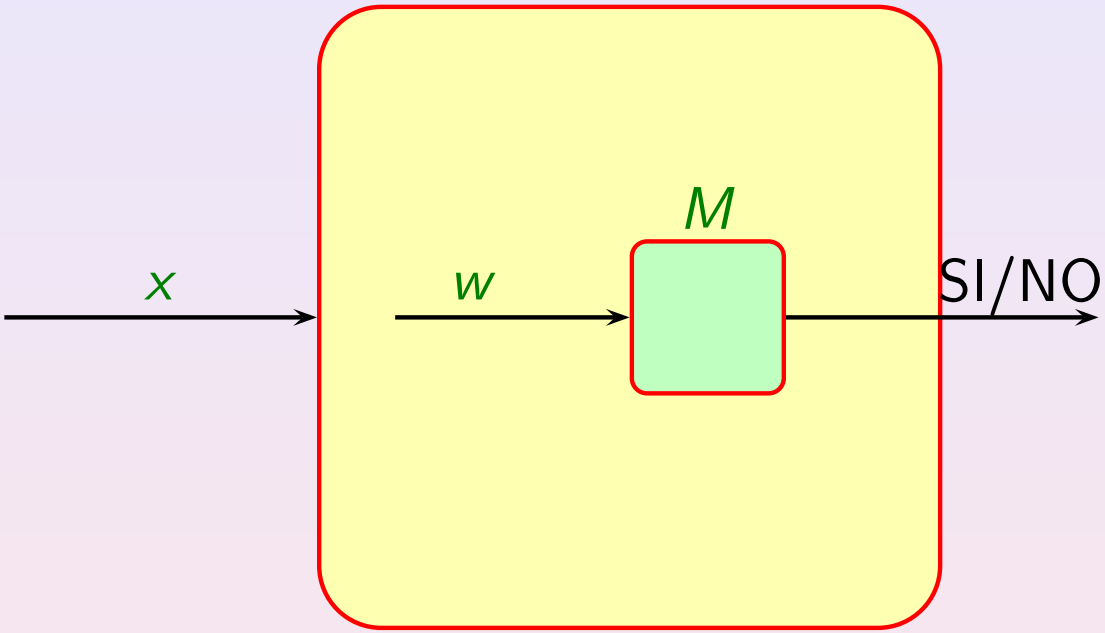
Vamos a demostrarlo usando los problemas asociados. Vamos a reducir UNIVERSAL(M,w) and C-VACIO(M'). Eso consiste en un algoritmo ALG(M,w) que calcula una MT M' de tal manera que UNIVERSAL(M,w) tenga la misma solución que C-VACIO(M'). Este funciona de la siguiente forma:

- Supongamos una entrada (M, w) vamos a construir una MT M' que funciona de la siguiente forma.
- M' ignora su entrada x y coloca en la cinta de entrada w. Si la longitud de w es n, esto se puede hacer con n estados. Cada estado q_i escribe el símbolo i de (M, w) y se mueve a la derecha. Después pasaría a un nuevo estado en el que borra lo que quede de x.
- M' se mueve a la izquierda hasta el primer símbolo de w.
- M' pasa al estado inicial de M con w y funciona como M para w.
- La salida de M' es la misma que la de M para w.

Está claro que M acepta w si y solo si M' acepta alguna palabra. De hecho $L(M') = A^*$ si M acepta w y $L(M') = \emptyset$ si M no acepta w.

Demostración Gráfica y Consecuencia

Dada una entrada (M, w) la reducción construye la siguiente MT:



Teorema

L_e no es r.e. (si lo fuese, entonces L_{ne} sería recursivo)

Propiedades de los r.e.

Propiedad lenguaje r.e. $\leftrightarrow$ Propiedad de los lenguajes de las MTs
Una propiedad de los lenguajes r.e. se identifica con el problema de saber si el lenguaje de una MT verifica esa propiedad.
Es una problema de decisión del tipo: Dada una MT M, ¿verifica el lenguaje $L(M)$ la propiedad P?

Ejemplo

Propiedades
Saber si el lenguaje de una MT es finito.

No es una propiedad de los r.e. saber si una MT tiene más de 10 estados.

Definición

Una propiedad de los lenguajes r.e. se dice trivial si para toda MT su lenguaje aceptado no verifica la propiedad o para toda MT su lenguaje aceptado siempre verifica la propiedad.

