Lezione di Informatica Teorica: Problemi Indecidibili e Classi di Calcolabilità

Appunti da Trascrizione Automatica

30 giugno 2025

Indice

1 Il Problema del Tassellamento (Tiling Problem)

Il Tiling Problem è un problema indecidibile che, a prima vista, non sembra strettamente legato alle Macchine di Turing. La sua indecidibilità viene dimostrata tramite una riduzione da un problema di arresto.

1.1 Definizione Intuitiva del Tiling Problem

Si consideri il primo quadrante cartesiano (una superficie infinita divisa in celle unitarie) che si vuole ricoprire con piastrelle. Ogni piastrella ha la stessa dimensione di una cella. Le piastrelle non possono essere ruotate o stirate. Esistono un numero finito di **tipi** di piastrelle, ma un numero infinito di piastrelle per ogni tipo.

Ogni piastrella è idealmente divisa in quattro porzioni, ciascuna con un colore. Le piastrelle possono essere affiancate (orizzontalmente o verticalmente) solo se i lati adiacenti hanno lo stesso colore. Una piastrella specifica, chiamata D_0, deve essere posizionata nella cella di origine $(0, 0)$.

Il problema è: dato un sistema di piastrelle (tipi e regole di affiancamento), è possibile ricoprire interamente il primo quadrante cartesiano?

1.2 Definizione Formale del Tiling Problem

Definizione 1 (Input del Tiling Problem)**.** *L'input per il problema del Tiling è un sistema di tiling* $T = (D, D_0, H, V)$*, dove:*

- D*: un insieme finito di* ***tipi*** *di piastrella.*
- $D_0 \in D$*: il tipo di piastrella che deve essere ammesso e posizionato nella cella di origine* $(0, 0)$*.*
- $H \subseteq D \times D$*: l'insieme delle* ***regole di affiancamento orizzontale****. Una coppia* $(d_1, d_2) \in H$ *indica che una piastrella di tipo* d_1 *può essere posizionata immediatamente a sinistra di una piastrella di tipo* d_2*.*
- $V \subseteq D \times D$*: l'insieme delle* ***regole di affiancamento verticale****. Una coppia* $(d_1, d_2) \in V$ *indica che una piastrella di tipo* d_1 *può essere posizionata immediatamente sotto una piastrella di tipo* d_2*.*

Definizione 2 (Tiling)**.** *Un* tiling *(o tassellamento) per un sistema* T *è una funzione* $f : \mathbb{N} \times \mathbb{N} \to D$ *tale che:*

- $f(0, 0) = D_0$ *(la piastrella iniziale è nel punto di origine).*
- *Per ogni* $m, n \in \mathbb{N}$*:* $(f(m, n), f(m + 1, n)) \in H$ *(rispetto delle regole di affiancamento orizzontale).*
- *Per ogni* $m, n \in \mathbb{N}$*:* $(f(m, n), f(m, n + 1)) \in V$ *(rispetto delle regole di affiancamento verticale).*

Il problema del Tiling chiede se per un dato sistema T *esista una tale funzione* f*.*

1.3 Indecidibilità del Tiling Problem: Riduzione da HALT_ϵ^C

Dimostriamo che il Tiling Problem è indecidibile, riducendo HALT_ϵ^C (il problema di determinare se una Macchina di Turing si **non** arresta sulla stringa vuota ϵ) al Tiling Problem.

Teorema 1. *$\text{HALT}_\epsilon^C \leq_m$ Tiling. Poiché HALT_ϵ^C è indecidibile, ne segue che anche Tiling è indecidibile.*

1.3.1 Idea della Riduzione

L'idea è codificare le computazioni di una Macchina di Turing M sul nastro semi-infinito (senza perdere generalità) utilizzando le piastrelle. Ogni "riga" di piastrelle nel tiling (ovvero, le piastrelle con lo stesso valore di n) rappresenterà una configurazione della macchina di Turing ad un determinato passo. I bordi orizzontali tra le righe di piastrelle codificheranno le configurazioni successive della MT.

La MT M non si arresta su ϵ se e solo se è possibile piastrellare l'intero quadrante.

1.3.2 Costruzione della Macchina di Turing N (funzione G)

Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ una Macchina di Turing (MT) data in input a G. La funzione G costruirà un sistema di tiling $T = (D, D_0, H, V)$. Useremo "etichette" per i colori delle porzioni di piastrella, che saranno simboli del nastro, stati della MT o combinazioni di essi. Il nastro è semi-infinito a destra.

