Tutorato di Automi e Linguaggi Formali

Soluzioni Homework 9: Indecidibilità e Riducibilità

Gabriel Rovesti
Corso di Laurea in Informatica – Università degli Studi di Padova

Tutorato 9 – 19-05-2025

1 Problemi Indecidibili e Diagonalizzazione

Esercizio 1. Il metodo della diagonalizzazione di Cantor è utilizzato per dimostrare l'esistenza di linguaggi non riconoscibili da macchine di Turing.

a) Spiegare formalmente perché l'insieme di tutte le macchine di Turing è numerabile. Fornire una funzione di enumerazione che associa ciascuna macchina di Turing a un numero naturale univoco.

b) Dimostrare, utilizzando il metodo della diagonalizzazione, che l'insieme di tutti i linguaggi su un alfabeto Σ è non numerabile. Spiegare chiaramente dove viene applicata la diagonalizzazione nella dimostrazione.

c) Basandosi sui risultati precedenti, spiegare perché deve esistere almeno un linguaggio che non è riconoscibile da alcuna macchina di Turing. Questa è una dimostrazione non costruttiva. Quale linguaggio specifico è stato introdotto nel corso come esempio di linguaggio non Turing-riconoscibile?

Soluzione. a) Numerabilità dell'insieme di tutte le macchine di Turing

Teorema 1. *L'insieme di tutte le macchine di Turing è numerabile.*

Proof. Per dimostrare che l'insieme di tutte le macchine di Turing è numerabile, dobbiamo costruire una funzione iniettiva dall'insieme delle macchine di Turing all'insieme dei numeri naturali.

Ricordiamo che una macchina di Turing M è formalmente definita come una 7-tupla $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, dove:

- Q è un insieme finito di stati

- Σ è l'alfabeto di input (finito)
- Γ è l'alfabeto del nastro (finito)
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $q_{accept} \in Q$ è lo stato di accettazione
- $q_{reject} \in Q$ è lo stato di rifiuto (con $q_{accept} \neq q_{reject}$)

Possiamo costruire una codifica per ogni macchina di Turing nel modo seguente:

1. **Codifica degli insiemi finiti:** Per ogni insieme finito, possiamo enumerare i suoi elementi e rappresentarli usando una codifica binaria. Ad esempio, per $Q = \{q_0, q_1, q_2, ..., q_n\}$, possiamo rappresentare ogni stato con la sua codifica binaria.
2. **Codifica degli alfabeti:** Analogamente, possiamo codificare Σ e Γ come sequenze di simboli, dove ogni simbolo viene rappresentato da un numero.
3. **Codifica della funzione di transizione:** La funzione di transizione δ può essere rappresentata come un insieme finito di quintuple $(q, \gamma, q', \gamma', D)$, dove $q, q' \in Q$, $\gamma, \gamma' \in \Gamma$ e $D \in \{L, R\}$. Ogni quintupla codifica una transizione: se la macchina è nello stato q e legge il simbolo γ, passerà allo stato q', scriverà il simbolo γ' e sposterà la testina nella direzione D.
4. **Codifica degli stati speciali:** Gli stati q_0, q_{accept} e q_{reject} possono essere codificati come indici nell'insieme Q.

Una volta che abbiamo codificato tutti questi componenti, possiamo concatenarli in una singola stringa binaria, usando opportuni delimitatori per separare le varie parti.

Più precisamente, definiamo una funzione di codifica $\text{cod} : \text{TM} \to \{0,1\}^*$ che mappa ogni macchina di Turing in una stringa binaria, e poi una funzione $f : \{0,1\}^* \to \mathbb{N}$ che mappa ogni stringa binaria in un numero naturale.

La funzione f può essere definita, ad esempio, come:

$$f(b_1 b_2 \dots b_n) = \sum_{i=1}^{n} b_i \cdot 2^{n-i}$$

dove $b_i \in \{0, 1\}$ è il i-esimo bit della stringa.

Poiché ogni macchina di Turing può essere codificata in modo univoco con una stringa binaria, e ogni stringa binaria può essere messa in corrispondenza biunivoca con un numero naturale, ne segue che l'insieme di tutte le macchine di Turing è numerabile. □

b) Non numerabilità dell'insieme di tutti i linguaggi

Teorema 2. *Dato un alfabeto finito Σ con almeno due simboli, l'insieme di tutti i linguaggi su Σ^* è non numerabile.*

Proof. Dimostriamo questo teorema utilizzando il metodo della diagonalizzazione di Cantor. Supponiamo, per assurdo, che l'insieme di tutti i linguaggi su Σ^* sia numerabile. Allora esiste una funzione biiettiva $f : \mathbb{N} \to \mathcal{P}(\Sigma^*)$, dove $\mathcal{P}(\Sigma^*)$ rappresenta l'insieme delle parti di Σ^*, cioè l'insieme di tutti i linguaggi su Σ^*.

Questo significa che possiamo enumerare tutti i linguaggi su Σ^* come $L_1, L_2, L_3, \ldots$, dove $L_i = f(i)$ per ogni $i \in \mathbb{N}$.

Ora, fissiamo un'enumerazione $w_1, w_2, w_3, \ldots$ di tutte le stringhe in Σ^*. Questa enumerazione è possibile perché Σ^* è numerabile (essendo Σ finito).

Consideriamo una tabella infinita dove le righe rappresentano i linguaggi L_i e le colonne rappresentano le stringhe w_j:

	w_1	w_2	w_3	w_4	$\ldots$
L_1	$[w_1 \in L_1]$	$[w_2 \in L_1]$	$[w_3 \in L_1]$	$[w_4 \in L_1]$	$\ldots$
L_2	$[w_1 \in L_2]$	$[w_2 \in L_2]$	$[w_3 \in L_2]$	$[w_4 \in L_2]$	$\ldots$
L_3	$[w_1 \in L_3]$	$[w_2 \in L_3]$	$[w_3 \in L_3]$	$[w_4 \in L_3]$	$\ldots$
L_4	$[w_1 \in L_4]$	$[w_2 \in L_4]$	$[w_3 \in L_4]$	$[w_4 \in L_4]$	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

dove $[w_j \in L_i]$ è 1 se $w_j \in L_i$ e 0 altrimenti.

Ora, definiamo un nuovo linguaggio L_d (il linguaggio "diagonale") come segue:

$$w_j \in L_d \Leftrightarrow w_j \notin L_j$$

ovvero, L_d contiene esattamente quelle stringhe w_j che non appartengono al corrispondente linguaggio L_j.

Questo linguaggio L_d è diverso da ogni linguaggio nella nostra enumerazione, perché:

- $L_d \neq L_1$ perché differiscono su w_1: $w_1 \in L_d \Leftrightarrow w_1 \notin L_1$
- $L_d \neq L_2$ perché differiscono su w_2: $w_2 \in L_d \Leftrightarrow w_2 \notin L_2$
- $\ldots$
- $L_d \neq L_i$ perché differiscono su w_i: $w_i \in L_d \Leftrightarrow w_i \notin L_i$
- $\ldots$

Questo è il punto in cui viene applicata la diagonalizzazione: costruiamo un nuovo linguaggio che differisce da ogni linguaggio nell'enumerazione esattamente sulla diagonale della tabella.

Pertanto, L_d è un linguaggio su Σ^* che non compare nella nostra enumerazione, contraddicendo l'ipotesi che f fosse una funzione biiettiva. Di conseguenza, l'insieme di tutti i linguaggi su Σ^* non può essere numerabile. □

c) Esistenza di linguaggi non Turing-riconoscibili

Teorema 3. *Esistono linguaggi che non sono riconoscibili da alcuna macchina di Turing.*

Proof. Abbiamo dimostrato al punto (a) che l'insieme di tutte le macchine di Turing è numerabile, cioè possiamo enumerare tutte le macchine di Turing come $M_1, M_2, M_3, \ldots$.

Abbiamo anche dimostrato al punto (b) che l'insieme di tutti i linguaggi su un alfabeto Σ è non numerabile.