Teorema de Rice

Toda propiedad no trivial sobre los lenguajes r.e. es indecidible

Demostración

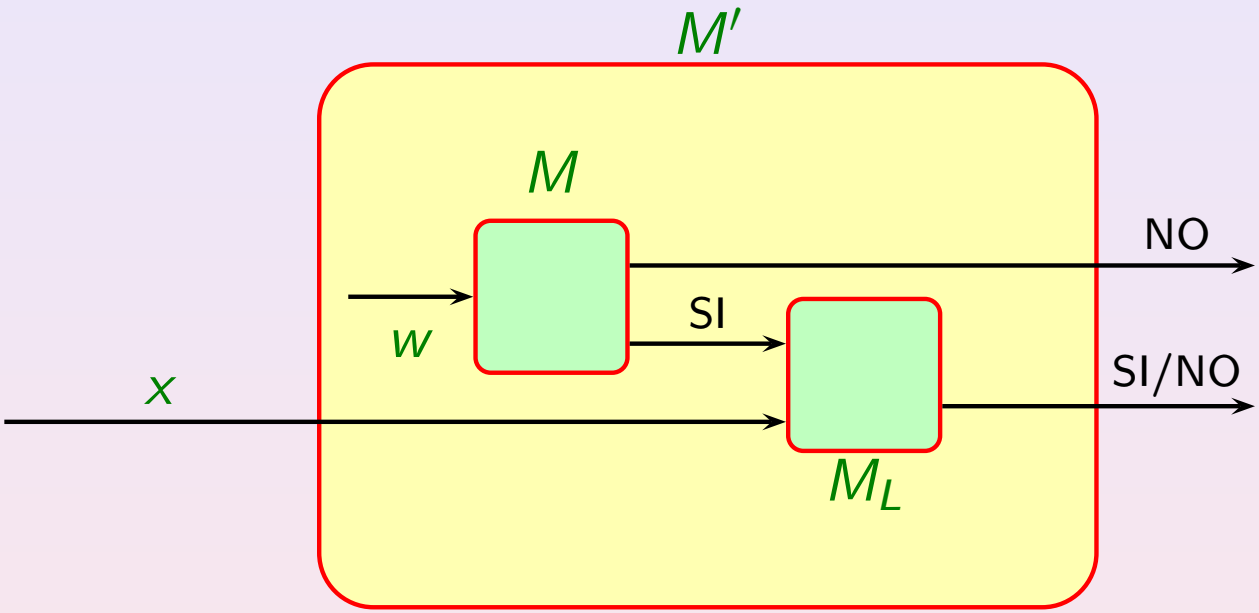
Llamemos NOTRIVIAL(M) a dicha propiedad no trivial y supongamos una propiedad no trivial y supongamos que el lenguaje vacío aceptado por la MT M_e no verifica la propiedad que otro lenguaje L aceptado por la MT M_L si verifica la propiedad.
Vamos a reducir el problema UNIVERSAL(M,w) a esta propiedad. Supongamos (M, w) una MT y su entrada construimos M' de la siguiente forma (se supone que tiene una entrada x en la primera cinta):

- M' tiene dos cintas. En la segunda coloca a w y empieza trabajando sobre esta cinta con las mismas transiciones de M. Si termina y no acepta, entonces M' no acepta.
- Si M termina y acepta con w, entonces empieza a funcionar como M_L sobre la entrada x en la primera cinta y tiene la misma salida que M_L.

Está claro que si M acepta w, entonces el lenguaje de M' es L y verifica la propiedad. Si M no acepta w, entonces el lenguaje de M' es vacío y no verifica la propiedad. Con esto se acaba la reducción y la propiedad no es decidible.
Si el lenguaje vacío acepta la propiedad se haría la misma transformación con la propiedad complementaria, demostrando que no es decidible y, por lo tanto, la propiedad original tampoco lo es.

Teorema de Rice. Gráfica de la Reducción

Dada una entrada para el problema universal (M, w) se construye la siguiente MT M'



Está claro UNIVERSAL(M,w) tiene respuesta 'SI', entonces M acepta w, entonces M' acepta L y la respuesta de NOTRIVIAL(M') es también 'SI'.
Si UNIVERSAL(M,w) tiene respuesta 'NO', entonces M no acepta w, y esta máquina M' acepta $\emptyset$ y la respuesta de NOTRIVIAL(M') es también 'NO'.
Como UNIVERSAL(M,w) no es decidible, NOTRIVIAL(M') tampoco lo es.

