Note sulla Computabilità

MT, Macchine a Doppia Pila, Macchine a Coda… = **Formalismi Massimi** (Tutti equivalenti tra loro). Nessun formalismo è più potente dei formalismi massimi

Problemi matematicamente formalizzati (e.g. risoluzione di sistemi lineari) può essere formalizzato come funzione $f : \mathbb{N} \to \mathbb{N}$ (In particolare $\mathbb{N} \leftrightarrow V^*$)

Ogni MT è equivalente a una MT con un alfabeto di 2 caratteri $\Rightarrow$ Non ci interessa la formalizzazione del problema per le MT (Per gli altri automi questo non vale!)
Conseguenza: L'insieme dei problemi risolvibili da MT non dipende dall'alfabeto.

Data una MT si può sempre scrivere un programma in un linguaggio di alto livello che la simula. (Ignorando vincoli di memoria limitata)
Viceversa dato un programma si può sempre progettare una MT che esegue il programma.

Tesi di Church (I Formulazione): Non esiste alcun formalismo per modellare una determinata computazione meccanica che sia più potente delle MT e dei formalismi ad esse equivalenti

In altre parole: non esistono formalismi più potenti di MT:

- Un problema risolto da MT => risolto anche da qualsiasi modello matematico equivalente alle MT
- Problema non risolvibile da MT => Non esiste un modello matematico che lo può risolvere

Algoritmo: Procedura di risoluzione automatica di problemi. Metodo astratto per descrivere programmi indipendentemente da codifiche specifiche.

Caratteristiche:

- Sequenza **finita** di istruzioni
- Deve essere eseguibile (immediatamente) da un processore meccanico di calcolo
- Il processore ha una memoria dati
- La computazione è discreta, codificata discretamente e procede a passi discreti
- Deterministico
- Input e output infiniti
- Memoria illimitata
- Non esiste un limite al numero di passi richiesti per effettuare un calcolo: possono esserci computazioni infinite

Tesi di Church (II Formulazione): Ogni algoritmo può essere codificato in termini di una MT (o formalismo equivalente)

Computabilità: Ogni funzione (o problema) per cui esiste una Mt che la calcoli (o risolva) si dice **computabile**, o **calcolabile**, o **risolvibile**.

Decidibilità: Un problema risolvibile la cui risposta è booleana e tale risposta esiste per ogni valore del dominio si dice **decidibile**. (In altre parole, il problema è formalizzato da una funzione calcolabile e totale)

Enumerazione Algoritmica: Dato un insieme S, S può essere **enumerato algoritmicamente** se è possibile stabilire una biiezione (detta **Godelizzazione**) fra S e $\mathbb{N}$ calcolabile attraverso un algoritmo (equivalentemente una MT).
e.g. il monoide libero di un alfabeto è enumerabile
e.g. L'insieme delle macchine di turing su un alfabeto A è enumerabile
Nota: ovviamente non è possibile in un tempo finito enumerare tutte le macchine, l'importante è poter calcolare, dato un elemento il suo numero (detto **numero di godel**) e viceversa.
Conseguenza: Tutte le funzioni computabili sono enumerabili, tutti i linguaggi riconoscibili sono enumerabili.
Nota: Le enumerazioni delle funzioni computabili e dei linguaggi riconoscibili non sono biiettive: esistono infinite MT che calcolano la stessa funzione o che riconoscono lo stesso linguaggio.

Esiste e si può costruire una MT in grado di calcolare $g(x, y) = f_y(x)$ per ogni y, x. Tale MT è detta **macchina di Turing universale**.

Il numero di funzioni $f : \mathbb{N} \to \mathbb{N}$ è $2^{\aleph_0}$ che non è numerabile (cardinalità del continuo). Ci sono più pproblemi da risolvere che modi di risolverli. Però il numero di funzioni **definibili** (o **denotabili**), cioè per cui esiste una codifica mediante un alfabeto che descrive il problema
Però l'insieme dei problemi definibili contiene strettamente l'insieme dei problemi risolvibili, per esempio l'**Halting Problem**

Teoremi Importanti e Conseguenze

Halting Problem: Nessuna MT può calcolare la funzione totale $g : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ definita come:

$$g(x, y) = \begin{cases} 1 & f_y(x) \neq \perp \\ 0 & f_y(x) = \perp \end{cases}$$

i.e. Nessuna MT può decidere a priori se una MT (leggasi programma) si fermerà (non entrerà "in loop") per un dato valore di ingresso.