Sia $\mathcal{L}_R$ l'insieme di tutti i linguaggi riconoscibili da macchine di Turing. Ogni linguaggio in $\mathcal{L}_R$ è riconosciuto da almeno una macchina di Turing nella nostra enumerazione. Pertanto, $|\mathcal{L}_R| \leq |\text{TM}|$, dove TM è l'insieme di tutte le macchine di Turing.

Poiché TM è numerabile, anche $\mathcal{L}_R$ è numerabile.

Ma l'insieme di tutti i linguaggi su Σ è non numerabile, quindi $|\mathcal{L}_R| < |\mathcal{P}(\Sigma^*)|$. Questo significa che ci sono linguaggi in $\mathcal{P}(\Sigma^*)$ che non appartengono a $\mathcal{L}_R$, cioè linguaggi che non sono riconoscibili da alcuna macchina di Turing.

Questa è una dimostrazione non costruttiva, nel senso che dimostra l'esistenza di linguaggi non Turing-riconoscibili senza esibire esplicitamente un tale linguaggio.

Un esempio specifico di linguaggio non Turing-riconoscibile introdotto nel corso è il complemento del problema dell'accettazione:

$$\overline{A_{TM}} = \{\langle M, w\rangle \mid M \text{ è una TM che non accetta la stringa } w\}$$

Questo linguaggio non è Turing-riconoscibile, come si può dimostrare utilizzando il fatto che A_{TM} è Turing-riconoscibile ma non decidibile, e il teorema che afferma che un linguaggio è decidibile se e solo se sia il linguaggio che il suo complemento sono Turing-riconoscibili. □

Esercizio 2. Consideriamo il seguente problema: data una TM M a nastro semi-infinito, determinare se esiste un input w su cui M sposta la testina a sinistra partendo dalla cella numero 2023 (ossia se in qualche momento durante la computazione la testina si muove dalla cella 2023 alla cella 2022).

a) Formulare questo problema come un linguaggio 2023_{TM}.

b) Dimostrare che il linguaggio 2023_{TM} è indecidibile mediante una riduzione da un problema noto. Specificare chiaramente la funzione di riduzione e verificare che soddisfi le proprietà necessarie.

c) Discutere se 2023_{TM} è Turing-riconoscibile, co-Turing-riconoscibile, o nessuno dei due. Giustificare la risposta.

Soluzione. a) Formulazione del problema come linguaggio 2023_{TM}

Formalizziamo il problema descritto come un linguaggio di macchine di Turing:

Definizione 1. Il linguaggio 2023_{TM} è definito come: $2023_{TM} = \{\langle M\rangle \mid M$ è una TM e $\exists w \in \Sigma^*$ tale che durante la computazione di M su w, la testina si muove dalla cella 2023 alla cella 2022$\}$

Dove $\langle M\rangle$ rappresenta la codifica della macchina di Turing M.

In altre parole, 2023_{TM} è l'insieme delle codifiche di tutte le macchine di Turing per cui esiste almeno un input che fa spostare la testina a sinistra partendo dalla cella 2023.

b) Dimostrazione dell'indecidibilità di 2023_{TM}

Teorema 4. *Il linguaggio* 2023_{TM} *è indecidibile.*

Proof. Dimostriamo l'indecidibilità di 2023_{TM} attraverso una riduzione dal problema della fermata $HALT_{TM}$.

Ricordiamo che $HALT_{TM} = \{\langle M, w\rangle \mid M \text{ è una TM che si ferma su input } w\}$ è un problema indecidibile ben noto.

Costruiamo una riduzione $f : HALT_{TM} \to 2023_{TM}$ tale che $\langle M, w\rangle \in HALT_{TM}$ se e solo se $f(\langle M, w\rangle) \in 2023_{TM}$.

Data una coppia $\langle M, w\rangle$, definiamo $f(\langle M, w\rangle) = \langle M'\rangle$ dove M' è una nuova macchina di Turing che opera come segue:

1. M' inizia con la testina sulla cella 0.
2. M' scrive l'input w sul nastro a partire dalla cella 0.
3. M' sposta la testina alla cella 0.
4. M' simula la macchina M su input w.
5. Se M si ferma su w (accettando o rifiutando), M' sposta la testina alla cella 2023.
6. Infine, M' sposta la testina a sinistra di una cella (dalla cella 2023 alla cella 2022).

Ora verifichiamo che questa riduzione soddisfa la proprietà richiesta:

- Se $\langle M, w\rangle \in HALT_{TM}$, allora M si ferma su input w. Di conseguenza, M' simulerà M su w, raggiungerà la cella 2023 e poi si sposterà alla cella 2022. Quindi, $\langle M'\rangle \in 2023_{TM}$.
- Se $\langle M, w\rangle \notin HALT_{TM}$, allora M non si ferma su input w. Di conseguenza, M' rimarrà intrappolata nella simulazione di M su w e non raggiungerà mai la fase in cui sposta la testina alla cella 2023. Quindi, $\langle M'\rangle \notin 2023_{TM}$.

Pertanto, abbiamo $\langle M, w\rangle \in HALT_{TM}$ se e solo se $f(\langle M, w\rangle) \in 2023_{TM}$.

Poiché $HALT_{TM}$ è indecidibile e abbiamo una riduzione da $HALT_{TM}$ a 2023_{TM}, ne segue che anche 2023_{TM} è indecidibile. □

c) Turing-riconoscibilità di 2023_{TM}

Teorema 5. *Il linguaggio* 2023_{TM} *è Turing-riconoscibile ma non co-Turing-riconoscibile.*

Proof. Dimostriamo prima che 2023_{TM} è Turing-riconoscibile. Per questo, descriviamo una macchina di Turing R che riconosce 2023_{TM}:

1. Data una macchina di Turing M, R esegue la seguente procedura:
2. Per ogni stringa $w \in \Sigma^*$ in ordine lessicografico:
   - Simula M su input w passo per passo.
   - Se durante la simulazione la testina di M si sposta dalla cella 2023 alla cella 2022, allora R accetta.

3. Se per nessuna stringa w la testina di M si sposta dalla cella 2023 alla cella 2022, allora R non accetta (potrebbe non fermarsi mai).

Questa macchina di Turing R accetta se e solo se esiste un input w tale che M sposta la testina dalla cella 2023 alla cella 2022. Pertanto, $L(R) = 2023_{TM}$, il che dimostra che 2023_{TM} è Turing-riconoscibile.

Ora, dimostriamo che 2023_{TM} non è co-Turing-riconoscibile. Se 2023_{TM} fosse co-Turing-riconoscibile, allora, dato che abbiamo dimostrato che è anche Turing-riconoscibile, per il teorema che caratterizza i linguaggi decidibili, 2023_{TM} sarebbe decidibile. Ma abbiamo già dimostrato che 2023_{TM} è indecidibile, quindi arriviamo a una contraddizione.

Un'altra argomentazione per dimostrare che 2023_{TM} non è co-Turing-riconoscibile è mostrare che $\overline{2023_{TM}}$ non è Turing-riconoscibile. $\overline{2023_{TM}}$ è il linguaggio delle macchine di Turing che non spostano mai la testina dalla cella 2023 alla cella 2022 per nessun input. Questo è equivalente al problema di determinare se una proprietà non banale (nel nostro caso, "non spostare mai la testina dalla cella 2023 alla cella 2022") vale per tutte le possibili computazioni di una macchina di Turing, che è indecidibile per il teorema di Rice. □

Esercizio 3. Un linguaggio L viene definito co-Turing-riconoscibile se il suo complemento $\overline{L}$ è Turing-riconoscibile.

a) Dimostrare formalmente il seguente teorema: un linguaggio è decidibile se e solo se è sia Turing-riconoscibile che co-Turing-riconoscibile.

b) Dato che $A_{TM} = \{\langle M, w\rangle \mid M \text{ è una TM che accetta la stringa } w\}$ è Turing-riconoscibile ma non decidibile, dimostrare che $\overline{A_{TM}}$ non può essere Turing-riconoscibile.

c) Descrivere un linguaggio che non è né Turing-riconoscibile né co-Turing-riconoscibile. Giustificare la risposta.

