Autómatas y lenguajes formales

Víctor Mijangos de la Cruz

vmijangosc @ ciencias.unam.com

IV. Jerarquía de Chomsky e Introdccuón a la Decibilidad

Jerarquía de Chomsky

Lenguajes recursivamente enumerables

Lenguaje recursivamente enumerable

Un lenguaje L es recursivamente enumerable si existe una máquina de Turing M tal que $L = L(M)$.

En lo siguiente, nos trazamos los siguientes objetivos:

1. Mostrar que hay lenguajes que no son RE.
2. Definir los lenguajes recursivos como un subconjunto de los lenguajes RE.
3. Definir los lenguajes dependientes del contexto como un subconjunto de los lenguajes recursivos.
4. Mostrar que los lenguajes independientes del contexto son un subconjunto de los dependientes del contexto.
5. Mostrar que los lenguajes regulares con un subconjunto de los independientes de contextos.
6. Introducir la decibilidad y ver su relación con los tipos de lenguajes.

Lenguaje RE

Como ejemplo de un lenguaje recursivamente enumerable podemos poner varios de los que ya hemos revisado.

- Todo lenguaje regular es un lenguaje RE.
- Todo lenguaje libre de contexto es un lenguaje RE.
- El lenguaje $L = \{a^n b^n c^n : n > 0\}$, que no es libre de contexto ni regular, es un lenguaje RE.
- El lenguaje universal usado para definir las máquinas de Turing universales es un lenguaje RE.

Un lenguaje no RE

Como mostramos anteriormente, cada máquina de Turing tiene un código asociado. Definimos la i-ésima máquina de Turing M_i como la máquina de Turing cuyo código es w_i tal que $i = f(w_i)$ (f es la función de indexación de cadenas binarias).
Nota: Si el código w_i no representa un código de transiciones válido, entonces $L(M_i) = \emptyset$. Por ejemplo, la novena máquina de Turing $w_6 = 11$ no contiene ninguna transición, pues comienza por 1, por lo que $L(M_6) = \emptyset$.

Lenguaje de la diagonal

Definimos el lenguaje de la diagonal, denotado como L_d, de la siguiente forma:

$$L_d = \{w_i : w_i \notin L(Mi)\}$$

Lenguaje de la diagonal

Podemos crear una tabla que para todo $i, j \in \mathbb{N}$ se defina la entrada de la tabla como:

$$T_{i,j} = \begin{cases} 1 & \text{si } w_j \in L(M_i) \\ 0 & \text{si } w_j \notin L(M_i) \end{cases}$$

M_i/w_j	1	2	3	4	$\cdots$
1	**0**	1	1	0	$\cdots$
2	1	**1**	0	0	$\cdots$
3	0	0	**1**	1	$\cdots$
4	0	1	0	**1**	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Decimos que el renglón i-ésimo es el **vector característico** del lenguaje $L(M_i)$.
Nota: El vector característico de L_d sería el complemento de la diagonal.

L_d no es RE

Teorema

El lenguaje de la diagonal L_d no es recursivamente enumerable (RE).

Supóngase lo contrario: que L_d es RE. Entonces $\exists M$ t.q. $L(M) = L_d$. Ya que tratamos con alfabeto binario $\Sigma = \{0, 1\}$, M debe estar en la lista de la tabla que acabamos de construir.
Supóngase que $M = M_i$, la máq. de Turing del i-ésimo renglón. Como veremos esto lleva a contradicción.
Nos preguntamos si $w_i \in L(M_i)$:

1. Si $w_i \in L_d$, $w_i \in L(M_i)$ por lo que por definición implica que $w_i \notin L_d$, lo que es una contradicción.
2. Si $w_i \notin L_d$, $w_i \notin L(M_i)$ pero por definición entonces $w_i \in L_d$

En cualquier caso, tenemos una contradicción, ésta surgió de suponer que L_d tiene una máquina de Turing que lo acepta. Por tanto, no existe ninguna máquina de Turing que acepte L_d. Esto es, L_d no es RE.

Lenguajes recursivos

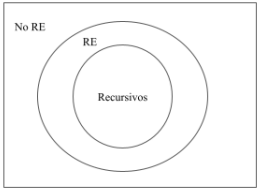
Lenguaje recursivo

Decimos que L es un lenguaje recursivo si $L = L(M)$ para alguna máquina de Turing M que cumple:

1. Si $w \in L$, entonces M acepta y se detiene.
2. Si $w \notin L$, entonces M se detiene sin entrar en un estado de aceptación.

