Lezione di Informatica Teorica: Macchina di Turing Universale e Linguaggi Indecidibili

Appunti da Trascrizione Automatica

30 giugno 2025

Indice

1 Introduzione e Ripasso

Nella lezione precedente abbiamo introdotto le Macchine di Turing Non-Deterministiche (NTM) e le classi di problemi decidibili (R) e semi-decidibili (RE).

- La simulazione di una **NTM** da parte di una **DTM** è possibile, sebbene con un costo significativo in termini di tempo di esecuzione (esponenziale o doppio esponenziale). Ai fini della sola *decidibilità* di un problema, l'efficienza temporale non è un fattore determinante.
- La classe **R** include tutti i problemi **decidibili**, cioè per i quali esiste un algoritmo (una macchina di Turing) che termina sempre, fornendo una risposta definita (sì/no).
- La classe **RE** include i problemi **semi-decidibili**. Per questi, esiste una macchina di Turing che garantisce una risposta "sì" in tempo finito se la stringa appartiene al linguaggio; tuttavia, se la stringa non appartiene al linguaggio, la macchina potrebbe non terminare mai. Questo li rende, di fatto, non algoritmicamente utili per il "no".
- Esistono problemi completamente **indecidibili**, che non rientrano né in R né in RE. Per questi non è garantita alcuna risposta.

2 La Macchina di Turing Universale (UTM)

Le Macchine di Turing (MT) studiate finora sono intrinsecamente fisse nel loro comportamento: la loro funzione di transizione è immutabile e determinano un unico compito. I computer moderni, al contrario, sono programmabili, ovvero possono eseguire programmi diversi. Per colmare questa apparente discrepanza, introduciamo il concetto di **Macchina di Turing Universale (UTM)**.

Definizione 1 (Macchina di Turing Universale (UTM))**.** *Una Macchina di Turing Universale (M_U) è una macchina di Turing che è in grado di* ***simulare il comportamento di qualsiasi altra Macchina di Turing*** *(M) quando le viene fornita la descrizione (codifica) di M e l'input w su cui M dovrebbe operare.*

2.1 Codifica di una Macchina di Turing (MT)

Per consentire a una UTM di simulare un'altra MT, il "programma" della MT (la sua funzione di transizione) deve essere codificato in una stringa binaria. Assumiamo, per semplicità, che le MT considerate abbiano un alfabeto di input binario $\Sigma = \{0, 1\}$.

Una generica istruzione della funzione di transizione δ ha la forma:

$$\delta(Q_i, X_j) = (Q_k, X_L, D_m)$$

dove:

- Q_i: stato corrente.
- X_j: simbolo letto sul nastro.
- Q_k: nuovo stato dopo la transizione.
- X_L: simbolo da scrivere sul nastro.
- D_m: direzione del movimento della testina (sinistra o destra).

La codifica segue queste convenzioni:

1. **Stati (Q_i)**: Ogni stato Q_i è codificato con una sequenza di i zeri.
   - Q_1 è lo stato iniziale, codificato come 0.
   - Q_2 è lo stato accettante, codificato come 00.
   - Esempio: Q_5 è codificato come 00000.
2. **Simboli di Nastro (X_j)**: Ogni simbolo X_j è codificato con una sequenza di j zeri.
   - X_1 è il simbolo 0, codificato come 0.
   - X_2 è il simbolo 1, codificato come 00.
   - X_3 è il simbolo blank (ϕ), codificato come 000.
3. **Direzioni (D_m)**:
   - D_1 (sinistra, L) è codificato come 0.
   - D_2 (destra, R) è codificato come 00.
4. **Separatori**:
   - Tra i componenti di una singola transizione (es. Q_i e X_j), si usa il simbolo 1.
   - Tra diverse transizioni (coppie $\delta(\dots)$), si usa la sequenza 11.