Otros Problemas Indecidibles

El Problema de las Correspondencias de Post (POST(A1,A2))

Tenemos un alfabeto de referencia A, y dos listas con la misma longitud $B_1 = w_1, \ldots, w_k$, $B_2 = u_1, \ldots, u_k$ de palabras sobre A. El problema es determinar si existe una secuencia no vacía de enteros $i_1, \ldots, i_m$ tales que $w_{i_1} \ldots w_{i_m} = u_{i_1} \ldots u_{i_m}$.

Podemos pensar en cada pareja (w_i, u_i) como un bloque de construcción:

w_i
u_i

.

La especificación del problema nos da un conjunto de bloques disponibles. Por ejemplo:

abb
a

b
abb

a
bb

Otros Problemas Indecidibles

El Problema de las Correspondencias de Post (POST(A1,A2))

Tenemos un alfabeto de referencia A, y dos listas con la misma longitud $B_1 = w_1, \ldots, w_k$, $B_2 = u_1, \ldots, u_k$ de palabras sobre A. El problema es determinar si existe una secuencia no vacía de enteros $i_1, \ldots, i_m$ tales que $w_{i_1} \ldots w_{i_m} = u_{i_1} \ldots u_{i_m}$.

Podemos pensar en cada pareja (w_i, u_i) como un bloque de construcción:

w_i
u_i

.

La especificación del problema nos da un conjunto de bloques disponibles. Por ejemplo:

abb	b	a
a	abb	bb

En este caso, la respuesta es afirmativa. Secuencia: $1,3,1,1,3,2,2$

abb	a	abb	abb	a	b	b	$\rightarrow$	$abbaabbabbabb$
a	bb	a	a	bb	abb	abb		$abbaabbabbabb$

Si visitáis la página web: `https://people.ksp.sk/~kuko/pcp/` podéis practicar con numerosos ejemplos de este puzzle.

El Problema de las Correspondencias de Post Modificado (POSTM(A1,A2))

PCP Modificado

Tenemos un alfabeto de referencia A, y dos listas con la misma longitud $B_1 = w_1, \ldots, w_k$, $B_2 = u_1, \ldots, u_k$ de palabras sobre A tales que $u_i, w_j \neq \varepsilon$ y un entero i. El problema es determinar si existe una secuencia no vacía de enteros $i_1, \ldots, i_m$ tales que $i_1 = i$ y $w_{i_1} \ldots w_{i_m} = u_{i_1} \ldots u_{i_m}$.

La única diferencia es que ahora nos dicen el bloque por el que necesariamente hay que comenzar y que las palabras son no vacías. Vamos a suponer siempre que el bloque por el que hay que empezar es el primero de la lista de los bloques (sólo hay que reordenar los bloques para que el que nos indican como bloque por el que hay que empezar aparezca primero.

Ejemplo

Supongamos el ejemplo:

a	ab	bba
baa	aa	bb

Como problema de las correspondencias de Post, tiene solución: 3,2,3,1 (respuesta afirmativa)
Efectivamente sale:

bba	ab	bba	a
bb	aa	bb	baa

Con lo que se lee $bbaabbbaa$ en la parte superior e inferior.
Sin embargo, como PCP modificado no tiene solución (respuesta negativa) ya que en ese caso, una solución tiene que empezar por el primer bloque, y así la palabra superior empieza por baa y la inferior por bb y ya, pongamos lo que pongamos después, las palabras resultantes no pueden ser iguales.

Semidecidibilidad del probl. de las correspondencias de Post

Se supone que la entrada al problema contiene k bloques

Un algoritmo no-determinista que acepta los casos positivos es:

Secuencia = []
Fin = Falso
Mientras No Fin
    Elige i entre 1 y k
    Añade i a Secuencia
    Elige Fin = Verdadero o Fin = False
Comprueba que eligiendo los bloques en el orden Secuencia es una solución del PCP
Si es solución responde 'SI', en caso contrario responde 'NO'

El algoritmo resuelve el problema, ya que si la respuesta es afirmativa puede responder de forma afirmativa y si es negativa NUNCA responderá de forma afirmativa.

El Problema de las Correspondencias de Post Modificado

Teorema

El problema de las correspondencias de Post modificado es indecidible si A tiene al menos 2 símbolos.

