Teoría de la computación

BlasAST

May 2025

Índice

1. Maquina de Turing Universal

La idea base de esto es que podemos poner (codificar) una MT en la cinta de otra mostrando las transiciones de la MT realizadas mediante su alfabeto y su input.
De esta forma forma se crea la *MTU* y esta se pondrá a leer y ejecutará las intrucciones que encuentre en la cinta, las cuales describen el funcionamiento de la MT. A partir del input, también presente en la cinta la *MTU* simulará el comportamiento de la MT escribiendo los resultados parciales y finales en otra porción de la cinta.

> Otra forma de verlo es que una MTU es basicamente un emulador que simula X consola.

Este formalismo de MTU nos permite hacer maquinas que ejecuten otras maquinas. Se utiliza diariamente para:

- En el Hardware (Cargamos aplicaciones sobre nuesto S.O, incluso el propio S.O es una MT)
- Intérpretes. Capaces de aceptar programas escritos en un lenguaje de programación de alto nivel como Python.
- Emuladores y Maquinas virtuales.

2. Concepto de Algoritmo

Un algoritmo puede tener varios tipos de definiciones pero siempre tiene el mismo significado:

- **La forma más facil de entenderlo es la siguiente.** Informalmente es una secuencia de pasos finitos que resuelven un problema.
- Más formalemnte es una secuencia finita de pasos que calculan una determinada función.
- La definición más formal sería: Toda aquella computación capaz de ser realizada en una MT

3. Computabilidad y Decibilidad

Una vez entendemos que es un algoritmo podemos pasar a entender que es **su computabilidad y decibilidad.**

Un algoritmo es computable cuando para cualquier input podemos obtener su output. Es decir, es un algoritmo 'resolvible' que resuelve dado que mediante el input que reciba y una cantidad finita de pasos termina y devuelve el resultado del computo.

Un algoritmo es decidible si con el input que recibe es capaz de determinar si dicho input pertenece o no al lenguaje. Devuelve respuestas binarias (accept, reject). **Si un algorimto es decidible es siempre computable.**

Para saber si es decidible la MT que reconoce dicho lenguaje del algoritmo debe ser capaz de terminar el programa aceptando todos los inputs, en caso contrario se rechaza la palabra.

Por último un algoritmo es semidecidible cuando hay una MT que decide.
Termina siempre que se acepta la palabra del input y si no la acepta, es decir, el input no pertenece al lenguaje, puede rechazar la palabra o no terminar nunca.

4. Problema de la parada (HALTING PROBLEM)

Supongamos que tenemos una MTU que nos puede devolver 2 cosas. Si la MTU termina nos devuelve True mientras que si la MTU no termina nos devuelve False. Pues bien, esta MTU tiene un problema a pesar de estar bien definida siguiendo las reglas de la definición de MT.

$$\text{Una MTU sería asi} \begin{cases} \text{true si MT(x) termina} \\ \text{false si MT(x)} \uparrow \text{ (no termina)} \end{cases}$$

Pero, si tratamos de construir una MTU que sea capaz de decirnos **siempre** si **cualquier** MT termina o no. Podemos darnos cuenta de que no es posible crear este tipo de MT.
Para explicarlo mejor se puede encontrar un contraejemplo de dicha MTU. Suponemos que esta MTU existe, lo que quiere decir por contraejemplo que lo contrario es verdadero y por ello también supondremos que no existe una MTU que cumpla esta condición.

En lógica se expresaría de la siguiente forma:

$Termina(MTU) \leftrightarrow \neg Termina(MTU)$

> Imaginemos que tenemos una MTU que termina cuando MT no termina y no termina cuando MT termina.

Ojo, true o false es el valor final, internamente termina cuando no termina y no termina cuando termina.

$$\text{MTU(MT,x).} \begin{cases} false \text{ si MT(x)} \uparrow \text{ (no termina)} \\ true \text{ si MT(x) termina} \end{cases}$$

Pues si ahora le pasamos a ella misma como MT tendriamos lo siguiente

$$\text{MTU(MTU,x)} \begin{cases} false \text{ si MTU(X)} \uparrow \text{ (no termina)} \\ true \text{ si MTU(X) termina} \end{cases}$$

El resultado de esto sería una contradicción dado que al aplicarse lo contrario del resultado internamente terminaria cuando no termina o terminaria cuando no termina lo cual entraría en un bucle.