Podemos dividir los lenguajes de la siguiente forma:

- Lenguajes recursivos.
- Lenguajes RE, pero no recursivos.
- Lenguajes que no son RE.



Complementos de lenguajes

Teorema

Si L es un recursivo, entonces su complemento L^c también lo es.

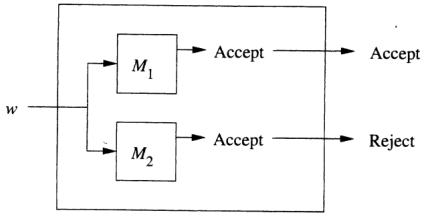
Teorema

Sea L un lenguaje cuyo complemento es recursivamente enumerable (RE), entonces L es recursivo y por tanto $L = (L^c)^c$ es recursivo.

Se construye una máq. de Turing M con M_1 (L) y M_2 (L^c) tal que:

- Si $w \in L$, M_1 acepta y M se detiene.
- Si $w \notin L$, entonces M_2 acepta y M se detiene.

Por tanto M acepta L y siempre se detiene.



Complementos de lenguajes

En resumen, se cuenta con 4 casos posibles:

1. Tanto L, como L^c son recursivos.
2. Ni L ni L^c son RE.
3. L es RE pero no recursivo, y L^c no es RE.
4. L^c es RE pero no recursivo y L no es RE.

Ejemplo: Tenemos los lenguajes:

- L_u (lenguaje universal) es RE pero no recursivo, por lo que L_u^c no es RE.
- L_d (lenguaje de la diagonal) no es RE, tampoco L_d^c lo es.

Lenguajes dependientes del contexto

Lenguaje dependiente del contexto

Un lenguaje se dice que es dependiente del contexto si tiene una gramática $G = (\Sigma, \Delta, S, R)$ donde sus reglas son de la forma:

$$\alpha X \beta \rightarrow \alpha \gamma \beta$$

Donde $\alpha, \beta, \gamma \in (\Sigma \cup \Delta)^*$, $X \in \Delta$.

Ejemplo: El lenguaje $L = \{a^n b^n c^n : n \geq 0\}$ tiene la gramática:

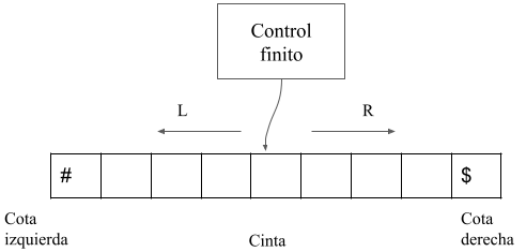
$$\begin{aligned} S &\rightarrow aBSc|\epsilon \\ Ba &\rightarrow aB \\ Bb &\rightarrow bb \\ Bc &\rightarrow bc \end{aligned}$$

Autómata linealmente acotado

Autómata linealmente acotado

Una autómata linealmente acotado (ALA) es una máquina de Turing no determinista $B = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ tal que:

- Tiene dos símbolos destacados $\#$ y $\$$ en Γ y,
- Para todo estado $p, q \in Q$ y $X \in \Gamma$, se tiene que $(p, \#, L), (p, \$, R) \notin \delta(q, X)$



ALAs y lenguajes dependientes del contexto

Teorema

Un lenguaje L es dependiente del contexto si y sólo si existe un autómata linealmente acotado B, tal que $L = L(B)$.

Teorema

Los lenguajes dependientes de contexto son recursivos.

Construimos una máquina de Turing M tal que:

1. M simula a B el autómata linealmente acotado de $L = L(B)$ sobre una cadena w. Si $w \in L$, M acepta la acepta.
2. Si $w \notin L$, B puede estar en cada uno de los $|Q|$ estados y la cinta de B tiene $|w| + 2$ símbolos (los símbolos de w y $\#$, $\$$. Cada celda puede tener un símbolo de Γ y el cabezal sólo lee uno, por tanto se pueden dar a lo más $|Q||\Gamma|^{|w|+2}(|w| + 2)$ configuraciones. Así si M rebasa este número de configuraciones se dentendrá sin aceptar.

CDL y CFL

Teorema

Toda gramática libre de contexto es una gramática dependiente del contexto.

Basta tomar las producciones $X \to \gamma$ como $\epsilon X \epsilon \to \epsilon \gamma \epsilon$

Teorema

Un autómata linealmente acotado $B = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ simula a un autómata con pila $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$.

