Ciencias de la Computación I

Nociones básicas de Computabilidad

Problemas y Lenguajes

Un problema se describe con un lenguaje

⇓

Cuanto más formal el lenguaje, más precisa la formulación del problema

Los problemas se clasifican en:

- Problemas de Decisión
- Problemas de Salida General

Los lenguajes formales se clasifican en:

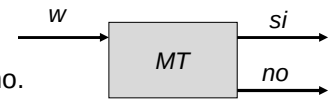
- Lenguajes Recursivos
- Lenguajes Recursivo Enumerables

Problemas

- Problema de Decisión:

Es un problema formulado como una pregunta y que tiene como únicas respuestas posibles SI – NO (o VERDADERO – FALSO)

Ejemplos:



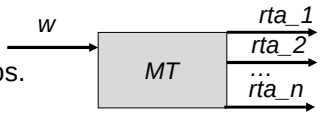
- Dado un número natural, decidir si es par o no.

- Dada una cadena arbitraria y un lenguaje regular, determinar si la cadena pertenece o no al lenguaje.

- Problema de Salida General:

Las posibles respuestas son respuestas “generales”.

Ejemplos:



- Calcular el promedio de una lista de números.

- Determinar un camino entre dos ciudades.

Problemas

- Los Problemas de Decisión se clasifican en:

DECIDIBLES: existe un ALGORITMO que para TODA INSTANCIA del problema devuelve la RESPUESTA correcta.

INDECIDIBLES: existe un PROCEDIMIENTO que sólo da RESPUESTA para ALGUNAS INSTANCIAS del problema.

- Los Problemas de Salida General se clasifican en:

SOLUBLES: existe un ALGORITMO que para TODA INSTANCIA del problema devuelve la RESPUESTA correcta.

INSOLUBLES: existe un PROCEDIMIENTO que sólo da RESPUESTA para ALGUNAS INSTANCIAS del problema.

Problemas

- La Teoría de Clasificación de Problemas se basa en el estudio de los Problemas de Decisión

- Todo problema de salida general con un cierto grado de dificultad, se puede transformar en un problema de decisión del mismo grado de dificultad (Reducción de problemas)

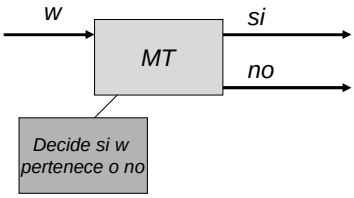
Ejemplos:

- Determinar un camino entre dos ciudades — Problema salida general
-¿Hay un camino entre dos ciudades? — Problema de decisión

Lenguajes Formales

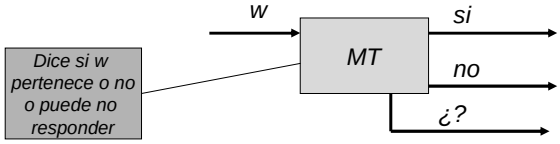
- Los lenguajes formales se clasifican en:

Lenguajes Recursivos: existe un algoritmo que determina para cualquier cadena si pertenece o no pertenece al lenguaje



Todo Lenguaje Recursivo es también Recursivo Enumerable.

Lenguajes Recursivo Enumerables: existe un procedimiento que sólo acepta las cadenas que pertenecen al lenguaje pero para las que no pertenecen no siempre da respuesta.



Lenguajes - Procedimientos

Lenguajes Recursivos ⟹ Se pueden asociar algoritmos

Lenguajes Regulares, Libres del Contexto y Sensibles al Contexto
Tipo 3 Tipo 2 Tipo 1

Lenguajes Recursivo Enumerables ⟹ Se pueden asociar procedimientos

Lenguajes Estructurados por Frases
Tipo 0

Lenguajes NO recursivo enumerables ⟹ No se pueden asociar procedimientos computacionales

Ejemplo: Lenguaje diagonal

Es importante analizar esto, para saber si al enfrentar un problema podemos construir o no un algoritmo. Y si podemos construirlo, el tiempo que tardará en dar una respuesta (complejidad del algoritmo)

Lenguajes - Problemas

Problemas		Lenguajes que los describen	
de Decisión	de Salida General		
Decidibles	Solubles	Recursivos (tipo 3, 2, 1)	Algoritmos o procedimientos efectivos
Indecidibles	Insolubles	Recursivo enumerables (tipo 0)	Procedimientos

Tesis de Turing

Las Máquinas de Turing capturan la noción de lo que es un algoritmo o un procedimiento efectivo llevado a cabo por un humano o por una máquina.

Si un problema no puede ser resuelto con una Máquina de Turing entonces no tiene solución computacional.