Esempio 1. *Codifica della transizione:* $\delta(Q_3, \phi) = (Q_5, 0, \rightarrow)$

- $Q_3 \rightarrow$ ***000***
- ϕ *(*X_3*)* $\rightarrow$ ***000***
- $Q_5 \rightarrow$ ***00000***
- 0 *(*X_1*)* $\rightarrow$ ***0***
- $\rightarrow$ *(*D_2*)* $\rightarrow$ ***00***

La codifica di questa singola transizione sarà: ***0001000100000100***

La codifica completa di una MT è la concatenazione di tutte le sue transizioni, separate da 11*. Ad esempio:* *Codifica*(δ_1)***11***δ_2***11***$\delta_3 \dots$

Osservazione 1. *Non tutte le stringhe binarie corrisponderanno a una MT validamente codificata. Per convenzione, le stringhe che non seguono il formato di codifica (es. quelle con* 111 *o* 1111*, o quelle non interpretabili come programma) sono considerate codifiche di MT che rifiutano qualsiasi input (il cui linguaggio è vuoto).*

2.2 Architettura e Funzionamento della UTM

La UTM (M_U) è tipicamente una Macchina di Turing a quattro nastri. Riceve come input una stringa che è la concatenazione della codifica di una MT M e di una stringa w, separate da una sequenza di 1 (es. 111 per distinguere dalla separazione delle transizioni). L'obiettivo di M_U è simulare il comportamento di M su w.

I quattro nastri di M_U hanno i seguenti ruoli:

1. **Nastro 1 (Input/Programma M)**: Contiene la stringa di codifica di M. Questo nastro è letto dalla M_U per conoscere le regole di transizione di M e non viene modificato durante la simulazione.
2. **Nastro 2 (Nastro Simulato di M)**: Contiene la rappresentazione del nastro della MT M. Inizialmente, l'input w viene copiato su questo nastro e ricodificato nel formato dei simboli X_j (es. 0 diventa 0, 1 diventa 00, etc.). La testina della M_U su questo nastro simula la posizione della testina di M.
3. **Nastro 3 (Stato Simulato di M)**: Contiene la codifica dello stato corrente in cui si trova la MT M nella simulazione. Inizia con lo stato iniziale di M (Q_1).
4. **Nastro 4 (Scratchpad/Ausiliario)**: Utilizzato per operazioni temporanee, come la gestione dello spazio sul Nastro 2 quando un simbolo deve essere riscritto e la sua codifica occupa più o meno spazio.

2.2.1 Passi di Simulazione

M_U esegue la simulazione passo per passo:

1. M_U legge lo stato corrente di M (dal Nastro 3) e il simbolo sotto la testina di M (dal Nastro 2).
2. Usando questa coppia (stato, simbolo), M_U cerca sul Nastro 1 (il "programma" di M) la transizione $\delta(Q_i, X_j) = (Q_k, X_L, D_m)$ corrispondente.
3. Una volta trovata la transizione, M_U aggiorna la simulazione:
   - Aggiorna lo stato sul Nastro 3 a Q_k.
   - Scrive il simbolo X_L sul Nastro 2 (spostando il contenuto se necessario, usando il Nastro 4).
   - Sposta la testina sul Nastro 2 nella direzione D_m.
4. M_U ripete questi passi. Se M entra in uno stato di accettazione (la M_U lo rileva leggendo il Nastro 3), allora M_U accetta. Se M si blocca in uno stato non accettante, M_U si blocca. Se M entra in un loop infinito, M_U entrerà anch'essa in un loop infinito.

In questo modo, M_U replica fedelmente il comportamento di M su w.

2.3 Connessione con i Computer Reali

Un computer che utilizziamo quotidianamente è l'equivalente di una Macchina di Turing Universale. Il suo "programma" fisso e immutabile non è l'applicazione che lanciamo, ma il **microcodice della CPU**. La CPU esegue ciclicamente un programma interno fisso (il microcodice) che è progettato per *interpretare* ed *eseguire* le istruzioni dei programmi che gli vengono dati in input (le

nostre applicazioni). Questo risolve il dilemma: i computer non "cambiano" programma, ma hanno un programma fisso (il microcodice) che li rende universali, capaci di simulare qualsiasi altro programma.