Q, Σ y F permanecen igual. Sólo basta ver que la cinta acotada de B puede simular la pila Γ de P.

Podemos formular las transiciones en dos cintas. Una cinta para la cadena de entrada, y una segunda cinta para la pila. La segunda cinta siempre iniciará con el símbolo Z_0. Estas cintas son acotadas, y pueden reducirse a una sola cinta.

Lenguajes recursivos que no son dependientes

Sea $G_1, G_2, ...$ todas las gramáticas dependientes de contextos. Sea $w_1 = \epsilon, w_2 = 0, w_3 = 1, w_4 = 00, ...$ una numeración binaria tal que w_i corresponde a G_i. Definimos el lenguaje:

$$L = \{w_i : w_i \notin L(G_i)\}$$

1. L es recursivo: pues podemos construir una máquina de Turing que tome el par (G_i, w_i) para revisar si w_i es aceptado o no por G_i que siempre se detenga, pues los lenguajes dependientes del contexto son recursivos.
2. L no es dependiente del contexto. Si suponemos que lo es $\exists k$, t.q. $L = L(G_K)$ con código w_k. Pero entonces, si $w_k \in L$, $w_k \notin L(G_k) = L$, y si $w_k \notin L$, entonces $w_k \in L(G_k) = L$.

Jerarquía con lenguajes dependientes

Hasta ahora tenemos:

1. Todo lenguaje libre de contexto es dependiente de contexto.
2. Hay lenguajes que son dependientes de contexto pero no libres de contexto (lema de Ogden).
3. Los lenguajes dependientes de contexto son recursivos.
4. Hay lenguajes recursivos que no son dependientes del contexto.

Por tanto, tenemos las contenciones propias:

$$CFL \subset CDL \subset RL \subset RE$$

Lenguajes regulares

Teorema

Toda gramática regular es una gramática libre de contexto.

Las gramáticas regulares son de la forma $X \to aY$, que es un caso particular de $X \to \gamma$ libre de contexto.

Teorema

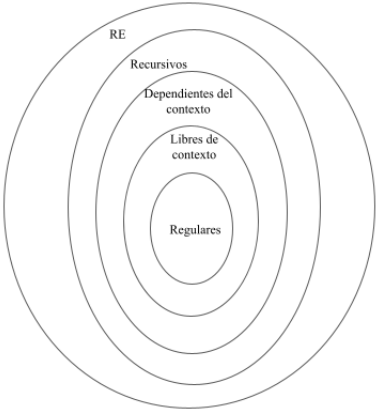
Un autómata finito puede ser simulado por un autómata de pila.

Basta ignorar la pila; esto se puede hacer tomando un único símbolo Z_0, tal que toda transición sea de la forma $\delta(q, a, Z_0) = (p, Z_0)$.

Jerarquía de Chomsky

La jerarquía de Chomsky determina que existe una relación de contención propia en el siguiente orden:

1. Lenguajes no Recursivamente Enumerables.
2. Lenguajes Recursivamente Enumerables (Tipo 0).
3. Lenguajes Recursivos.
4. Lenguajes Dependientes del Contexto (Tipo 1).
5. Lenguajes Independientes del Contexto (Tipo 2).
6. Lenguajes Regulares (Tipo 1).



Jerarquía de Chomsky

Lenguaje	**Gramática**	**Autómata**
RE	$\alpha \to \gamma$	Máquina de Turing
Recursivo	$\alpha \to \gamma$	Máq. de Turing decidible
Dependiente del contexto	$\alpha X \beta \to \alpha \gamma \beta$	Autómata linealmente acotado
Independiente del contexto	$X \to \gamma$	Autómata de Pila
Regular	$X \to aY$	Autómata finito

Introducción a la indecibilidad

Computabilidad y lenguajes

Reconocer las cadenas de un lenguaje una manera formal de expresar un **problema**.
Ya que dado un alfabeto finito Σ, el conjunto de cadenas en Σ^* es numerable, por lo que el número de problemas, lenguajes es $2^{|\Sigma^*|} = \aleph_1$.

Objetivo: *¿Qué lenguajes pueden ser definidos por un dispositivo computacional?*

El problema de no conocer cuando (si es el caso) algo va a ocurrir en un programa determina la incapacidad de determinar qué hace un programa.

Decibilidad

Problema decidible

Un problema, asociado a un lenguaje L recursivamente enumerable, es decidible si L es un leguaje recursivo. En otro caso, el problema es **indecidible**.