Dimostrazione: Supponiamo che g sia computabile. Allora la funzione h definita come:

$$h(x) = \begin{cases} 1 & g(x, x) = 0 \\ \perp & \text{otherwise} \end{cases}$$

è anch'essa computabile. Allora esiste $i \in \mathbb{N}$ t.c. $f_i = h$. Ma allora se calcoliamo $h(i)$ abbiamo due casi:

- $h(i) = f_i(i) = 1$, ma allora $g(i,i) = 0$, che porta a $f_i(i) = \perp$ che è un assurdo
- $h(i) = f_i(i) = \perp$, ma allora $g(i,i) = 1$, che porta a $f_i(i) \neq \perp$ che è ancora un assurdo

Quindi h non può essere computabile e di conseguenza non può essere computabile nemmeno g.

Lemma di HP: Nessuna MT può calcolare la funzione

$$k(x) = \begin{cases} 1 & f_x(x) \neq \perp \\ 0 & \text{otherwise} \end{cases}$$

Nota: è un lemma dell'Halting Problem, cioè è utilizzato nella dimostrazione di HP.

Dimostrazione: Supponiamo che k sia computable. Allora la funzione h definita come:

$$h(x) = \begin{cases} 1 & k(x) = 0 \\ \perp & \text{otherwise} \end{cases}$$

è anch'essa computabile. Allora esiste $i \in \mathbb{N}$ t.c. $f_i = h$. Ma allora se calcoliamo $h(i)$ abbiamo due casi:

- $h(i) = f_i(i) = 1$, ma allora $k(i) = 0$ che porta a $f_i(i) = \perp$ che è un assurdo
- $h(i) = f_i(i) = \perp$, ma allora $k(i) = 1$ che porta a $f_i(i) \neq \perp$, che è un assurdo

Quindi h non può essere computabile e di conseguenza non può essere computabile nemmeno g.

In generale:

- Se un problema è irrisolvibile (AKA Non computabile) una sua generalizzazione è sicuramente irrisolvibile
- Però una sua specifizzazione può essere risolvibile
- Se un problema è risolvibile una sua generalizzazione può essere irrisolvibile
- Però una specifizzazione rimane risolvibile

Calcolo di Funzioni Totali: Nessuna MT pul calcolare la funzione:

$$k(x) = \begin{cases} 1 & \forall x f_y(x) \neq \perp \\ 0 & \text{otherwise} \end{cases}$$

i.e. Nessuna MT può verificare se una funzione fissata è totale o meno

Dimostrazione: Supponiamo che k sia computabile, allora la biiezione $g : \mathbb{N} \to \mathbb{N}$ definita come $g(x) = w$, dove w indica il numero di Godel di una MT che calcola una funzione totale, è anch'essa computabile.

Poichè il numero di MT totali è infinito $g(x)$ è anche totale.

Consideriamo poi la funzione definita come $h(x) = f_{g(x)}(x) + 1 = f_w(x) + 1$.

- $f_{g(x)}$ è una funzione computabile e totale, quindi anche h sarà computabile e totale
- Poichè g è totale e strettamente monotona ($x' > x \Rightarrow g(x') > g(x)$) allora esiste g^{-1} computabile (perchè g è computabile)

Poichè h è computabile e totale, sia w_0 l'indice di una macchina di turing tale per cui $f_{w_0} = h$. Inoltre, essendo l'indice di una MT che calcola una funzione totale esiste x_0 t.c. $g(x_0) = w_0$ e viceversa $g^{-1}(w_0) = x_0$.

Osserviamo che $h(x_0) = f_{g(x_0)}(x_0) + 1 = f_{w_0}(x_0) + 1$. Però abbiamo anche detto che $f_{w_0} = h$ quindi si trova un assurdo.