3 Linguaggi Indecidibili: L'Argomento di Diagonalizzazione

Abbiamo affermato l'esistenza di problemi completamente indecidibili (non in RE). Una delle dimostrazioni più celebri di ciò è l'argomento di diagonalizzazione di Cantor.

3.1 Enumerazione di Stringhe e Macchine di Turing

1. **Enumerazione di stringhe binarie (Σ^*)**: L'insieme di tutte le stringhe binarie finite (Σ^*, dove $\Sigma = \{0, 1\}$) è **numerabile**. Possiamo ordinarle per lunghezza crescente e, a parità di lunghezza, lessicograficamente. Questo ci permette di parlare della 1^a stringa, 2^a stringa, ecc. (es., $\epsilon, 0, 1, 00, 01, 10, 11, \dots$). La cardinalità di Σ^* è $\aleph_0$ (aleph-zero), la stessa dei numeri naturali.
2. **Enumerazione delle Macchine di Turing (MT)**: Poiché ogni MT può essere codificata come una stringa binaria finita, e le stringhe binarie sono numerabili, anche l'insieme di tutte le Macchine di Turing è **numerabile**. Possiamo quindi parlare della 1^a MT (M_1), della 2^a MT (M_2), ecc.

3.2 Vettore Caratteristico di un Linguaggio

Definizione 2 (Vettore Caratteristico (Q_L))**.** *Per un linguaggio L su un alfabeto Σ, il suo **vettore caratteristico** Q_L è una sequenza binaria infinita. Per ogni i-esima stringa w_i (secondo l'ordine di enumerazione di Σ^*), la i-esima componente di Q_L è 1 se $w_i \in L$, e 0 se $w_i \notin L$. Questo vettore descrive in modo univoco il linguaggio L.*

3.3 Costruzione del Linguaggio di Diagonalizzazione (L_D)

Consideriamo una tabella infinita dove:

- Le **colonne** sono indicizzate dalle stringhe enumerate di Σ^*: $w_1, w_2, w_3, \dots$
- Le **righe** sono indicizzate dalle MT enumerate: $M_1, M_2, M_3, \dots$
- La cella (i, j) contiene 1 se la MT M_i **accetta** la stringa w_j, e 0 altrimenti.

Ogni riga i di questa tabella rappresenta il vettore caratteristico del linguaggio $L(M_i)$ accettato dalla macchina M_i.

Esempio di una porzione di tabella:

	w_1	w_2	w_3	w_4	$\dots$	
M_1	0	1	0	0	$\dots$	$(L(M_1))$
M_2	1	1	0	1	$\dots$	$(L(M_2))$
M_3	0	0	0	1	$\dots$	$(L(M_3))$
M_4	1	0	0	1	$\dots$	$(L(M_4))$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	

Procediamo alla costruzione del linguaggio di diagonalizzazione (L_D):

1. **Prendere la diagonale (D)**: Estrarre la sequenza dei bit sulla diagonale principale: $D = d_1d_2d_3d_4\ldots$, dove d_i è il valore della cella (i, i) (cioè se M_i accetta w_i). *Nell'esempio sopra:* $D = 0101\ldots$
2. **Complementare la diagonale ($\overline{D}$)**: Invertire ogni bit della sequenza D. $\overline{D} = \overline{d_1d_2d_3d_4}\ldots$ *Nell'esempio sopra:* $\overline{D} = 1010\ldots$
3. **Definire L_D**: Interpretiamo $\overline{D}$ come il vettore caratteristico di un nuovo linguaggio, L_D. Formalmente, $L_D = \{w_i \mid \text{l'i-esima bit di } \overline{D} \text{ è } 1\}$.