El concepto de decibilidad se asocia al de **algoritmo**, pues se espera que un algoritmo se detenga aunque no encuentre una solución al problema.

Teorema

Los lenguajes linealmente acotados son decidibles.

Ejemplo: L_u es indecidible

Teorema

El lenguaje universal L_u es indecidible.

El lenguaje L_u es el lenguaje de los pares (M, w). L_u es RE porque existe U máquina de Turing Universal que lo acepta. Para probar que no es recursivo supongamos lo contrario y asumamos que sí lo es. Como L_u recursivo, L_u^c es recursivo, por tanto $\exists M$ t.q. $L_u^c = L(M)$. Modificamos M para construir M' tal que:

1. Para cada cadena $w \in \Sigma^*$ se copia como $w111w$.
2. M' simula a M sobre la entrada $w111w$.

De esta forma, so $w = w_i$ entonces M' determina si w_i no es aceptado por M_i, el par (M_i, w_i); esto es, M' determina L_u^c, por lo que nos dirá cuando $w_i \notin L(M_i)$. Pero esto es el lenguaje de la diagonal, que sabemos que no es RE, por tanto no existe una máquina de Turing. Por tanto, L_u no puede ser recursivo; i.e., es indecidible.

Determinar indecibilidad

Podemos preguntarnos entonces si podemos determinar cuando un algoritmo para un problema se detendrá siempre.
Esto es, nos preguntamos:

> *Dada un problema/lenguaje existe un algoritmo/máquina de Turing que siempre se detenga.*

Esto equivale a preguntar si el problem es decidible o indecidible. Sin embargo, veremos que decidir esto es por sí mismo imposible.

Dos programas

Podemos proponer dos algoritmos que hagan exactamente lo mismo, imprimir "Hello, World!"; sin embargo, como se puede ver, uno de estos algoritmos siempre va a hacer la impresión, mientras que el otro va a correr indefinidamente.

Algorithm Imprimir "Hello, World!"

```
1: procedure PRINT1( )
2:     printf(``Hello, World!'')
3: end procedure
```

```
1: procedure PRINT2(n ∈ ℕ)
2:     total ← 3
3:     while True do
4:         for x = 1; x ≤ total − 2 do
5:             for y = 1; y ≤ totla − x − 1 do
6:                 if x^n + y^n = z^n then
7:                     printf(``Hello,
   World!'')
8:                 end if
9:                 total, y ← +1
10:            end for
11:            x ← x + 1
12:        end for
13:    end while
14: end procedure
```

Problema de detención

Problema de detención

Dado un programa P que toma el argumento de entrada I, el problema de la detención busca determinar cuándo P al correr con la entrada I terminará; es decir, regresará un valor de salida.

De aquí, podemos preguntarnos ¿Qué problemas cuentan con un algoritmo que termine?

Programa hipotético

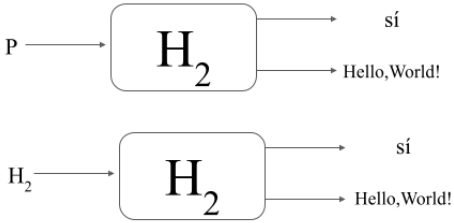
Supóngase que existe un programa hipotético H que toma como entrada un programa P y la entrada de este programa I, de tal forma que H determina si P termina o no.

Si H siempre responde que "sí" para un programa P, entonces decimos que P es **decidible**. De otra forma, es indecidible.

Programa hipotético (2)

Podemos hacer las siguientes modificaciones al programa H para formar H_2:

1. La entrada de H_2 es sólo P (I no se toma en cuenta)
2. H_2 pregunta que hará P si su entrada es este mismo P
3. H_2 imprime "sí" si P imprime "Hello, World!", y si P no lo imprime, entonces H_2 imprime "Hello, World!".



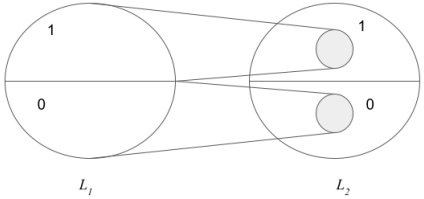
Paradoja: Si H_2 se toma a sí mismo como entrada, entonces imprimirá "Hello, World!" si y sólo si no imprime "Hello, World!".

Reducciones

Reducción

Dada un problema L_1 y otro L_2, una reducción de L_1 a L_2 es una máquina de Turing que toma una instancia de L_1 de entrada y se detiene con una instancia de L_2 en su cinta.