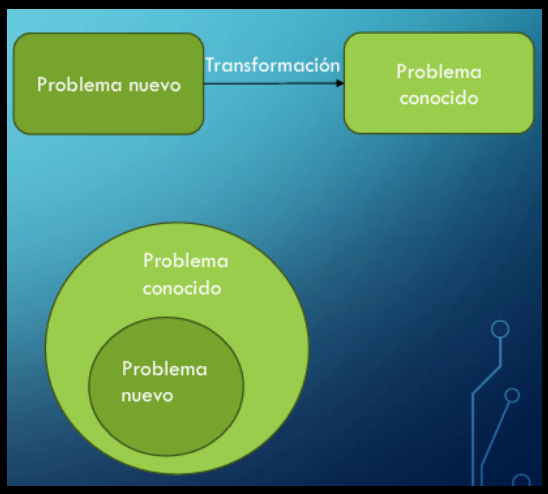
Debido a esto la conclusión es que no existe una MT capaz de decirno si cualquier MT (incluso ella misma) termina o no.

Lo que si podemos hacer son demostraciones a mano o con una MT para demostrar que otra MT concreta termina o no. Esto es parte del estudio del área de Verificación de Software.

5. Reducibilidad

Para resolver un problema a veces puede ser útil simplificarlo en otros problemas más pequeños que sepamos resolver.

Esta idea se basa en tratar de demostrar que el problema que queremos resolver es en realidad otro caso en particular de otro problema que ya sabemos como resolverlo. $P_n \subseteq P_c$



Generalmente es necesario realizar transformaciones al nuevo problema para expresarlo en términos del problema conocido. Una forma de hacerlo es mediante la inclusión de lenguajes.

6. Inclusión de lenguajes

Una vez tenemos dos lenguajes podemos ver si un lenguaje esta contenido en otro. Esto se resuelve normalmente haciendo el complemento del lenguaje más general y haciendo la intersección de ambos y posteriormente si la intersección de ambas es vacía.

Lo anterior como formula sería:

$\mathcal{L}(P_n) \subseteq \mathcal{L}(P_c) \leftrightarrow \overline{\mathcal{L}(P_c)} \cap \mathcal{L}(P_n) = \emptyset$

7. Otros formalismos

Existen otros formalismos de computación equivalentes en poder expresivo al igual que las MT. Algunas son:

- Cálculo Lambda
- Lenguajes Formales
- Funciones Parcialmente Recursivas
- Variaciones de MT: Varias cintas, cintas bidimensionales, etc.
- Autómatas finitos con dos pilas o con dos contadores
- Maquina de post
- Autómatas celulares
- El juego de la Vida de John Conway
- Computador cuántico
- Lenguajes de programación modernos si tuvieran memoria ilimitada.

8. Tesis de Church-Turing

La tesis de Church Turing establece que una MT es capaz de computar/ejecutar todo aquiello a lo que se le puede llamar algoritmo. Pasa lo mismo con el Cálculo Lambda creado por Church, siendo igual de expresivas que las MTs y por ello se hace la tesis de que ambas abarcan todo lo que es efectivamente un algoritmo en su definición más amplia.

Dado a que todavía no ha sido demostrado sigue siendo una tesis porque el concepto de algoritmo no pertenece a ninguna clase mátematica y no se puede realizar la demostración.
Casi todos los matemáticos e informáticos lo dan por verdadero y consideran que es la frontera entre lo computable(algoritmo) y lo no computable (todo aquello que ni una MT puede calcular)

Todos los formalismos equivalentes a una MT cumplen:

- Realizan una cantidad finita de trabajo en cada paso.
- Acceso aleatorio (a no ser que se usen X operaciones) a memoria finita.