3.4 Definizione Formale di L_D

Il linguaggio L_D può essere più intuitivamente definito come:

Definizione 3 (Linguaggio di Diagonalizzazione L_D)**.** $L_D = \{\langle M_i\rangle \mid M_i \text{ è una MT e } M_i \text{ rifiuta la propria codifica } \langle M_i\rangle\}$ *dove* $\langle M_i\rangle$ *(o* w_i*) denota la stringa che codifica la MT* M_i*. In altre parole, una stringa* $\langle M_i\rangle$ *appartiene a* L_D *se e solo se la macchina* M_i *non accetta* $\langle M_i\rangle$*.*

3.5 Teorema: L_D non appartiene a RE

Teorema 1. *Il linguaggio* L_D *non appartiene alla classe RE (Linguaggi Ricorsivamente Enumerabili). In altre parole, non esiste alcuna Macchina di Turing che accetta* L_D*.*

Dimostrazione. Supponiamo per contraddizione che L_D sia ricorsivamente enumerabile. Se $L_D \in RE$, allora esisterebbe una Macchina di Turing, diciamo M_k, che accetta L_D. Poiché tutte le MT sono enumerate nella nostra tabella, M_k corrisponde a una specifica riga k in quella tabella.

Consideriamo la stringa w_k, che è la codifica $\langle M_k\rangle$ della macchina M_k. Ora poniamoci la domanda: $w_k \in L_D$ o $w_k \notin L_D$?

(i) **Caso 1: Supponiamo $w_k \in L_D$.**

- Per definizione di L_D, se $w_k \in L_D$, allora M_k deve rifiutare $\langle M_k\rangle$ (cioè w_k).
- Se M_k rifiuta w_k, allora la cella (k, k) della tabella deve essere 0.
- D'altra parte, poiché M_k accetta L_D, e abbiamo supposto $w_k \in L_D$, M_k dovrebbe accettare w_k. Ciò implicherebbe che la cella (k, k) è 1.

Abbiamo una contraddizione: la cella (k, k) non può essere contemporaneamente 0 e 1.

(ii) **Caso 2: Supponiamo $w_k \notin L_D$.**

- Per definizione di L_D, se $w_k \notin L_D$, allora M_k deve accettare $\langle M_k\rangle$ (cioè w_k).
- Se M_k accetta w_k, allora la cella (k, k) della tabella deve essere 1.
- D'altra parte, poiché M_k accetta L_D, e abbiamo supposto $w_k \notin L_D$, M_k dovrebbe rifiutare w_k. Ciò implicherebbe che la cella (k, k) è 0.

Abbiamo di nuovo una contraddizione: la cella (k, k) non può essere contemporaneamente 1 e 0.

Poiché in entrambi i casi si arriva a una contraddizione, l'assunzione iniziale che $L_D \in RE$ deve essere falsa. Pertanto, L_D non appartiene a RE. □

3.6 Intuizione sull'Esistenza di Linguaggi Indecidibili

La ragione profonda per l'esistenza di linguaggi indecidibili (e non ricorsivamente enumerabili) è una questione di **cardinalità** degli insiemi:

- L'insieme di tutte le **Macchine di Turing** è numerabile (ha cardinalità $\aleph_0$). Ogni MT riconosce esattamente un linguaggio.
- L'insieme di tutti i **linguaggi** su un alfabeto infinito (come Σ^*) è l'insieme potenza di Σ^*. Poiché Σ^* è numerabile, il suo insieme potenza ha la cardinalità del continuo ($2^{\aleph_0}$), che è strettamente maggiore di $\aleph_0$.

Questo significa che ci sono **molti più linguaggi che Macchine di Turing**. Dal momento che non ci sono abbastanza MT per riconoscere tutti i linguaggi possibili, devono esistere linguaggi per i quali non esiste alcuna MT in grado di accettarli. L_D è un esempio concreto di uno di questi linguaggi.

4 Proprietà sui Linguaggi Ricorsivi e Ricorsivamente Enumerabili

Introduciamo due importanti teoremi che collegano le classi R e RE con le operazioni di complemento.

Teorema 2 (Chiusura di R sotto Complemento)**.** *Se un linguaggio L appartiene alla classe R (è ricorsivo), allora anche il suo complemento $\overline{L}$ appartiene a R.*

Dimostrazione. Sia $L \in R$. Per definizione, esiste una MT M_L che decide L. Questo significa che per ogni stringa w data in input, M_L termina sempre, accettando w se $w \in L$ e rifiutando w se $w \notin L$.