En términos generales, la reducción L_1 a L_2 consiste en un algoritmo que toma instancias de L_1 y las lleva a ubstancias de L_2 (con las mismas soluciones).



Reudcciones e indecibilidad

Teorema

Si existe una reducción de L_1 a L_2, entonces:

1. Si L_1 es indecidible, entonces L_2 también lo es.
2. Si L_1 no es RE, entonces L_2 tampoco lo es.

1) Supóngase que L_1 es indecidible. Si L_2 fuera decidible, podemos combinar la reducción R de L_1 a L_2 con el algoritmo que decide a L_2 M, de tal forma que $M \circ R$ sería un algoritmo que decide L_1, que contradice la suposición inicial. Por tanto L_2 es indecidible.

2) Ahora, si L_1 no es RE, suponiendo que L_2 fuera RE, entonces $\exists M$ máq. de Turing que acepta las entradas de L_2. Entonces para todo $w \in L_1$, $\exists x \in L_2$ t. q. x es la reducción de w. Por tanto, si x es aceptado por M, esto implica que w es aceptado en la composición $M \circ R$. Esto implicaría que L_1 es RE, que es una contradicción. Por tanto, L_2 no es RE.

Ejemplo

Considérense los siguientes problemas:

- Supóngase que nuestro problema L_1 consiste en dividir un ángulo en 6 partes iguales usando escuadra y compás.
- El problema L_2 es reducir un ángulo en 3 partes iguales.

La reducción de L_1 a L_2 puede plantearse con el siguiente algoritmo:

1. Divide el ángulo en 6 partes iguales.
2. Ignora cada división impar del paso anterior.

Entonces, si podemos mostrar que L_2 es indecidible, también lo será L_1.

Lenguaje vacío

Lenguaje vacío

El lenguaje vacío es el lenguaje de los códigos de todas las máquinas de Turing con lenguaje vación:

$$L_e = \{M : L(M) = \emptyset\}$$

Lenguaje no-vacío

El lenguaje no-vacío corresponde a los códigos de las máquinas de Turing cuyo lenguaje es diferente del vacío:

$$L_{ne} = \{M : L(M) \neq \emptyset\}$$

Estos lenguajes corresponden a los problemas de determinar si una máquina de Turing acepta o no alguna entrada.

El no-vacío es RE

Teorema

El lenguaje no vacío L_{ne} es Recursivamente Enumerable.

Construiremos una máquina de Turing M de la siguiente forma:

1. M toma como entrada al código M_i.
2. M genera una entrada w que M_i debe aceptar.
3. M prueba si M_i acepta w. Para esto simula una máquina universal de Turing U.
4. Si M_i acepta w, M acepta a M_i

Por tanto, existe una máq. de Turing que acepta L_{ne}

Dos corolarios del teorema anterior

Corolario

El lenguaje no vacío L_{ne} no es recursivo. Esto es, el problema de determinar si un lenguaje es diferente del vacío es indecidible.

La construcción de M es una reducción de L_{ne} a L_u, que sabemos que es indecidible.

Corolario

El lenguaje vacío L_e no es Recursivamente Enumerable.

Como $L_e = L_{ne}^c$, y ya que L_{ne} es indecidible, entonces por el Teorema de complementos concluimos que L_e no es recursivamete enumerable.

Propiedades de lenguajes

Propiedad

Una propiedad sobre los lenguajes Recursivamente Enumerable es un conjunto de lenguajes Recursivamente Enumerables. Esto es, la propiedad ϕ se define como $\phi = \{L : L \in RE\}$

Ejemplo: La propiedad de ser libre de contexto es el conjunto de todos los lenguajes libres de contexto. La propiedad de ser vacío son todos los lenguajes vacíos.

Propiedad trivial

Una propiedad es **trivial** si es el conjunto vacío ($\emptyset$) o es el total de todos los lenguajes RE. De otra forma, decimos que la propiedad es **no trivial**.

Lenguaje de una propiedad

Lenguaje de una propiedad

Dada una propiedad ϕ sobre los lenguajes RE, el lenguaje de la propiedad, denotado L_ϕ, es el conjunto de los códigos para las máquinas de Turing M_i tal que $L(M_i) \in \phi$. Esto es:

$$L_\phi = \{M_i : L(M_i) \in \phi\}$$

Ejemplo: La propiedad de que el lenguaje sea vacío tiene como lenguaje a L_e. Mientras que la propiedad de que el lenguaje sea no vacío tiene lenguaje L_{ne}.

