

Ciencias de la Computación I

Apuntes de Turing y problemas computacionales

Tipo	Lenguaje	Maquina Abstracta	Gramática
TIPO 3	Regular	Autómata Finito Det.	Regular
TIPO 2	Libre del Contexto	Aut. Pila Det y No Det.	Libre del Contexto
TIPO 1	Sensible al Contexto	Aut. Linealmente Acotado	Sensibles al contexto
TIPO 0	Estructurado por frases	Maq. Turing Det.	Estructuradas por frases o contractivas

Capacidades de los lenguajes formales:

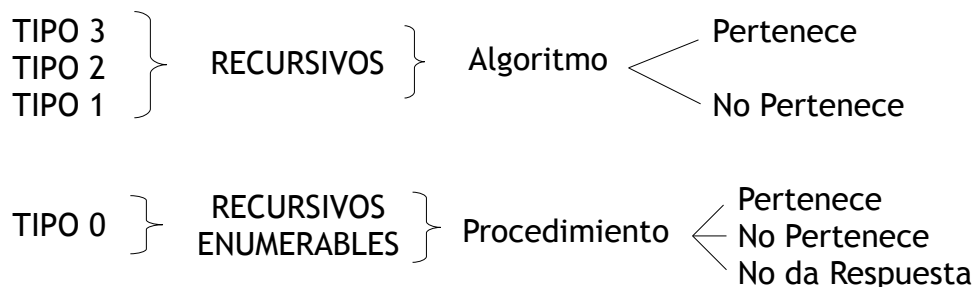
- Reconocer cadenas del lenguaje a través de las máquinas abstractas.
- Generar cadenas del lenguaje a través de las gramáticas del lenguaje.

Aclaraciones importantes:

Un *Procedimiento* da respuesta para algunas instancias de un problema.

Un *Algoritmo* da respuesta para todas las instancias del problema.

Los tipos de lenguajes 1, 2 y 3 se denominan *Lenguajes recursivos* y para ellos existen algoritmos que determinen la pertenencia y la no pertenencia de cadenas al lenguaje. El tipo 0 se denomina *recursivo enumerable* y para este, existen procedimientos que pueden determinar la pertenencia de una cadena, pero no pueden determinar la no pertenencia en todas las instancias posibles.



Un problema puede ser descrito con un Lenguaje y la solución algorítmica a dicho problema se describe con un lenguaje recursivo.

- Problemas de decisión (Si/No)
 - Decidibles: Existe un algoritmo que da respuesta para todas las instancias. (Se describen con lenguajes Recursivos)
 - Indecidibles: Existe un procedimiento que da respuesta a algunas instancias. (Se describen con lenguajes recursivo-enumerables)
- Problemas de salida general
 - Solubles: Existen algoritmos
 - Insolubles: Existen procedimientos

Sobre la tesis de Turing...

"Todo algoritmo o procedimiento efectivo es Turing-computable"

Esta es una de las versiones más aceptadas, puesto que constituye el resultado derivado de la revisión de los trabajos de A. Church y A. Turing.

En esta versión, se afirma que los algoritmos o procedimientos efectivos, elaborados tanto por un ser humano como por una máquina, pueden ser captados por una máquina de Turing.

Por tanto, en esta interpretación presentamos la Máquina de Turing como un modelo abstracto de un ordenador, puesto que es capaz de ejecutar el mismo conjunto de instrucciones que un computador.

"La clase de las funciones que pueden ser calculadas mediante un método definido coincide con la clase de las funciones calculables mediante una Máquina de Turing"

Fuente: http://es.wikibooks.org/wiki/La_tesis_de_Church-Turing

Ejemplo sobre problemas de decisión:

Problema de correspondencia de POST:

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

$$w_{i1} w_{i2} \dots w_{in} = x_{i1} x_{i2} \dots x_{in} / w_{ij} \in A \text{ y } x_{ij} \in B$$

Cadena	Lista A	Lista B
1	1	111
2	1011	10
3	10	0

Realizando distintas combinaciones entre cadenas de la lista A y cadenas de la lista B se debería poder lograr una combinación de la forma antes mencionada.

Aplicando la siguiente combinación de cadenas, se logra una primer igualdad:

A	10	111	11	10
B	10	111	11	10

Tiene solución y es: Cadena formada por las subcadenas 2 ; 1 ; 1 ; 3

En el siguiente ejemplo, no se puede alcanzar la combinación deseada, por lo tanto, se dice que no tiene solución.

Cadena	Lista A	Lista B
1	10	101
2	011	11
3	101	011

En la construcción de la cadena siguiente, se ve a modo de ejemplo que **no es posible** hallar una

solución al problema:

A	10	101	101	
B	10	101	101	1

» Se puede asegurar que el problema es **Indecidible**

Otro ejemplo de problema Indecidible:

Determinar si una GLC (Gramática Libre del Contexto) es ambigua o no.

Problema de Halting (Indecidible)

Para el problema de Halting no existe un algoritmo que pueda decidir si un programa arbitrario se detendrá o no. Su demostración se realiza por contradicción/absurdo.

- ◆ Suponemos que existe una función Halt (En nuestro caso particular, escrita en Pascal) que resuelve el problema de Halting:
 - True: Cuando el programa analizado termina.
 - False Cuando el programa analizado no termina.
- ◆ Suponemos:
 - Está escrita en Pascal
 - Da respuesta en un tiempo finito
 - Sus respuestas son True y False
 - El código que analiza “Halt” está guardado en el archivo “dato_p”.

Pseudo-Código de nuestro programa:

```
program test;
var siempre:boolean;
Function Halt: Boolean;
  Begin
  ...
  End;
begin
  siempre:= False;
  If Halt Then
    repeat
      siempre:= false
    until siempre
  else
    writeln("Este programa
termina")
end.
```

Veamos que ocurre si en *dato.p* se coloca el código del programa *test* ...

Se pueden analizar los siguientes dos casos:

1. Si Halt es TRUE, entonces, de acuerdo a lo supuesto, el programa analizado (*TEST*) termina luego, *TEST* termina. Por otro lado, mirando el código de *test*, si halt es true entra en ciclo infinito, por lo que *test* NO termina (Contradicción)
2. Si Halt es False, entonces, de acuerdo a lo supuesto, el programa analizado (*TEST*) no termina; luego, *Test* NO termina. Por otro lado, mirando el código de *test*, *test* escribe un mensaje y termina; esto significaría que Halt termina (Contradicción)

Ya que en ambos casos se producen contradicciones, se asegura que la función **HALT NO EXISTE**

Apunte realizado por:

Grupo

Como Usted ya Sabe

Apuntes, exámenes, modelos de examen,
información, noticias, chat y más!



**COMO USTED
YA SABE .COM.AR**

www.comoustedyasabe.com.ar