Todo procedimiento computacional puede ser realizado por una MT y la MT puede reconocer lenguajes de tipo 0.

Los lenguajes de tipo 0 describen todo problema que puede ser resuelto por una computadora. Es decir, pueden construirse procedimientos computacionales.

Entonces todo problema que puede ser resuelto por una computadora, puede ser descrito por un lenguaje de tipo 0 en la jerarquía.

Problemas Indecidibles: Ejemplos

Decidir si una cadena pertenece o no pertenece a un lenguaje recursivo enumerable (tipo 0).

Dada una MT M y una cadena w, decidir si M se detiene con la cadena w. Se puede describir con el Lenguaje Universal
Lu = {<M, w>/ M acepta w} (es un lenguaje recursivo enumerable)

Decidir si una gramática libre del contexto es ambigua o no.

Dado un programa, y un conjunto de datos de entrada podemos decidir si ese programa termina o no su ejecución (Problema del Halting)

Problema de Correspondencia Post

Problema de Correspondencia Post

Dadas dos listas A y B de cadenas sobre un alfabeto, determinar si existe una concatenación de cadenas de A y B de la forma

$x_{i1}\ x_{i2} \ldots x_{im} = w_{i1}\ w_{i2} \ldots w_{im}$ donde $x_{ij} \in A$ y $w_{ij} \in B$

(si elijo el indice i para x elijo el mismo indice i para w)

Ejemplo 1:

Lista A	Lista B
X_i	W_i
1	111
10111	10
10	0

Por ejemplo:

$x_2\ x_1\ x_1\ x_3$

X_i 101111110

W_i 101111110

$w_2\ w_1\ w_1\ w_3$

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

Para esta entrada, si tomamos los indices 2, 1, 1, 3
se obtiene una solución al problema

Problema de Correspondencia Post

Ejemplo 2:

Lista A	Lista B
Xi	Wi
10	101
011	11
101	011

$x_1\ x_3\ x_1$

1010110

101011101

$w_1\ w_3\ w_1$

$x_1\ x_3\ x_3\ x_3$

10101101101

101011011011

$w_1\ w_3\ w_3\ w_3$

- Se debe comenzar con la primera cadena de ambas listas
- Después la única elección posible es la tercera cadena, pero siempre queda un “uno de mas” en la cadena armada a partir de la lista B.
- Se puede seguir infinitamente buscando si existe solución

Para algunas instancias de las listas A y B se puede obtener una respuesta
Para otras instancias de las listas A y B algunas veces no se puede dar respuesta

Problema de Correspondencia Post es indecidible

Problema del Halting

No existe un algoritmo que pueda decidir correctamente si un programa arbitrario terminará su ejecución o no.

⬇

Es un problema indecidible

Demostración (por contradicción)

Suponemos que existe una función booleana HALT escrita en Pascal que resuelve el problema del Halting.

La función HALT cumple las siguientes condiciones:
- Analiza el código ubicado en un archivo llamado datos.p
- Está escrita en Pascal
- Termina después de un tiempo finito (es un algoritmo)
- Da una respuesta correcta al problema del Halting: True si el programa en datos.p termina y False en caso contrario

Problema del Halting

```
Supongamos el siguiente código
program Test
  var siempre:Boolean
  function Halt: Boolean
                              { analiza el código en datos.p }
begin
Siempre:=FALSE
  if Halt then
        repeat
           siempre:=FALSE
        until siempre
  else
    writeln “El programa termina”
end
```

¿Qué sucede si en datos.p ponemos una copia del código de TEST?
Como HALT es una función booleana, hay que analizar dos casos:

Problema del Halting

1) HALT retorna TRUE
   a) Analizando el código de HALT, el programa en datos.p termina, es decir TEST TERMINA
   b) Analizando el código de TEST, si HALT es TRUE, TEST entra en un ciclo infinito, entonces TEST NO TERMINA
a) y b) dicen que TEST termina y TEST no termina CONTRADICCION

2) HALT retorna FALSE
   a) Analizando el código de HALT, el programa en datos.p no termina, es decir TEST NO TERMINA
   b) Analizando el código de TEST, si HALT es FALSE, TEST escribe un mensaje y termina, entonces TEST TERMINA
a) y b) dicen que TEST no termina y TEST termina CONTRADICCION

En ambos casos se plantea una contradicción. La conclusión es que HALT no se comporta como se supuso.
Esta contradicción se alcanza independientemente del código que pueda aparecer entre begin y end de HALT, cuando HALT se usa en un programa como TEST.

Conclusión: NO existe un algoritmo para decidir si un programa termina o no su ejecución. Es un problema indecidible ya que encontramos una instancia (en este caso Test) para la cual Halt no puede dar respuesta correcta.