Teorema

La decibilidad de una propiedad ϕ es equivalente a que L_ϕ sea recursivo.

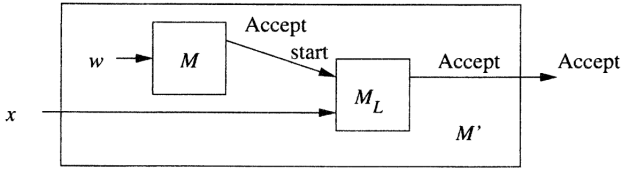
Al mostrar que L_{ne} no es recursivo, mostramos que la propiedad de un lenguaje de ser no vacío es indecidible.

Lema de Rice

Teorema

Toda propiedad no trivial de los lenguajes Recursivamente Enumerables es indecidible.

Sea ϕ una propiedad no trivial de los lenguajes RE; esto es, $\exists L \in \phi$ con $L \neq \phi$. Y ya que L es RE, tomemos M_L t.q. $L(M_L) = L$.
Reduciremos L_ϕ a L_u con el siguiente algoritmo/máq. de Turin M' (de dos cintas):



- La entrada es un par (M, w)
- La primera cinta simula M en w.
- La segunda cinta simula M_L sobre una entrada x.

Demostración (Continuación)

La máq. de Turing M', que funge como la reducción, acturaá de la siguiente forma:

- Para simular M bajo w: M' escribe M y w en su primera cinta y simula una máquina Universal de Turing que toma como entrada el par (M, w).
- Si M no acepta w, M' no hace nada más, y en este caso, M' no acepta su propia entrada x, por lo que $L(M') = \emptyset$ (trivial), por lo que $M' \notin L_\phi$.
- Si M acepta w, entonces M' simula M_L con su propia entrada x; esto es, $L(M') = L$ y $M' \in L_\phi$.

Vemos que M' acepta una entrada sólo si la máq. Universal acepta (M, w). Esto es $M' \in L_\phi$ si $(M, w) \in L_u$. Hemos reducido L_ϕ a L_u.
Por tanto, concluimos que L_ϕ es recursivo. Esto es, el problema de determinar una propiedad no trivial sobre RE es indecidible.

Problemas indecidibles

Problema de Correspondencia de Post

Par de correspondencia

Dadas dos listas $A = \{w_1, w_2, ..., w_k\}$ y $B\{x_1, x_2, ..., x_k\}$, un par de correspondencia es un par $(w_i, x_i) \in A \times B$.

Problema de Correspondencia de Post

El problema de correspondencia de Post (PCP) consiste en: dos listas $A = \{w_1, w_2, ..., w_k\}$ y $B\{x_1, x_2, ..., x_k\}$ de la misma longitud de símbolos.
El objetivo es encontrar una solución (secuencia de índices) $i_1, i_2, ..., i_m$ tal que $w_{i_1} w_{i_2} ... w_{i_k} = x_{i_1} x_{i_2} ... x_{i_k}$, donde $w_{i_1} w_{i_2} ... w_{i_k} \in A^*$, $x_{i_1} x_{i_2} ... x_{i_k} \in B^*$.

Ejemplo

Una **instancia del PCP** es considerar valores específicos para las listas A y B. Por ejemplo, la siguiente asignación es una instancia:

	A	B
i	w_i	x_i
1	1	111
2	10111	10
3	10	0

una solución para esta instancia del PCP es la secuencia de índices $2, 1, 1, 3$ puesto que:

$$w_2 w_1 w_1 w_3 = 101111110 = x_2 x_1 x_1 x_3$$

Ejemplo (Instancia sin solución)

Si consideramos la siguiente instancia del PCP:

	A	B
i	w_i	x_i
1	10	101
2	011	11
3	101	011

Vemos que no existe una solución posible.

Indecibilidad del PCP

Teorema

El Problema de Correspondencia de Post (PCP) es indecidible.