IMPORTANTE: Un problema può essere irrisolvibile solamente quando bisogna prendere in considerazione infiniti casi possibili (per esempio infiniti possibili argomenti di funzione) ma non di fronte al calcolo di una funzione costante. Una funzione costante $f_{yes}(x) = 1, \forall x$ o la funzione costante $f_{no}(x) = 0, \forall x$ sono banalmente computabili e quindi il problema di decisione la cui funzione può essere una tra le due funzioni costanti è **sicuramente decidibile**.

IMPORTANTE: Se quindi un problema è formalizzabile attraverso una funzione con dominio finito allora **il problema è certamente decidibile**, infatti basta calcolare ogni valore per il dominio finito.

Decidibilità di Insiemi

Dato un insieme S, si dice **funzione caratteristica** $c_S : \mathbb{N} \to \{0, 1\}$ la funzione definita come:

$$c_S(x) = \begin{cases} 1 & x \in S \\ 0 & x \notin S \end{cases}$$

Un insieme S è detto **ricorsivo** o **decidibile** se e solo se la sua funzione caratteristica è computabile.

Se un insieme è decidibile il problema dell'appartenenza di un elemento a tale insieme può essere risolto meccanicamente mediante una MT che implementa la funzione caratteristica.

Nota: La funzione c_S è totale.

Un insieme S è detto **ricorsivamente enumerabile** o **semidecidibile** se e solo se è l'insieme vuoto ($S = \emptyset$) oppure è l'immagine di una funzione totale e computabile g_S, cioè

$$S = I_{g_S} = \{g_S(0), g_S(1), ...\} = \{x | \exists x, y \in \mathbb{N} \wedge x = g_S(y)\}$$

Per un insieme ricorsivamente enumerabile si possono, per l'appunto, enumerare tutti gli elementi dell'insieme. Se vogliamo controllare l'appartenenza di un elemento generico x all'insieme S potremmo enumerare tutti gli elementi e verificare se x è uguale a uno di questi elementi. Se lo è, allora la computazione si ferma, altrimenti non si può concludere nè che $x \in S$ nè che $x \notin S$

Teorema Importante: Se S è ricorsivo, è anche RE. Inoltre S è ricorsivo se e solo se S e $\bar{S}$ sono RE.

Nota: Per la definizione degli insiemi ricorsivamente enumerabili la cardinalità degli insiemi RE è anch'essa $\aleph_0$ (Cardinalità delle funzioni computabili).

Nota: Conseguenza di ciò è che la cardinalità degli insiemi RE è minore della cardinalità di tutti i sottoinsiemi di $\mathbb{N}$ (Essendo l'insieme delle parti di $\mathbb{N}$)

Teorema: Per ogni S se

- $i \in S$ implica che f_i sia totale
- Se f è totale allora $\exists i \in S$ t.c. $f_i = f$

allora S non è RE.

Questo implica che:

- Non è possibile enumerare ricorsivamente tutte (e sole) le funzioni totali computabili
- Non è possibile enumerare ricorsivamente nemmeno l'insieme di tutte le funzioni totali (Che tra l'altro è l'insieme massimale che soddisfa le ipotesi del teorema)

NOTA: Funzione totale $\neq$ MT totale! Una macchina di Turing che ha tutte transizioni definite può calcolare anche una funzione parziale

Non Estensibilità di Funzioni Parziali Non esiste una funzione totale e computabile h che sia un'estensione della seguente funzione:

$$g(x) = \begin{cases} f_x(x) + 1 & f_x(x) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

In altre parole questo è un esempio di funzione parziale che non può essere estesa ad una funzione totale, in quanto risulterebbe in una funzione totale ma non computabile.