Si A tiene dos símbolos, podemos codificar palabras de cualquier alfabeto, de manera que sea un homomorfismo y tal que el problema codificado tenga solución si y solo si el original la tiene. Por ejemplo, si el alfabeto es $A = \{a, b, c\}$, podemos codificar $a = 00$, $b = 01$, $c = 11$. Así una ficha se codificaría símbolo a símbolo:

$$\begin{array}{|c|}\hline abb \\ \hline ca \\ \hline \end{array} \rightarrow \begin{array}{|c|}\hline 000101 \\ \hline 1100 \\ \hline \end{array}$$

Codificando así todas las fichas, el problema original tiene solución si y solo si el problema codificado con dos símbolos tiene solución.

PCPM no Decidible

Vamos a reducir el lenguaje universal UNIVERSAL(M, w) a este problema.
Vamos a suponer una MT M y una palabra w. La pregunta es si $w \in L(M)$. Vamos a construir un problema de correspondencias de Post modificado con la misma solución.
El alfabeto que vamos a considerar para el alfabeto del problema de las correspondencias es el alfabeto de la MT, más el conjunto de estados, más un separador que no esté en los conjuntos anteriores $*$

Demostración (Cont.)

- Si $w = a_1 \dots a_n$, introducimos el bloque $\begin{array}{|c|}\hline * \\ \hline *q_0 a_1 \dots a_n * \\ \hline \end{array}$, que será el bloque de comienzo.
- Por cada transición $\delta(q, a) = (q', b, D)$, introducimos el bloque $\begin{array}{|c|}\hline qa \\ \hline bq' \\ \hline \end{array}$
- Por cada transición $\delta(q, a) = (q', b, I)$, introducimos el bloque $\begin{array}{|c|}\hline cqa \\ \hline q'cb \\ \hline \end{array}$ para cada c del alfabeto de la MT
- Para cada símbolo a de la MT introducir $\begin{array}{|c|}\hline a \\ \hline a \\ \hline \end{array}$
- Los bloques $\begin{array}{|c|}\hline * \\ \hline * \\ \hline \end{array}$, $\begin{array}{|c|}\hline * \\ \hline \#* \\ \hline \end{array}$, $\begin{array}{|c|}\hline * \\ \hline *\# \\ \hline \end{array}$
- Para cada $q \in F$ (estado final), añade los bloques $\begin{array}{|c|}\hline aq \\ \hline q \\ \hline \end{array}$, $\begin{array}{|c|}\hline qa \\ \hline q \\ \hline \end{array}$, y el bloque $\begin{array}{|c|}\hline q** \\ \hline * \\ \hline \end{array}$.

Demostración (Cont.)

- La idea de la reducción es representar en los bloques de una posible solución el cálculo que realiza la MT para la palabra de entrada w escribiendo el historial de sus configuraciones.
- Cada configuración (q, u, v) se representa por la cadena uqv y las distintas configuraciones se separan por $*$.
- En la parte de abajo se lleva una configuración de ventaja respecto a la parte de arriba. Eso ocurre para cada cálculo, pero cuando se llega a un estado de aceptación, entonces la parte de abajo empieza a disminuir símbolo a símbolo paso a paso y copiándose arriba, hasta que quede abajo $\ldots * q *$ y arriba $\ldots * qa *$ ó $\ldots * aq *$ donde q es el estado final. Entonces podemos completar con el último bloque $\begin{array}{|c|}\hline q** \\ \hline * \\ \hline \end{array}$ y se completa la solución.
- Así se obtiene una solución si y solo si el cálculo de la MT llega a un estado de aceptación.
- Como el Problema Universal se reduce al PCP modificado y el problema universal no es decidible, entonces el PCP no es decidible.

Ejemplo

Consideremos la MT
$M = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \{0,1,X,Y,\#\}, \delta, q_0, \#, \{q_4\})$ donde las transiciones no nulas son las siguientes:

$$\begin{array}{ll} \delta(q_0,0) = (q_1, X, D) & \delta(q_0, Y) = (q_3, Y, D) \\ \delta(q_1,0) = (q_1, 0, D) & \delta(q_1, 1) = (q_2, Y, I) \\ \delta(q_1, Y) = (q_1, Y, D) & \delta(q_2, 0) = (q_2, 0, I) \\ \delta(q_2, X) = (q_0, X, D) & \delta(q_2, Y) = (q_2, Y, I) \\ \delta(q_3, Y) = (q_3, Y, D) & \delta(q_3, \#) = (q_4, \#, D) \end{array}$$

y la palabra de entrada 000111
Una solución del PCP modificado, tiene que empezar con la ficha:

$*$
$*q_0000111*$

Ejemplo (Cont.)