Para mostrar este resultado, primero crearemos un nuevo problema de correspondecia modificado (MPCP), veremos que este MPCP es indecidible y que el PCP original se reduce al MPCP.
PCP Modificado: Dada una instancia del PCP con $A = [w_1, ..., w_k]$ y $B = [x_1, ..., x_k]$ contruios el PCP modificado o MPCP de la siguiente forma:

- Además de los símbolos considerados en la instancia, se agregan dos nuevos símbolos y $\$$.
- Se construyen las instancias como: $C = [y_0, y_1, ..., y_k, y_{k+1}]$ y $D = [z_0, z_1, ..., z_k, z_{k+1}]$ tal que:
  1. $\forall i \in 1, 2, ..., k$, $y_i = w_i*$ y $z_i = *x_i$
  2. Para el primer símbolo tenemos que $y_0 = *y_1$ y $z_0 = z_1$
  3. Mientras que $y_{k+1} = \$ = z_{k+1}$

Demostración (continuación)

Indecibilidad de MPCP: Construímos una máquina de Turing M y veremos que pasa con el problema: "M acepta w si la construcción de MPCP tiene solución". La máquina de Turing tiene las siguientes reglas:

1. El primer par es $A = [\#]$ y $B = [\#q_0w\#]$
2. Los símbolos de la cinta y el separador $\#$ se pueden agregar a cada lista. Se tienen los pares: $A = [X, \#]$ y $B = [X, \#]$ para todo $X \in \Gamma$ (símbolo de cinta). Los símbolos que no involucran el estado pueden copiarse.
3. Para simular el movimiento de M para todo $q \in Q, \setminus F, p \in Q, X, Y, Z \in \Gamma$ tenemos:

A	B	
qX	Yp	si $\delta(q, X) = (p, Y, R)$
ZqX	pZY	si $\delta(q, X) = (p, Y, L)$
$q\#$	$Yp\#$	si $\delta(q, B) = (p, Y, R)$
$Zq\#$	$pZY\#$	si $\delta(q, B) = (p, Y, L)$

4. Si el ID al final de B es de aceptación, se tiene una solución. SI $q \in F$, entonces para todo X, Y existen pares $A = [XqY, Xq, qY]$ y $B = [q, q, q]$
5. El par final es $A = [q\#\#]$ y $B = [\#]$

Demostración (continuación)

Si $w \in L(M)$, se comienza con un par de cadenas (regla 1) y se simula la computación de M con w. Usamos el par de la regla 3 para copiar el estado de cada ID y simular el movimiento de M. Se usa el par de 2 para copiar el símbolo de cinta y la marca $\#$. Si M alcanza un estado final, el par de 4 y un uso final del par de 5 permiten que la cadena A alcance a B y se obtiene solución.
Si tenemos las listas $A = [\#]$ y $B = [\#q_0w\#]$ una solución parcial (i.e. A es prefijo de B). Los pares de 4 y 5 son inútiles. Los estados y uno o más símbolos de cinta en una iD pueden manejar el par en 3 y otros símbolos de cinta; y $\#$ debe ser manejado por 2. Al menos que M alcance el estado de aceptación, las soluciones parciales son de la forma $A = x$ y $B = xy$ (prefijo). Así, si hay una solución, M entra en estado de aceptación. Esto es, *MPCP* es RE.
Por otra parte, si no hay solución M seguirá ejecutándose. Es decir, no es recursivo.
Por tanto, el *MPCP* es indecidible.

Demostración (continuación)

Reducción de MPCP a PCP: Supóngase $i_1, i_2, .., i_m$ es una solución para MPCP con listas A y B, entonces $w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}$.
Reemplazamos los símbolos según la construcción del MPCP que hicimos y obtenemos:

$$*y_1 y_{i_1} \cdots y_{i_m} = z_1 z_{i_1} \cdots z_{i_m}$$

Pero como $y_0 = *y_1$ y $z_0 = z_1$, entonces $y_0 y_{i_1} \cdots y_{i_m} = z_0 z_{i_1} \cdots z_{i_m}$ y tomando el índice $k+1$ con $y_{k+1} = \$$ y $z_{k+1} = *\$$, tenemos:

$$y_0 y_{i_1} \cdots y_{i_m} y_{k+1} = z_0 z_{i_1} \cdots z_{i_m} zk + 1$$

Esto muestra que $0, i_1, ..., i_m, k+1$ es una solución del PCP. Esto es, el PCP es una reducción de MPCP.
PCP es indecidible: Finalmente como resultado de que PCP es reducción de MPCP y de que el MPCP es indecidible, concluimos que el PCP es indecidible.

Ambigüedad de CFG

Teorema

El problema de determinar cuando una gramática libre de contexto es ambigua es indecidible.