Soluzione. a) Caratterizzazione dei linguaggi decidibili

Teorema 6. *Un linguaggio L è decidibile se e solo se è sia Turing-riconoscibile che co-Turing-riconoscibile.*

Proof. Dobbiamo dimostrare entrambe le direzioni dell'equivalenza.

Direzione ($\Rightarrow$): Supponiamo che L sia decidibile. Allora esiste una macchina di Turing M che decide L, cioè per ogni input w:

- Se $w \in L$, allora M accetta w.
- Se $w \notin L$, allora M rifiuta w.

Crucialmente, M si ferma sempre, sia che accetti che rifiuti l'input.

Per dimostrare che L è Turing-riconoscibile, osserviamo che la stessa macchina M riconosce L: accetta esattamente gli input che sono in L.

Per dimostrare che L è co-Turing-riconoscibile, cioè che $\overline{L}$ è Turing-riconoscibile, costruiamo una macchina di Turing M' che riconosce $\overline{L}$. M' opera come segue:

- Su input w, M' simula M su w.

- Se M rifiuta w, allora M' accetta w.
- Se M accetta w, allora M' rifiuta w (o entra in un loop, ma questo non è rilevante per la riconoscibilità).

Poiché M si ferma sempre, M' accetta esattamente gli input che sono in $\overline{L}$. Quindi, $\overline{L}$ è Turing-riconoscibile, cioè L è co-Turing-riconoscibile.

Direzione ($\Leftarrow$): Supponiamo che L sia sia Turing-riconoscibile che co-Turing-riconoscibile. Allora esistono due macchine di Turing:

- M_1 che riconosce L: accetta se $w \in L$, non si ferma o rifiuta se $w \notin L$.
- M_2 che riconosce $\overline{L}$: accetta se $w \notin L$, non si ferma o rifiuta se $w \in L$.

Costruiamo una macchina di Turing M che decide L. M opera come segue:

1. Su input w, M simula sia M_1 che M_2 su w in parallelo (cioè, eseguendo un passo di M_1, poi un passo di M_2, poi un altro passo di M_1, e così via).
2. Se M_1 accetta w, allora M accetta w.
3. Se M_2 accetta w, allora M rifiuta w.

Poiché w è o in L o in $\overline{L}$, esattamente una tra M_1 e M_2 accetterà w. Quindi, M si fermerà sempre e deciderà correttamente l'appartenenza di w a L. Pertanto, L è decidibile. □

b) Non Turing-riconoscibilità di $\overline{A_{TM}}$

Teorema 7. *Il linguaggio $\overline{A_{TM}} = \{\langle M, w\rangle \mid M$ è una TM che non accetta la stringa $w\}$ non è Turing-riconoscibile.*

Proof. Sappiamo che A_{TM} è Turing-riconoscibile ma non decidibile. Supponiamo, per assurdo, che $\overline{A_{TM}}$ sia Turing-riconoscibile. Allora, per il teorema dimostrato nel punto (a), A_{TM} sarebbe sia Turing-riconoscibile che co-Turing-riconoscibile, e quindi decidibile. Ma questo contraddice il fatto che A_{TM} non è decidibile.

Quindi, $\overline{A_{TM}}$ non può essere Turing-riconoscibile. □

c) Linguaggio né Turing-riconoscibile né co-Turing-riconoscibile

Per descrivere un linguaggio che non è né Turing-riconoscibile né co-Turing-riconoscibile, costruiamo un linguaggio che mescola elementi di A_{TM} e $\overline{A_{TM}}$.

Definizione 2. Definiamo il linguaggio L_{mix} come:

$$L_{mix} = \{\langle M, w, 0\rangle \mid \langle M, w\rangle \in A_{TM}\} \cup \{\langle M, w, 1\rangle \mid \langle M, w\rangle \in \overline{A_{TM}}\}$$

Cioè, L_{mix} contiene tutte le triple $\langle M, w, 0\rangle$ tali che M accetta w, e tutte le triple $\langle M, w, 1\rangle$ tali che M non accetta w.

Teorema 8. *Il linguaggio L_{mix} non è né Turing-riconoscibile né co-Turing-riconoscibile.*

Proof. **Non Turing-riconoscibilità di L_{mix}:** Supponiamo, per assurdo, che L_{mix} sia Turing-riconoscibile. Allora esiste una macchina di Turing R che riconosce L_{mix}.

Costruiamo una macchina di Turing R' che riconosce $\overline{A_{TM}}$ come segue:

- Su input $\langle M, w\rangle$, R' simula R su input $\langle M, w, 1\rangle$.
- Se R accetta $\langle M, w, 1\rangle$, allora R' accetta $\langle M, w\rangle$.

Per costruzione, R' accetta $\langle M, w\rangle$ se e solo se R accetta $\langle M, w, 1\rangle$, il che accade se e solo se $\langle M, w\rangle \in \overline{A_{TM}}$. Quindi, R' riconosce $\overline{A_{TM}}$.

Ma abbiamo dimostrato al punto (b) che $\overline{A_{TM}}$ non è Turing-riconoscibile, quindi arriviamo a una contraddizione. Pertanto, L_{mix} non può essere Turing-riconoscibile.

Non co-Turing-riconoscibilità di L_{mix}: Supponiamo, per assurdo, che L_{mix} sia co-Turing-riconoscibile, cioè che $\overline{L_{mix}}$ sia Turing-riconoscibile. Allora esiste una macchina di Turing S che riconosce $\overline{L_{mix}}$.

Osserviamo che $\overline{L_{mix}} = \{\langle M, w, 0\rangle \mid \langle M, w\rangle \in \overline{A_{TM}}\} \cup \{\langle M, w, 1\rangle \mid \langle M, w\rangle \in A_{TM}\}$.

Costruiamo una macchina di Turing S' che riconosce $\overline{A_{TM}}$ come segue:

- Su input $\langle M, w\rangle$, S' simula S su input $\langle M, w, 0\rangle$.
- Se S accetta $\langle M, w, 0\rangle$, allora S' accetta $\langle M, w\rangle$.

Per costruzione, S' accetta $\langle M, w\rangle$ se e solo se S accetta $\langle M, w, 0\rangle$, il che accade se e solo se $\langle M, w, 0\rangle \in \overline{L_{mix}}$, cioè se e solo se $\langle M, w\rangle \in \overline{A_{TM}}$. Quindi, S' riconosce $\overline{A_{TM}}$.

Ma abbiamo dimostrato al punto (b) che $\overline{A_{TM}}$ non è Turing-riconoscibile, quindi arriviamo a una contraddizione. Pertanto, L_{mix} non può essere co-Turing-riconoscibile.

Abbiamo così dimostrato che L_{mix} non è né Turing-riconoscibile né co-Turing-riconoscibile.

□

2 Riducibilità e Dimostrazione di Indecidibilità

Esercizio 4. Consideriamo il concetto di riducibilità mediante funzione:

a) Definire formalmente cosa significa che un linguaggio A è riducibile mediante funzione a un linguaggio B (notazione: $A \leq_m B$). Spiegare il ruolo della funzione di riduzione f e quali proprietà deve soddisfare.

b) Dimostrare che se $A \leq_m B$ e B è decidibile, allora A è decidibile. Spiegare come costruire un decisore per A utilizzando un decisore per B e la funzione di riduzione f.

Soluzione. a) Definizione formale di riducibilità mediante funzione

Definizione 3 (Riducibilità mediante funzione)**.** Un linguaggio A è riducibile mediante funzione a un linguaggio B (notazione: $A \leq_m B$) se esiste una funzione calcolabile $f : \Sigma^* \to \Sigma^*$ tale che per ogni stringa $w \in \Sigma^*$:

$$w \in A \text{ se e solo se } f(w) \in B$$

La funzione f è chiamata funzione di riduzione da A a B.