$*$
$*q_0000111*$

Cómo $\delta(q_0, 0) = (q_1, X, D)$, tenemos la ficha $\begin{array}{|c|}\hline q_0 0 \\ \hline X q_1 \\ \hline \end{array}$ la única forma de proceder en una posible solución es:

$*$	$q_0 0$	0	0	1	1	1	$*$
$*q_0000111*$	Xq_1	0	0	1	1	1	$*$

Si nos fijamos, hemos copiado en la parte superior la configuración que teníamos en la parte inferior y en la inferior se ha añadido la configuración del siguiente paso en el cálculo de la MT. Se puede comprobar que esto es siempre así.

Ejemplo (Cont.)

De esta forma, al final del cálculo de la MT llegaremos a una situación en la que tendremos lo siguiente en las partes superior e inferior de una solución (representamos como una única ficha la solución parcial):

$u*$
$u*XXXYYY\#q_4*$

donde u es la misma palabra en las dos partes. Podemos ver que aún no es una solucion, pero como q_4 es estado final, podemos continuar con los bloques

$u*$	X	X	X	Y	Y	Y	$\#q_4$	$*$
$u*XXXYYY\#q_4*$	X	X	X	Y	Y	Y	q_4	$*$

Si nos damos cuenta, el estado final permite copiar lo que hay de más en la parte inferior en la parte superior, y en la parte inferior se copia todo menos un símbolo que esté al lado del estado final.

Ejemplo (Cont.)

Si vamos repitiendo este proceso, cada vez podemos disminuir lo que hay en la parte inferior en un símbolo de la cinta, hasta que lleguemos a una situación en la que tenemos:

$w*$
$w*q_4*$

Ahora completamos la solución con el bloque $\begin{array}{|c|}\hline q_4** \\ \hline * \\ \hline \end{array}$, que existe por ser q_4 final, obteniendo una solución:

$w*$	q_4**
$w*q_4*$	$*$

Así se pueden completar dos palabras iguales si y solo si M llega a un estado final cuando tiene a w como entrada.

El PCP es indecidible

Teorema

El problema de las correspondencias de Post es indecidible

Se reduce el problema de las correspondencias de Post modificado al problema de las correspondencias de Post PCPM(A1,A2) $\propto$ PCP(B1,B2)

Supongamos que nos dan un conjunto de bloques $\left[\begin{array}{c}u_1\\v_1\end{array}\right], \left[\begin{array}{c}u_2\\v_2\end{array}\right], \ldots, \left[\begin{array}{c}u_n\\v_n\end{array}\right]$ y que el primero necesariamente es el bloque 1, como un ejemplo del problema PCPM(A1,A2)

Construimos el siguiente problema de Post, PCP(B1,B2), con dos símbolos nuevos: $\blacklozenge, \blacksquare$ y con los bloques:

$$\left[\begin{array}{c}\blacklozenge u_1\\ \blacklozenge v_1 \blacklozenge\end{array}\right], \left[\begin{array}{c}\blacklozenge u_1\\ v_1 \blacklozenge\end{array}\right], \left[\begin{array}{c}\blacklozenge u_2\\ v_2 \blacklozenge\end{array}\right], \ldots, \left[\begin{array}{c}\blacklozenge u_n\\ v_n \blacklozenge\end{array}\right], \left[\begin{array}{c}\blacklozenge \blacksquare\\ \blacksquare\end{array}\right]$$

Donde, si $u = a_1 a_2 \ldots a_n$, se entiende que

$$\blacklozenge u = \blacklozenge a_1 \blacklozenge a_2 \ldots \blacklozenge a_n, \quad u \blacklozenge = a_1 \blacklozenge a_2 \ldots \blacklozenge a_n \blacklozenge, \quad \blacklozenge u \blacklozenge = \blacklozenge a_1 \blacklozenge a_2 \ldots \blacklozenge a_n \blacklozenge$$

PCP No decidible. Continuación

- PCPM(B1,B2): $\boxed{\begin{matrix}u_1\\v_1\end{matrix}}, \boxed{\begin{matrix}u_2\\v_2\end{matrix}}, \ldots, \boxed{\begin{matrix}u_n\\v_n\end{matrix}}$
- PCP(B1,B2):

$$\boxed{\begin{matrix}\blacklozenge u_1\\ \blacklozenge v_1 \blacklozenge\end{matrix}}, \boxed{\begin{matrix}\blacklozenge u_1\\ v_1 \blacklozenge\end{matrix}}, \boxed{\begin{matrix}\blacklozenge u_2\\ v_2 \blacklozenge\end{matrix}}, \ldots, \boxed{\begin{matrix}\blacklozenge u_n\\ v_n \blacklozenge\end{matrix}}, \boxed{\begin{matrix}\blacklozenge\blacksquare\\ \blacksquare\end{matrix}}$$

Como se puede ver, las soluciones a ambos problemas son equivalentes. Si el modificado tiene solución, entonces los bloques asociados con los dos símbolos nuevos son una solución del PCP finalizados por el bloque $\boxed{\begin{matrix}\blacklozenge\blacksquare\\ \blacksquare\end{matrix}}$.