Possiamo costruire una nuova MT $M_{\overline{L}}$ per decidere $\overline{L}$ come segue: $M_{\overline{L}}$ simula M_L su qualsiasi input w. Poiché M_L termina sempre:

- Se M_L accetta w (cioè $w \in L$), allora $M_{\overline{L}}$ modifica il suo comportamento per rifiutare w.
- Se M_L rifiuta w (cioè $w \notin L$), allora $M_{\overline{L}}$ modifica il suo comportamento per accettare w.

In pratica, $M_{\overline{L}}$ può essere costruita da M_L semplicemente scambiando gli stati di accettazione e rifiuto (o, più precisamente, rendendo accettanti gli stati che prima conducevano al rifiuto e viceversa, assicurandosi che tutte le computazioni terminino). Dato che M_L termina sempre, anche $M_{\overline{L}}$ terminerà sempre, fornendo una decisione per ogni input. Quindi, $M_{\overline{L}}$ è una macchina che decide $\overline{L}$, il che implica $\overline{L} \in R$. □

Teorema 3 (Condizione Necessaria e Sufficiente per R)**.** *Un linguaggio L appartiene alla classe R (è ricorsivo) se e solo se L appartiene alla classe RE* ***e*** *il suo complemento $\overline{L}$ appartiene anch'esso alla classe RE.*

$$L \in R \iff L \in RE \land \overline{L} \in RE$$

Dimostrazione. ($\Rightarrow$) **Se** $L \in R$**, allora** $L \in RE \land \overline{L} \in RE$**.** Se $L \in R$, allora per definizione L è decidibile. Ogni linguaggio decidibile è anche ricorsivamente enumerabile, quindi $L \in RE$. Inoltre, per il teorema precedente, se $L \in R$, allora $\overline{L} \in R$. Di conseguenza, anche $\overline{L} \in RE$.

($\Leftarrow$) **Se** $L \in RE \land \overline{L} \in RE$**, allora** $L \in R$**.** Supponiamo che $L \in RE$ e $\overline{L} \in RE$.

- Poiché $L \in RE$, esiste una Macchina di Turing M_L che accetta L (garantisce terminazione per i "sì", ma potrebbe non terminare per i "no").

- Poiché $\overline{L} \in RE$, esiste una Macchina di Turing $M_{\overline{L}}$ che accetta $\overline{L}$ (garantisce terminazione per i "sì" in $\overline{L}$, ovvero per i "no" in L).

Possiamo costruire una nuova MT $M_{decider}$ che decide L. $M_{decider}$ opera come segue su qualsiasi input w:

1. $M_{decider}$ simula M_L e $M_{\overline{L}}$ in **parallelo**. Questo può essere fatto, ad esempio, alternando i passi di esecuzione delle due macchine (un passo di M_L, poi un passo di $M_{\overline{L}}$, poi un altro passo di M_L, ecc., utilizzando un approccio multi-nastro o di interleaving).
2. Se la simulazione di M_L accetta w (ovvero M_L entra in uno stato di accettazione), allora $M_{decider}$ **accetta** w.
3. Se la simulazione di $M_{\overline{L}}$ accetta w (ovvero $M_{\overline{L}}$ entra in uno stato di accettazione), allora $M_{decider}$ **rifiuta** w.

Dato che w deve appartenere o a L o a $\overline{L}$ (ma non a entrambi), una delle due macchine (M_L o $M_{\overline{L}}$) è garantita a terminare e accettare w. Pertanto, $M_{decider}$ terminerà sempre su ogni input w, fornendo una risposta definita ("sì" o "no"). Questo dimostra che $M_{decider}$ decide L, e quindi $L \in R$. □

Questo teorema è cruciale: esso stabilisce che un problema è decidibile se e solo se sia il problema stesso che il suo complemento sono ricorsivamente enumerabili. Queste proprietà saranno fondamentali per classificare altri importanti problemi nelle prossime lezioni.