Il ruolo della funzione di riduzione f è quello di trasformare le istanze del problema A in istanze del problema B in modo tale che la risposta (appartenenza al linguaggio) sia preservata. In altre parole, f trasforma il problema di decidere se $w \in A$ nel problema equivalente di decidere se $f(w) \in B$.

Le proprietà che la funzione di riduzione f deve soddisfare sono:

1. **Calcolabilità**: f deve essere una funzione calcolabile, cioè deve esistere una macchina di Turing che calcola $f(w)$ per ogni input w.
2. **Preservazione dell'appartenenza**: f deve preservare l'appartenenza ai linguaggi, cioè $w \in A$ se e solo se $f(w) \in B$. Questa proprietà garantisce che la riduzione sia corretta.

La riducibilità mediante funzione è un concetto fondamentale nella teoria della calcolabilità e della complessità, perché permette di confrontare la difficoltà intrinseca di diversi problemi. Se $A \leq_m B$, allora B è almeno "difficile" quanto A, nel senso che una soluzione per B può essere utilizzata per ottenere una soluzione per A.

b) Dimostrazione: se $A \leq_m B$ e B è decidibile, allora A è decidibile

Teorema 9. *Se $A \leq_m B$ e B è decidibile, allora A è decidibile.*

Proof. Supponiamo che $A \leq_m B$ e che B sia decidibile. Allora:

- Esiste una funzione di riduzione calcolabile $f : \Sigma^* \to \Sigma^*$ tale che per ogni $w \in \Sigma^*$, $w \in A$ se e solo se $f(w) \in B$.
- Esiste una macchina di Turing M_B che decide B, cioè per ogni stringa $x \in \Sigma^*$, M_B accetta x se $x \in B$ e rifiuta x se $x \notin B$.

Costruiamo una macchina di Turing M_A che decide A come segue:

1. Su input w, M_A calcola $f(w)$. Questo è possibile perché f è una funzione calcolabile.
2. M_A simula M_B su input $f(w)$.
3. Se M_B accetta $f(w)$, allora M_A accetta w.
4. Se M_B rifiuta $f(w)$, allora M_A rifiuta w.

Verifichiamo che M_A decide correttamente A:

- Se $w \in A$, allora $f(w) \in B$ (per la proprietà della funzione di riduzione). Quindi, M_B accetta $f(w)$ e, di conseguenza, M_A accetta w.
- Se $w \notin A$, allora $f(w) \notin B$ (per la proprietà della funzione di riduzione). Quindi, M_B rifiuta $f(w)$ e, di conseguenza, M_A rifiuta w.

Inoltre, M_A si ferma sempre, poiché sia il calcolo di $f(w)$ che la simulazione di M_B su $f(w)$ terminano sempre (la prima perché f è calcolabile, la seconda perché M_B è un decisore).

Pertanto, M_A è un decisore per A, il che dimostra che A è decidibile. □

Un'importante conseguenza di questo teorema è il suo contropositivo: se A è indecidibile e $A \leq_m B$, allora B è indecidibile. Questo è il principio alla base delle dimostrazioni di indecidibilità mediante riduzione: per dimostrare che un nuovo problema B è indecidibile, si può mostrare che un problema noto indecidibile A (come il problema della fermata) è riducibile a B.

Esercizio 5. Consideriamo il problema di determinare se un PDA accetta qualche stringa nella forma $\{ww \mid w \in \{0,1\}^*\}$.

a) Formulare questo problema come un linguaggio WW_{PDA}.

b) Dimostrare che il linguaggio WW_{PDA} è indecidibile mediante una riduzione appropriata. Quale problema utilizzereste come punto di partenza e perché?

c) Spiegare perché questo risultato è interessante nel contesto dei linguaggi context-free, dato che è noto che $\{ww \mid w \in \{0,1\}^*\}$ non è un linguaggio context-free.

Soluzione. a) Formulazione del problema come linguaggio WW_{PDA}

Definizione 4. Il linguaggio WW_{PDA} è definito come:

$$WW_{PDA} = \{\langle P \rangle \mid P \text{ è un PDA e } L(P) \cap \{ww \mid w \in \{0,1\}^*\} \neq \emptyset\}$$

Dove $\langle P \rangle$ rappresenta la codifica dell'automa a pila P.

In altre parole, WW_{PDA} è l'insieme delle codifiche di tutti gli automi a pila che accettano almeno una stringa della forma ww, dove w è una stringa di 0 e 1.

b) Dimostrazione dell'indecidibilità di WW_{PDA}

Teorema 10. *Il linguaggio WW_{PDA} è indecidibile.*

Proof. Dimostriamo l'indecidibilità di WW_{PDA} attraverso una riduzione dal problema dell'accettazione $A_{TM} = \{\langle M, w \rangle \mid M \text{ è una TM che accetta la stringa } w\}$, che è noto essere indecidibile.

Data una coppia $\langle M, w \rangle$, dove M è una macchina di Turing e w è una stringa, costruiamo un PDA $P_{M,w}$ con la seguente proprietà: $P_{M,w}$ accetta almeno una stringa della forma xx (con $x \in \{0,1\}^*$) se e solo se M accetta w.

Il PDA $P_{M,w}$ opera come segue:

1. **Generazione di un'istantanea:** $P_{M,w}$ utilizza il suo non determinismo per generare una possibile istantanea (configurazione) dell'esecuzione di M su input w. Un'istantanea è una stringa che rappresenta:

   - Il contenuto del nastro di M
   - La posizione della testina
   - Lo stato corrente di M

   Questa istantanea viene codificata in una stringa $x \in \{0,1\}^*$.

2. **Verifica dell'istantanea:** $P_{M,w}$ utilizza un secondo stack (simulato all'interno di un unico stack fisico) per verificare che l'istantanea generata rappresenti effettivamente una configurazione accettante di M su w.

3. **Duplicazione e confronto:** Dopo aver generato un'istantanea accettante x, $P_{M,w}$ deve riconoscere la stringa xx. Per fare questo:

   - Memorizza i primi $|x|/2$ simboli nello stack
   - Accetta solo stringhe di lunghezza pari
   - Dopo aver letto metà dell'input (contata con un contatore sullo stack), entra in una fase di confronto
   - Confronta i simboli rimanenti con quelli memorizzati nello stack, accettando solo se corrispondono perfettamente

Osserviamo che:

- Se M accetta w, allora esiste almeno una configurazione accettante nell'esecuzione di M su w. $P_{M,w}$ può non deterministicamente generare questa configurazione e, quando verifica che è una configurazione accettante, accetterà la stringa xx dove x è la codifica di questa configurazione.
- Se M non accetta w, allora non esiste alcuna configurazione accettante nell'esecuzione di M su w. Di conseguenza, $P_{M,w}$ non accetterà alcuna stringa della forma xx.

È importante notare che un PDA con due stack è Turing-equivalente, ma noi utilizziamo un'implementazione che codifica il secondo stack all'interno del primo, mantenendo la potenza espressiva di un PDA standard.

Pertanto, $\langle M, w\rangle \in A_{TM}$ se e solo se $\langle P_{M,w}\rangle \in WW_{PDA}$. Poiché A_{TM} è indecidibile, ne segue che anche WW_{PDA} è indecidibile. □

c) Interesse del risultato nel contesto dei linguaggi context-free

Questo risultato è particolarmente interessante nel contesto dei linguaggi context-free per diversi motivi:

1. **Non context-free vs. Intersezione con context-free:** Il linguaggio $L_{ww} = \{ww \mid w \in \{0,1\}^*\}$ è noto per non essere context-free. Questo può essere dimostrato utilizzando il lemma del pumping per i linguaggi context-free. Tuttavia, il nostro risultato mostra che, nonostante L_{ww} non sia context-free, determinare se un linguaggio context-free (cioè un linguaggio accettato da un PDA) ha intersezione non vuota con L_{ww} è un problema indecidibile.