Por otra parte en cualquier solución del PCP, hay que empezar por la ficha $\boxed{\begin{matrix}\blacklozenge u_1\\ \blacklozenge v_1 \blacklozenge\end{matrix}}$ ya que es la única que comienza en ambas partes por el mismo símbolo $\blacklozenge$ y además terminar por $\boxed{\begin{matrix}\blacklozenge\blacksquare\\ \blacksquare\end{matrix}}$. Si a todos los bloques distintos del último les quitamos los dos símbolos especiales obtenemos una solución del PCP modificado que empieza por la ficha 1,

Problemas sobre Gramáticas

Suponemos que G, G_1 y G_2 son gramáticas independientes del contexto dadas y R es un lenguaje regular.

- Saber si $L(G_1) \cap L(G_2) = \emptyset$.
- Determinar si $L(G) = T^*$, donde T es el conjunto de símbolos terminales.
- Comprobar si $L(G_1) = L(G_2)$.
- Determinar si $L(G_1) \subseteq L(G_2)$.
- Determinar si $L(G_1) = R$.
- Comprobar si $L(G)$ es regular.
- Determinar si G es ambigua.
- Conocer si $L(G)$ es inherentemente ambiguo.
- Comprobar si $L(G)$ es determinista.

Reducción: ambigüedad

Teorema

Saber si una gramática independiente del contexto es ambigua (AMBIGUA(G)) es indecidible.

Se reduce el problema de las correspondencias de Post. Supongamos el alfabeto A y los bloques $\boxed{\begin{array}{c}u_1\\v_1\end{array}}, \cdots, \boxed{\begin{array}{c}u_k\\v_k\end{array}}$.

Sea $B = A \cup \{b_1, \ldots, b_k\}$ donde $b_i \notin A$ y construimos la siguiente gramática:

$S \to C|D$

$C \to u_i C b_i | u_i b_i, \quad i = 1, \ldots, k$

$D \to v_i D b_i | v_i b_i, \quad i = 1, \ldots, k$

La solución al problema de las correspondencias de Post es equivalente a que la gramática sea ambigua.

Demostración

Supongamos el alfabeto A y los bloques $\boxed{\begin{matrix}u_1\\v_1\end{matrix}}, \cdots, \boxed{\begin{matrix}u_k\\v_k\end{matrix}}$.

Sea $B = A \cup \{b_1, \ldots, b_k\}$ donde $b_i \notin A$ y construimos la siguiente gramática:
$S \to C|D, \quad C \to u_i C b_i | u_i b_i, \quad i = 1, \ldots, k, \quad D \to v_i D b_i | v_i b_i, \quad i = 1, \ldots, k,$
Se basa en lo siguiente:

- Las palabras generadas por la gramática son las generadas a partir de C más las generadas a partir de D
- Las palabras generadas a partir de C son de la forma $u_{i_1} \ldots u_{i_l} b_{i_l} \ldots b_{i_1}$, donde $i_j \in \{1, \ldots, k\}$. Sólo hay una forma de generar una de estas palabras a partir de C.
- Las palabras generadas a partir de D son de la forma $v_{i_1} \ldots v_{i_l} b_{i_l} \ldots b_{i_1}$, donde $i_j \in \{1, \ldots, k\}$. Sólo hay una forma de generar una de estas palabras a partir de D.
- La gramática es ambigua cuando una misma palabra u es generada a partir de C y a partir de D, es decir cuando $u = u_{i_1} \ldots u_{i_n} b_{i_l} \ldots b_{i_n} = v_{i'_1} \ldots v_{i'_{n'}} b_{i'_1} \ldots b_{i'_{n'}}$. Esto solo ocurre si $n' = n$ e $i_j = i'_j, \forall j$. Es decir cuando existen $(i_1, \ldots, i_n)$ tal que $u_{i_1} \ldots u_{i_n} = v_{i_1} \ldots v_{i_n}$, es decir cuando el PCP tiene solución.