Reducimos el problema de la ambigüedad al PCP. Para esto, construimos dada una instancia $A = [w_1, ..., w_k]$ y $B = [x_1, ..., x_k]$ construimos dos gramáticas G_A y G_B.
Tomamos un conjunto de símbolos $a_1, ..., a_k$ que representan elecciones de pares de cadenas en una instancia del PCP. Así a_i representa la elección de w_i de A y de x_i de B.
La gramática para A es la siguiente:

$$\begin{aligned} S &\to w_1 A a_1 | \cdots | w_k A a_k \\ &\to w_1 a_2 | \cdots | w_k a_k \end{aligned}$$

Claramente, el lenguaje de esta gramática es:

$$L_A = \{w_1 w_2 \cdots w_k a_1 a_2 \cdots a_k : w_i \in A\}$$

Demostración (continuación)

Ahora construimos una gramática similar para B:

$$S \to x_1 B a_1 | \cdots | x_k B a_k$$
$$\to x_1 a_2 | \cdots | x_k a_k$$

La cual produce el lenguaje $L_B = \{x_1 x_2 \cdots x_k a_1 a_2 \cdots a_k : x_i \in B\}$. Finalmente, construimos una gramática G_{AB} como sigue:

- Los no terminales $\Delta = \{S, A, B\}$ con S inicial.
- Los terminales son los símbolos $\Sigma = \{w_i, x_i, a_i : i = 1, ..., k\}$
- Las producciones son $S \to A|B$ y las producciones de G_A y G_B.

Demostración (continuación)

G_{AB} es ambigua si y sólo si la instancia (A, B) del PCP tiene solución.
$\Longrightarrow]$ G_{AB} ambigua. Podemos observar que la ambigüedad en esta gramática produce dos derivaciones:

1. $S \Longrightarrow w_{i_1} A a_{i_1} \Longrightarrow^* w_{i_1} \cdots w_{i_m} a_{i_1} \cdots a_{i_m}$
2. $S \Longrightarrow w_{i_1} B a_{i_1} \Longrightarrow^* w_{i_1} \cdots w_{i_m} a_{i_1} \cdots a_{i_m}$

Ya que una derivación es con variables A y otra con B tenemos que
$w_{i_1} \cdots w_{i_m} a_{i_1} \cdots a_{i_m} = x_{i_1} \cdots x_{i_m} a_{i_1} \cdots a_{i_m}$; esto es $w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}$. Por tanto, $i_1, i_2, ..., i_m$ es una solución de la instancia (A, B).
$\Longleftarrow]$ Si (A, B) tiene solución, sea esta $i_1, ..., i_m$, por como construimos la gramática entonces tenemos derivaciones: $S \Longrightarrow w_{i_1} A a_{i_1} \Longrightarrow^* w_{i_1} \cdots w_{i_m} a_{i_1} \cdots a_{i_m}$ y $S \Longrightarrow x_{i_1} B a_{i_1} \Longrightarrow^* x_{i_1} \cdots x_{i_m} a_{i_1} \cdots a_{i_m}$. Pero ya que $i_1, ..., i_m$ es solución tenemos que $w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}$. Por lo que producimos la misma cadena con distintos árboles. Esto es G_{AB} es ambigua.
Esto es una reducción al PCP. Por tanto, decidir si una gramática es ambigua es indecidible.

Otros problemas indecidibles de CFL

Teorema

Sean G_1 y G_2 gramáticas libres de contexto, y sea R una expresión regular. Los siguientes problemas son indecidibles:

- Determinar si $L(G_1) \cap L(G_2) = \emptyset$
- Determinar si $L(G_1) = L(G_2)$
- Determinar si $L(G_1) = L(R)$
- Determinar si $L(G_1) = \Sigma^*$
- Determinar si $L(G_1) \subseteq L(G_2)$
- Determinar si $L(R) \subseteq L(G_1)$

Ecuaciones diofantinas

Una ecuación difoantina es un polinomio:

$$P(x_1, x_2, ..., x_n) = c$$

aplicado sobe valores enteros $x_1, x_2, ..., x_n, c \in \mathbb{Z}$.
Un sistema de ecuaciones diofantinas es un sistema de ecuaciones que busca resolverse con valores enteros. Podemos plantear el siguiente problema:

Dado un sistema de ecuaciones diofantinas, ¿existe una solución entere?

Este problema es **indecidible**.
Planteando $L = \{< w >: w\text{sistema de ecuaciones con solución entera}\}$, se ha demostrado que todo lenguaje RE puede reducirse a un lenguaje de ecuaciones diofantinas. Por tanto, existen instancias que son indecidibles (los lenguas que son RE pero no recursivos).