2. **Limitazioni dei PDA:** I PDA sono modelli computazionali più potenti rispetto agli automi a stati finiti, ma meno potenti rispetto alle macchine di Turing. Il nostro risultato evidenzia un limite fondamentale dei PDA: sebbene non possano accettare il linguaggio L_{ww} nella sua interezza, possono accettare sottoinsiemi di L_{ww} in modi così complessi che diventa indecidibile determinare se un PDA dato accetta qualche stringa in L_{ww}.

3. **Proprietà dei linguaggi context-free:** Questo risultato si collega a un tema più ampio nella teoria dei linguaggi formali: quali proprietà dei linguaggi context-free sono decidibili? Sappiamo che alcune proprietà, come l'essere vuoto o finito, sono decidibili per i linguaggi context-free. Il nostro risultato aggiunge all'elenco delle proprietà indecidibili, mostrando che l'intersezione non vuota con un linguaggio specifico non context-free come L_{ww} è indecidibile.

4. **Complessità dell'intersezione:** In generale, l'intersezione di due linguaggi context-free può non essere context-free. Il nostro risultato suggerisce che l'intersezione di un linguaggio context-free con un linguaggio non context-free può essere così complessa che persino determinare se è vuota diventa indecidibile.

In sintesi, questo risultato illustra un fenomeno interessante nella teoria dei linguaggi formali: anche se un linguaggio L (in questo caso, L_{ww}) non è abbastanza "espressivo" da essere context-free, il problema di determinare se un linguaggio context-free ha intersezione non vuota con L può essere indecidibile. Questo evidenzia la complessità e la ricchezza della teoria dei linguaggi formali, dove anche problemi apparentemente semplici possono rivelarsi indecidibili.

Esercizio 6. Sia $E_{TM} = \{\langle M \rangle \mid M \text{ è una TM tale che } L(M) = \emptyset\}$ il problema del vuoto per macchine di Turing.

a) Dimostrare che $A_{TM} \leq_m \overline{E_{TM}}$ (dove $\overline{E_{TM}}$ è il complemento di E_{TM}) fornendo una riduzione mediante funzione esplicita. Spiegare come questa riduzione trasforma un'istanza di A_{TM} in un'istanza di $\overline{E_{TM}}$.

b) Utilizzando la riduzione precedente e il fatto che A_{TM} è indecidibile, dimostrare che E_{TM} è indecidibile.

c) Dimostrare che E_{TM} è co-Turing-riconoscibile ma non Turing-riconoscibile.

Soluzione. a) Riduzione da A_{TM} a $\overline{E_{TM}}$

Teorema 11. *$A_{TM} \leq_m \overline{E_{TM}}$, cioè il problema dell'accettazione è riducibile mediante funzione al complemento del problema del vuoto.*

Proof. Ricordiamo che:

- $A_{TM} = \{\langle M, w \rangle \mid M \text{ è una TM che accetta la stringa } w\}$
- $E_{TM} = \{\langle M \rangle \mid M \text{ è una TM tale che } L(M) = \emptyset\}$
- $\overline{E_{TM}} = \{\langle M \rangle \mid M \text{ è una TM tale che } L(M) \neq \emptyset\}$

Definiamo una funzione di riduzione $f : \{\langle M, w \rangle\} \to \{\langle M' \rangle\}$ come segue: data una coppia $\langle M, w \rangle$, dove M è una macchina di Turing e w è una stringa, costruiamo una nuova macchina di Turing M' tale che:

- M' accetta ogni stringa di input x se e solo se $x = w$ e M accetta w.

La macchina di Turing M' opera come segue:

1. Su input x, M' verifica se $x = w$.

2. Se $x \neq w$, allora M' rifiuta immediatamente.

3. Se $x = w$, allora M' simula M su input w.

4. Se M accetta w, allora M' accetta x. Altrimenti, M' rifiuta x o non si ferma, a seconda del comportamento di M su w.

Ora dimostriamo che questa funzione f è una riduzione valida da A_{TM} a $\overline{E_{TM}}$:

- Se $\langle M, w\rangle \in A_{TM}$, allora M accetta w. Di conseguenza, M' accetta esattamente una stringa, cioè w. Quindi, $L(M') = \{w\} \neq \emptyset$, e $\langle M'\rangle \in \overline{E_{TM}}$.
- Se $\langle M, w\rangle \notin A_{TM}$, allora M non accetta w (cioè, M rifiuta w o non si ferma su w). Di conseguenza, M' non accetta alcuna stringa. Quindi, $L(M') = \emptyset$, e $\langle M'\rangle \notin \overline{E_{TM}}$.

Pertanto, $\langle M, w\rangle \in A_{TM}$ se e solo se $f(\langle M, w\rangle) = \langle M'\rangle \in \overline{E_{TM}}$, il che dimostra che f è una riduzione valida da A_{TM} a $\overline{E_{TM}}$. □

b) Indecidibilità di E_{TM}

Teorema 12. *Il linguaggio E_{TM} è indecidibile.*

Proof. Abbiamo dimostrato al punto (a) che $A_{TM} \leq_m \overline{E_{TM}}$. Sappiamo che A_{TM} è indecidibile. Per il teorema della riducibilità mediante funzione, se $A \leq_m B$ e A è indecidibile, allora anche B è indecidibile. Quindi, $\overline{E_{TM}}$ è indecidibile.

Ora, se $\overline{E_{TM}}$ è indecidibile, allora anche E_{TM} è indecidibile. Questo perché se E_{TM} fosse decidibile, allora potremmo decidere $\overline{E_{TM}}$ semplicemente invertendo la risposta del decisore per E_{TM}. Quindi, E_{TM} è indecidibile. □

c) E_{TM} è co-Turing-riconoscibile ma non Turing-riconoscibile

Teorema 13. *Il linguaggio E_{TM} è co-Turing-riconoscibile ma non Turing-riconoscibile.*

Proof. **E_{TM} è co-Turing-riconoscibile:** Per dimostrare che E_{TM} è co-Turing-riconoscibile, dobbiamo dimostrare che $\overline{E_{TM}}$ è Turing-riconoscibile.

Ricordiamo che $\overline{E_{TM}} = \{\langle M\rangle \mid M \text{ è una TM tale che } L(M) \neq \emptyset\}$, cioè l'insieme delle codifiche di macchine di Turing che accettano almeno una stringa.

Costruiamo una macchina di Turing R che riconosce $\overline{E_{TM}}$:

1. Su input $\langle M\rangle$, R enumera tutte le stringhe $w \in \Sigma^*$ in ordine lessicografico.

2. Per ogni stringa w, R simula M su input w.

3. Se M accetta w, allora R accetta $\langle M\rangle$.

Se $L(M) \neq \emptyset$, allora esiste almeno una stringa w tale che M accetta w. R eventualmente enumererà questa stringa e verificherà che M la accetta, quindi R accetterà $\langle M\rangle$. D'altra parte, se $L(M) = \emptyset$, allora M non accetta alcuna stringa, quindi R non accetterà mai $\langle M\rangle$. Pertanto, $L(R) = \overline{E_{TM}}$, il che dimostra che $\overline{E_{TM}}$ è Turing-riconoscibile, cioè E_{TM} è co-Turing-riconoscibile.

E_{TM} **non è Turing-riconoscibile:** Abbiamo dimostrato al punto (b) che E_{TM} è indecidibile. Se E_{TM} fosse Turing-riconoscibile, allora, dato che è anche co-Turing-riconoscibile (come dimostrato sopra), per il teorema che caratterizza i linguaggi decidibili (vedi esercizio 3), E_{TM} sarebbe decidibile. Ma questo contraddice il fatto che E_{TM} è indecidibile. Pertanto, E_{TM} non può essere Turing-riconoscibile. □

3 Analisi di Problemi di Indecidibilità avanzati

Esercizio 7. Sia $FORTY - TWO_{TM} = \{\langle M, w\rangle \mid M$ termina la computazione su w avendo solo 42 sul nastro$\}$.

a) Dimostrare che $FORTY - TWO_{TM}$ è indecidibile mediante una riduzione da un problema noto. Specificare chiaramente la funzione di riduzione e verificare che soddisfi le proprietà necessarie.

b) Dimostrare che $FORTY-TWO_{TM}$ è Turing-riconoscibile. Descrivere una macchina di Turing che riconosce questo linguaggio.

c) Discutere la difficoltà di questo problema rispetto al problema dell'accettazione A_{TM} e al problema della fermata $HALT_{TM}$.