Condizioni per insiemi RE: Un insieme S è RE se e solo se

- $S = D_h$ per qualche funzione parziale computabile h (i.e. $S = \{x | h(x) \neq \perp\}$)
- $S = I_g$ per qualche funzione parziale computabile g

Da questo teorema si può ricavare che ogni a ogni insieme semidecidibile (o RE) corrisponde una funzione **semicaratteristica** per cui:

$$c_S(x) = \begin{cases} 1 & x \in S \\ \perp & \text{otherwise} \end{cases}$$

Conseguenza: $K = \{x | f(x) \neq \perp\}$ è un insieme semidecidibile che non è semidecidibile. Infatti $h(x) = f_x(x)$ è computabile ed il suo dominio è K. D'altra parte la funzione caratteristica:

$$c_S(x) = \begin{cases} 1 & f_x(x) \neq \perp \\ 0 & \text{otherwise} \end{cases}$$

è non computabile (Per il lemma dell'HP)

Teoremi sulla Decidibilità

Teorema di Kleene (Del punto fisso): Sia t una funzione totale e computabile. Allora esiste p tale che $f_p = f_{t(p)}$.

f_p è detto punto fisso di t.

Teorema di Rice: Sia F un insieme qualunque di funzioni computabili. L'insieme S (n.d.r. Infinito!) definito come $S = \{x | f_x \in F\}$ degli indici di MT che calcolano le funzioni di F è decidibile se e solo se $F = \emptyset$ oppure se F è l'insieme di tutte le funzioni computabili.

Conseguenza del teorema di Rice: non è possibile decidere se una MT può calcolare una funzione fissata g in modo algoritmico e non si può verificare se due programmi siano equivalenti. In generale non si può stabilire se un algoritmo gode di una proprietà non banale.

Note Importanti:

I SEGUENTI PROBLEMI SONO NOTORiAMENTE **INCOMPUTABILI**

Già listati precedentemente:

1. Halting Problem (Una generica TM si ferma per un generico input?)
2. Verificare se una data funzione è totale

I SEGUENTI PROBLEMI SONO NOTORIAMENTE **INDECIDIBILI**

Per le grammatiche **context-free**:

1. “Emptyness of Intersection”: Determinare se l’intersezione dei linguaggi generati da due grammatiche è vuoto (o in generale un linguaggio stesso) (Per i linguaggi regolari questo problema è decidibile)
2. “Grammar-class-membership”: Determinare, data una grammatica G ed un insieme di grammatiche Γ se $G \in \Gamma$
3. “Language-Class-Membership”: Determinare data una grammatica G ed un insieme di linguaggi $\mathscr{L}$ se $L(G) \in \mathscr{L}$

Per le macchine di Turing:

1. “L’Alacre Castoro”: determinare la MT in un insieme di MT con lo stesso numero di stati che effettua il maggior numero di passi di computazioni, alla fine terminando
2. Verificare se una MT accetta una stringa vuota

I SEGUENTI PROBLEMI SONO NOTORIAMENTE **DECIDIBILI**

1. Emptyness di un linguaggio regolare
2. Se si ha la chiusura rispetto al complemento e all’intersezione e, inoltre, si ha la decidibilità dell’emptyness di un linguaggio allora è decidibile verificare se due linguaggi sono uguali.
3. Equivalenza di due polinomi

Schema Riassuntivo dei Problemi di Decidibilità su Linguaggi

Classe di linguaggio \ Problema	$x \in L$	$L = \varnothing$	$L_1 = L_2$	$L_1 \cap L_2 = \varnothing$
Accettati da MT	I	I	I	I
Regolari	D	D	D	D
Liberi da contesto	D*	D*	I*	I
Accettati da AP	D	D*	D**[6]	I

D: decidibile
I: indecidibile

Note Varie

- Il problema di verificare se una formula di FOL è **soddisfacibile** è un problema sicuramente decidibile anche se la formula è aperta. Inoltre, se fattibile, si può provare a vedere se esiste un assegnamento per cui è vera e quindi la si può rendere effettivamente decisa.
- Ogni grammatica genera un linguaggio ricorsivamente enumerabile: questo è ovvio se si prende in considerazione la MT che da quella grammatica genera stringhe del linguaggio. La funzione che è calcolata dalla MT ha come immagine l’insieme di tutte le stringhe del linguaggio, quindi per definizione è ricorsivamente enumerabile