Soluzione. a) Indecidibilità di $FORTY$**-**TWO_{TM}

Teorema 14. *Il linguaggio* $FORTY$*-*TWO_{TM} *è indecidibile.*

Proof. Dimostriamo l'indecidibilità di $FORTY$-TWO_{TM} attraverso una riduzione dal problema della fermata $HALT_{TM}$, che è noto essere indecidibile.

Ricordiamo che $HALT_{TM} = \{\langle M, w\rangle \mid M$ è una TM che si ferma su input $w\}$.

Definiamo una funzione di riduzione $f : \{\langle M, w\rangle\} \to \{\langle M', w\rangle\}$ come segue: data una coppia $\langle M, w\rangle$, dove M è una macchina di Turing e w è una stringa, costruiamo una nuova macchina di Turing M' che opera come segue:

1. Su input x, M' simula M su input x.
2. Se M si ferma su x (accettando o rifiutando), allora M' cancella tutto il contenuto del nastro e scrive solo il numero `42` sul nastro.
3. M' si ferma.

Ora dimostriamo che questa funzione f è una riduzione valida da $HALT_{TM}$ a $FORTY$-TWO_{TM}:

- Se $\langle M, w\rangle \in HALT_{TM}$, allora M si ferma su input w. Di conseguenza, M' simula M su w, e quando M si ferma, M' cancella il nastro e scrive `42`. Quindi, M' termina la computazione su w avendo solo `42` sul nastro, e $\langle M', w\rangle \in FORTY$-$TWO_{TM}$.
- Se $\langle M, w\rangle \notin HALT_{TM}$, allora M non si ferma su input w. Di conseguenza, M' rimane intrappolata nella simulazione di M su w e non raggiunge mai la fase in cui cancella il nastro e scrive `42`. Quindi, M' non termina la computazione su w avendo solo `42` sul nastro, e $\langle M', w\rangle \notin FORTY$-$TWO_{TM}$.

Pertanto, $\langle M, w\rangle \in HALT_{TM}$ se e solo se $f(\langle M, w\rangle) = \langle M', w\rangle \in FORTY\text{-}TWO_{TM}$, il che dimostra che f è una riduzione valida da $HALT_{TM}$ a $FORTY\text{-}TWO_{TM}$.

Poiché $HALT_{TM}$ è indecidibile e abbiamo una riduzione da $HALT_{TM}$ a $FORTY\text{-}TWO_{TM}$, ne segue che anche $FORTY\text{-}TWO_{TM}$ è indecidibile. □

b) Turing-riconoscibilità di $FORTY - TWO_{TM}$

Teorema 15. *Il linguaggio $FORTY - TWO_{TM}$ è Turing-riconoscibile.*

Proof. Per dimostrare che $FORTY - TWO_{TM}$ è Turing-riconoscibile, costruiamo una macchina di Turing R che riconosce $FORTY - TWO_{TM}$:

1. Su input $\langle M, w\rangle$, R simula M su input w passo per passo.
2. Se M non si ferma, allora R non si ferma (rimane intrappolata nella simulazione).
3. Se M si ferma, R verifica il contenuto finale del nastro di M.
4. Se il nastro contiene solo il numero 42, allora R accetta $\langle M, w\rangle$.
5. Se il nastro contiene qualsiasi altra cosa, allora R rifiuta $\langle M, w\rangle$.

Verifichiamo che R riconosce correttamente $FORTY - TWO_{TM}$:

- Se $\langle M, w\rangle \in FORTY - TWO_{TM}$, allora M termina la computazione su w avendo solo 42 sul nastro. In questo caso, R simula M su w, che eventualmente si ferma. Quando M si ferma, R verifica che il nastro contenga solo 42, e poiché questa condizione è soddisfatta, R accetta $\langle M, w\rangle$.
- Se $\langle M, w\rangle \notin FORTY - TWO_{TM}$, allora o M non si ferma su input w, o M si ferma ma non ha solo 42 sul nastro. Nel primo caso, R non si ferma (rimane intrappolata nella simulazione di M su w). Nel secondo caso, R simula M su w fino a quando M si ferma, ma poi verifica che il nastro non contiene solo 42, e quindi rifiuta $\langle M, w\rangle$.

Pertanto, $L(R) = FORTY - TWO_{TM}$, il che dimostra che $FORTY - TWO_{TM}$ è Turing-riconoscibile. □

c) Discussione sulla difficoltà del problema

Il problema $FORTY - TWO_{TM}$ è strettamente legato ai problemi dell'accettazione A_{TM} e della fermata $HALT_{TM}$, ma presenta alcune differenze significative:

1. **Confronto con $HALT_{TM}$:**
   - $HALT_{TM}$ si concentra solo sulla fermata della macchina di Turing, indipendentemente dallo stato finale del nastro.
   - $FORTY - TWO_{TM}$ richiede non solo che la macchina si fermi, ma anche che il nastro contenga un valore specifico al termine della computazione.
   - Abbiamo dimostrato che $FORTY - TWO_{TM}$ è almeno difficile quanto $HALT_{TM}$ (attraverso la riduzione), il che significa che $FORTY - TWO_{TM}$ è almeno indecidibile quanto $HALT_{TM}$.

- D'altra parte, $FORTY - TWO_{TM}$ può essere considerato un caso speciale di $HALT_{TM}$ con vincoli aggiuntivi, il che potrebbe suggerire che non è più difficile di $HALT_{TM}$.

2. **Confronto con A_{TM}:**

   - A_{TM} si concentra sull'accettazione della macchina di Turing, cioè sul fatto che la macchina si fermi in uno stato di accettazione.
   - $FORTY - TWO_{TM}$ si concentra sul contenuto finale del nastro, indipendentemente dallo stato di fermata (accettazione o rifiuto).
   - Questi due problemi sono concettualmente diversi, poiché A_{TM} riguarda il comportamento della macchina (lo stato in cui si ferma), mentre $FORTY - TWO_{TM}$ riguarda il risultato della computazione (il contenuto del nastro).

3. **Classificazione nella gerarchia della computabilità:**

   - Come $HALT_{TM}$, $FORTY - TWO_{TM}$ è Turing-riconoscibile ma non decidibile.
   - Questo lo colloca nella stessa classe di complessità di $HALT_{TM}$ nella gerarchia della computabilità.
   - A_{TM} è anch'esso Turing-riconoscibile ma non decidibile, ma il suo complemento non è Turing-riconoscibile.
   - Non abbiamo dimostrato se il complemento di $FORTY - TWO_{TM}$ è Turing-riconoscibile, ma è plausibile che non lo sia, il che lo collocherebbe nella stessa classe di A_{TM}.

4. **Aspetti pratici:**

   - Dal punto di vista pratico, $FORTY - TWO_{TM}$ potrebbe essere considerato più complesso di $HALT_{TM}$ e A_{TM} perché richiede di analizzare non solo il comportamento della macchina, ma anche lo stato specifico del nastro al termine della computazione.
   - Questo tipo di problema è rilevante in contesti di verifica del software, dove si vuole verificare non solo che un programma termini, ma anche che produca un output specifico.

In sintesi, $FORTY - TWO_{TM}$ è un problema indecidibile che condivide molte caratteristiche con $HALT_{TM}$ e A_{TM}, ma si focalizza su un aspetto diverso della computazione: il contenuto finale del nastro. Mentre la sua indecidibilità può essere dimostrata riducendo da $HALT_{TM}$, il problema ha una propria identità e complessità che lo rende interessante nel contesto della teoria della computabilità.

Esercizio 8. Una Turing Machine moltiplica correttamente se, dati in input due numeri binari separati da #, termina la computazione con la loro moltiplicazione (in binario) sul nastro.

a) Formulare questo problema come un linguaggio $MULT_{TM}$.

b) Dimostrare che il linguaggio $MULT_{TM}$ è indecidibile mediante una riduzione appropriata.

c) Spiegare perché il problema di verificare se una TM calcola correttamente una funzione specifica (come la moltiplicazione) è generalmente indecidibile, mentre sappiamo che esistono TM che calcolano correttamente tali funzioni.

Soluzione. a) Formulazione del problema come linguaggio $MULT_{TM}$

Definizione 5. Il linguaggio $MULT_{TM}$ è definito come: $MULT_{TM} = \{\langle M\rangle \mid M$ è una TM tale che per ogni coppia di numeri binari x, y, su input $x\#y$, M si ferma con $x \cdot y$(in binario) sul nastro$\}$.

Dove $\langle M\rangle$ rappresenta la codifica della macchina di Turing M, e $x\cdot y$ denota il prodotto dei numeri binari x e y.

In altre parole, $MULT_{TM}$ è l'insieme delle codifiche di tutte le macchine di Turing che, date in input due stringhe binarie separate da #, calcolano correttamente il loro prodotto in binario e terminano con questo risultato sul nastro.

b) Indecidibilità di $MULT_{TM}$

Teorema 16. *Il linguaggio* $MULT_{TM}$ *è indecidibile.*

Proof. Dimostriamo l'indecidibilità di $MULT_{TM}$ attraverso una riduzione dal problema $HALT_{ALL_{TM}}$, che è noto essere indecidibile.

$HALT_{ALL_{TM}} = \{\langle M\rangle \mid M$ è una TM che si ferma su ogni input possibile$\}$

Costruiamo una riduzione $f : HALT_{ALL_{TM}} \to MULT_{TM}$ tale che $\langle M\rangle \in HALT_{ALL_{TM}}$ se e solo se $f(\langle M\rangle) \in MULT_{TM}$.

Data una macchina di Turing M, definiamo $f(\langle M\rangle) = \langle M'\rangle$ dove M' è una nuova macchina di Turing che opera come segue:

1. Su input $x\#y$, M' calcola immediatamente il prodotto $z = x\cdot y$ usando un algoritmo standard di moltiplicazione binaria. Nota: la verifica che x e y siano stringhe binarie valide è un dettaglio superfluo che può essere omesso, poiché la definizione di $MULT_{TM}$ già presuppone che l'input sia composto da numeri binari.

2. M' estrae una stringa w da x e y (ad esempio, concatenandoli come $w = x \circ y$).

3. M' simula M su input w.

4. **Punto cruciale:** Indipendentemente dal risultato della simulazione di M su w, se M si ferma (sia accettando che rifiutando), M' restituisce z (il prodotto già calcolato) come risultato. Se M non si ferma, anche M' non si fermerà.

Questo comportamento garantisce che M' moltiplichi correttamente tutti gli input se e solo se M si ferma su tutti gli input possibili.

Ora verifichiamo che questa riduzione soddisfa la proprietà richiesta:

- Se $\langle M\rangle \in HALT_{ALL_{TM}}$, allora M si ferma su ogni input possibile. Di conseguenza, per ogni coppia di stringhe binarie x e y, M' su input $x\#y$ calcolerà $z = x \cdot y$, simulerà M su w (che si fermerà perché M si ferma su ogni input), e infine restituirà z sul nastro. Quindi, M' moltiplica correttamente ogni coppia di numeri binari, e $\langle M'\rangle \in MULT_{TM}$.

- Se $\langle M \rangle \notin HALT_{ALL_{TM}}$, allora esiste almeno un input w_0 su cui M non si ferma. Consideriamo la coppia di stringhe binarie x_0 e y_0 che generano w_0 secondo la procedura usata da M'. Su input $x_0 \# y_0$, M' rimarrà intrappolata nella simulazione di M su w_0 e non completerà mai la computazione. Quindi, M' non moltiplica correttamente tutte le coppie di numeri binari, e $\langle M' \rangle \notin MULT_{TM}$.

Pertanto, $\langle M \rangle \in HALT_{ALL_{TM}}$ se e solo se $f(\langle M \rangle) = \langle M' \rangle \in MULT_{TM}$, il che dimostra che f è una riduzione valida da $HALT_{ALL_{TM}}$ a $MULT_{TM}$.

Poiché $HALT_{ALL_{TM}}$ è indecidibile e abbiamo una riduzione da $HALT_{ALL_{TM}}$ a $MULT_{TM}$, ne segue che anche $MULT_{TM}$ è indecidibile. □

c) Indecidibilità della verifica di funzioni specifiche

Il problema di verificare se una macchina di Turing calcola correttamente una funzione specifica (come la moltiplicazione) è generalmente indecidibile, mentre sappiamo che esistono macchine di Turing che calcolano correttamente tali funzioni. Questa apparente contraddizione è dovuta a diverse ragioni fondamentali:

1. **Esistenza vs. Verifica:**
   - **Esistenza:** Sappiamo costruttivamente che esistono macchine di Turing che calcolano correttamente funzioni come la moltiplicazione, l'addizione, ecc. Possiamo esplicitamente progettare tali macchine e verificare manualmente la loro correttezza.
   - **Verifica automatica:** Il problema di decidere automaticamente se una macchina di Turing arbitraria calcola correttamente una funzione specifica è indecidibile. Non esiste un algoritmo generale che, data una descrizione di una macchina di Turing, possa determinare con certezza se quella macchina calcola correttamente la funzione in questione per tutti i possibili input.
2. **Comportamento su tutti gli input:**
   - Per verificare che una macchina di Turing calcoli correttamente una funzione f, dobbiamo verificare che la macchina si comporti correttamente su ogni possibile input valido. Questo richiede di analizzare un numero potenzialmente infinito di casi.
   - Il problema è che non possiamo simulare la macchina su tutti i possibili input in un tempo finito, e non esiste un modo generale per "riassumere" il comportamento della macchina su tutti gli input.
3. **Riduzione a problemi indecidibili:**
   - Come abbiamo dimostrato, il problema di verificare se una macchina di Turing calcola correttamente la moltiplicazione può essere ridotto dal problema $HALT_{ALL_{TM}}$, che è indecidibile.
   - In generale, molti problemi di verifica di proprietà semantiche di programmi (come il calcolo di funzioni specifiche) possono essere ridotti da problemi indecidibili come il problema della fermata o il problema dell'accettazione.

4. **Teorema di Rice:**

   - Il teorema di Rice afferma che ogni proprietà non banale della funzione calcolata da una macchina di Turing è indecidibile.
   - La proprietà "calcola correttamente la moltiplicazione" è certamente non banale (cioè, non è soddisfatta da tutte le macchine di Turing o da nessuna), quindi per il teorema di Rice, questa proprietà è indecidibile.

5. **Approcci parziali e tecniche di verifica:**

   - Nonostante l'indecidibilità generale, esistono approcci parziali e tecniche di verifica formale che possono essere utilizzati per aumentare la fiducia nella correttezza di programmi specifici.
   - Questi approcci includono dimostrazione assistita, model checking, analisi statica, ecc. Tuttavia, nessuno di questi approcci può garantire la completa automazione e la decidibilità per programmi arbitrari.

In sintesi, l'indecidibilità della verifica di funzioni specifiche è un esempio del divario fondamentale tra ciò che possiamo costruire e ciò che possiamo decidere automaticamente. Possiamo costruire macchine di Turing che calcolano correttamente funzioni specifiche, ma non possiamo decidere automaticamente se una macchina di Turing arbitraria calcola correttamente tali funzioni per tutti i possibili input. Questo limite fondamentale ha profonde implicazioni per la verifica del software e la dimostrazione automatica di teoremi.

Esercizio 9. Sia $SELF_{TM} = \{\langle M \rangle \mid M \text{ è una TM che accetta la propria codifica } \langle M \rangle\}$.

a) Dimostrare che $SELF_{TM}$ è indecidibile utilizzando una tecnica di diagonalizzazione. Fornire una dimostrazione dettagliata.

b) Dimostrare che $SELF_{TM}$ è Turing-riconoscibile. Descrivere una macchina di Turing che riconosce questo linguaggio.

c) Descrivere una macchina di Turing U che, per ogni macchina di Turing M, ha la proprietà che U accetta $\langle M \rangle$ se e solo se M non accetta $\langle M \rangle$. Spiegare perché l'esistenza di U porta a una contraddizione.

Soluzione. a) Indecidibilità di $SELF_{TM}$ mediante diagonalizzazione

Teorema 17. *Il linguaggio $SELF_{TM} = \{\langle M \rangle \mid M$ è una TM che accetta la propria codifica $\langle M \rangle\}$ è indecidibile.*

Proof. Dimostriamo l'indecidibilità di $SELF_{TM}$ utilizzando il metodo della diagonalizzazione. Supponiamo, per assurdo, che $SELF_{TM}$ sia decidibile. Allora esiste una macchina di Turing D che decide $SELF_{TM}$, cioè:

- Se $\langle M \rangle \in SELF_{TM}$ (ovvero, M accetta $\langle M \rangle$), allora D accetta $\langle M \rangle$.
- Se $\langle M \rangle \notin SELF_{TM}$ (ovvero, M non accetta $\langle M \rangle$), allora D rifiuta $\langle M \rangle$.

Ora, costruiamo una nuova macchina di Turing N che opera come segue:

1. Su input x, N simula D su input x.
2. Se D accetta x, allora N rifiuta.
3. Se D rifiuta x, allora N accetta.

In altre parole, N fa esattamente l'opposto di ciò che D fa. Considerando il comportamento di N sulla propria codifica $\langle N \rangle$, abbiamo due casi possibili:

- Caso 1: N accetta $\langle N \rangle$. Secondo la definizione di N, questo significa che D rifiuta $\langle N \rangle$. Ma secondo la definizione di D, D rifiuta $\langle N \rangle$ se e solo se N non accetta $\langle N \rangle$. Questa è una contraddizione.
- Caso 2: N non accetta $\langle N \rangle$. Secondo la definizione di N, questo significa che D accetta $\langle N \rangle$. Ma secondo la definizione di D, D accetta $\langle N \rangle$ se e solo se N accetta $\langle N \rangle$. Questa è anche una contraddizione.

Poiché entrambi i casi portano a una contraddizione, la nostra ipotesi che $SELF_{TM}$ sia decidibile deve essere falsa. Quindi, $SELF_{TM}$ è indecidibile. □

b) Turing-riconoscibilità di $SELF_{TM}$

Teorema 18. *Il linguaggio $SELF_{TM}$ è Turing-riconoscibile.*

Proof. Per dimostrare che $SELF_{TM}$ è Turing-riconoscibile, descriviamo una macchina di Turing R che riconosce $SELF_{TM}$:

1. Su input $\langle M \rangle$, R simula M su input $\langle M \rangle$.
2. Se M accetta $\langle M \rangle$, allora R accetta $\langle M \rangle$.
3. Se M rifiuta $\langle M \rangle$ o non si ferma, allora R non si ferma (rimane intrappolata nella simulazione).

Verifichiamo che R riconosce correttamente $SELF_{TM}$:

- Se $\langle M \rangle \in SELF_{TM}$, allora M accetta $\langle M \rangle$. Di conseguenza, R simulerà M su $\langle M \rangle$, che eventualmente accetterà, e quindi R accetterà $\langle M \rangle$.
- Se $\langle M \rangle \notin SELF_{TM}$, allora M non accetta $\langle M \rangle$ (cioè, M rifiuta $\langle M \rangle$ o non si ferma su $\langle M \rangle$). Di conseguenza, R rimarrà intrappolata nella simulazione di M su $\langle M \rangle$ e non accetterà mai $\langle M \rangle$.

Pertanto, $L(R) = SELF_{TM}$, il che dimostra che $SELF_{TM}$ è Turing-riconoscibile. □

c) Macchina universale di Turing e contraddizione

Descriviamo una macchina di Turing U che, per ogni macchina di Turing M, ha la proprietà che U accetta $\langle M \rangle$ se e solo se M non accetta $\langle M \rangle$.

La macchina di Turing U opera come segue:

1. Su input $\langle M \rangle$, U simula M su input $\langle M \rangle$.
2. Se M accetta $\langle M \rangle$, allora U rifiuta $\langle M \rangle$.
3. Se M rifiuta $\langle M \rangle$, allora U accetta $\langle M \rangle$.
4. Se M non si ferma su $\langle M \rangle$, allora U non si ferma su $\langle M \rangle$.

Questa definizione di U pone un problema: cosa succede quando U viene eseguita sulla propria codifica $\langle U \rangle$?

Consideriamo il comportamento di U su input $\langle U \rangle$:

- U simula U su input $\langle U \rangle$.
- Se U accetta $\langle U \rangle$, allora U rifiuta $\langle U \rangle$.
- Se U rifiuta $\langle U \rangle$, allora U accetta $\langle U \rangle$.

Questo porta a una contraddizione: se U accetta $\langle U \rangle$, allora U rifiuta $\langle U \rangle$, e se U rifiuta $\langle U \rangle$, allora U accetta $\langle U \rangle$. Non c'è un comportamento consistente per U su input $\langle U \rangle$.

Questa contraddizione è essenzialmente lo stesso paradosso del rasoio di Russell nella teoria degli insiemi: "Il barbiere rade tutti e soli coloro che non si radono da soli. Chi rade il barbiere?" Se il barbiere si rade da solo, allora non dovrebbe radersi da solo. Se non si rade da solo, allora dovrebbe radersi da solo.

Analogamente, nel nostro caso, U è definita per comportarsi in modo opposto a qualsiasi macchina di Turing quando questa viene eseguita sulla propria codifica. Ma U stessa è una macchina di Turing, quindi quando viene eseguita sulla propria codifica, dovrebbe comportarsi in modo opposto a se stessa, il che è una contraddizione logica.

Questa contraddizione dimostra che non può esistere una macchina di Turing U con la proprietà specificata. In particolare, dimostra che il problema di determinare se una macchina di Turing accetta la propria codifica non può essere decidibile, confermando il risultato della parte (a).

L'esistenza di questa contraddizione è anche alla base del teorema di incompletezza di Gödel e di molti altri risultati limitativi nella logica e nella teoria della calcolabilità.

4 Appendice: PCP

Definizione 6 (Post Correspondence Problem (PCP))**.** Il *Post Correspondence Problem* è un problema di decisione definito come segue:

- **Input**: Due liste di stringhe $A = (a_1, a_2, \ldots, a_n)$ e $B = (b_1, b_2, \ldots, b_n)$ su un alfabeto finito Σ.
- **Domanda**: Esiste una sequenza di indici $i_1, i_2, \ldots, i_k$ (con $1 \leq i_j \leq n$ per ogni j) tale che $a_{i_1} a_{i_2} \ldots a_{i_k} = b_{i_1} b_{i_2} \ldots b_{i_k}$?

Il PCP è un problema fondamentale nella teoria della calcolabilità ed è noto essere indecidibile. In questo documento, assumiamo come dato questo risultato classico, sebbene la sua dimostrazione originale (dovuta a Emil Post nel 1946) si basi su una riduzione dal problema della fermata per macchine di Turing.

Teorema 19. *Il Post Correspondence Problem è indecidibile.*

Il PCP è particolarmente utile per dimostrare l'indecidibilità di problemi relativi a linguaggi e grammatiche context-free, dato che può essere facilmente codificato in termini di operazioni di string matching e manipolazioni sintattiche tipiche di questi formalismi.