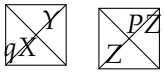
Le piastrelle avranno tipi $d \in D$ definiti per simulare il comportamento di M:

1. **Piastrelle di Inerzia (Simboli del Nastro Lontani dalla Testina):** Queste piastrelle rappresentano le porzioni del nastro in cui la testina non è presente e quindi il contenuto del nastro rimane invariato. Per ogni simbolo $X \in \Gamma$, definiamo una piastrella del tipo:

   Queste piastrelle assicurano che un simbolo X sul bordo inferiore venga replicato sul bordo superiore, simulando l'inerzia del nastro. Il "colore" del bordo superiore-destro deve combaciare con quello inferiore-sinistro della piastrella adiacente, e così via.

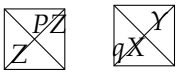
2. **Piastrelle di Transizione (Spostamento della Testina a Destra):** Queste piastrelle simulano una transizione $\delta(q, X) = (p, Y, R)$. Una tale transizione significa che la MT, nello stato q leggendo X, si sposta nello stato p, scrive Y e sposta la testina a destra. Queste transizioni coinvolgono due celle del nastro. Per ogni $q, p \in Q \setminus \{q_{acc}, q_{rej}\}$ (stati non terminali) e per ogni $X, Y, Z \in \Gamma$:

   (Le etichette interne alle piastrelle sono per chiarezza espositiva; i "colori" effettivi sono sulle porzioni.) Le etichette dei "colori" sui bordi saranno:

   - Bordo inferiore della prima piastrella: qX (stato q e simbolo letto X).
   - Bordo superiore della prima piastrella: Y (simbolo scritto Y).
   - Bordo inferiore della seconda piastrella: Z (simbolo successivo Z).
   - Bordo superiore della seconda piastrella: pZ (stato p e testina su Z).
   - Bordo verticale tra le due piastrelle: $p \to R$ (simbola il movimento a destra dello stato p).

3. **Piastrelle di Transizione (Spostamento della Testina a Sinistra):** Simulano una transizione $\delta(q, X) = (p, Y, L)$. La MT, nello stato q leggendo X, si sposta nello stato p, scrive Y e sposta la testina a sinistra. Per ogni $q, p \in Q \setminus \{q_{acc}, q_{rej}\}$ e per ogni $X, Y, Z \in \Gamma$:



   Le etichette dei "colori" sui bordi saranno:

   - Bordo inferiore della prima piastrella: Z (simbolo precedente Z).
   - Bordo superiore della prima piastrella: pZ (stato p e testina su Z).
   - Bordo inferiore della seconda piastrella: qX (stato q e simbolo letto X).
   - Bordo superiore della seconda piastrella: Y (simbolo scritto Y).
   - Bordo verticale tra le due piastrelle: $p \to L$ (simbola il movimento a sinistra dello stato p).

4. **Piastrella Iniziale (D_0):** Questa piastrella forza la prima riga a codificare la configurazione iniziale di M su ϵ. q_0 è lo stato iniziale e B è il simbolo di blank. La piastrella D_0 è:

   I colori sui bordi della piastrella D_0 impongono la configurazione iniziale. Le piastrelle adiacenti a destra saranno quelle di tipo 1 che codificano B (blank).

1.3.3 Dimostrazione del Teorema

Dobbiamo dimostrare che $M \in \text{HALT}_\epsilon^C \iff T$ ammette un tiling.

Direzione 1: $M \in \mathbf{HALT}_\epsilon^C \implies T$ **ammette un tiling** Se M non si arresta su ϵ, significa che la sua computazione su ϵ è infinita, generando una sequenza infinita di configurazioni: $C_0, C_1, C_2, \ldots$. Il sistema di tiling T è stato costruito appositamente per simulare queste configurazioni.

- La piastrella D_0 posizionata in $(0, 0)$ forza il bordo superiore della prima riga a codificare la configurazione iniziale C_0.
- Le piastrelle di tipo 1 (inerzia) coprono tutte le posizioni del nastro che non contengono la testina.
- Le piastrelle di tipo 2 e 3 (transizione) replicano l'effetto della funzione di transizione di M. Per ogni passo di computazione di M, è possibile trovare una riga di piastrelle che codifica la configurazione successiva C_{i+1} a partire dalla configurazione C_i (codificata dalla riga inferiore).

Poiché M non si arresta, genera infinite configurazioni. Grazie alla costruzione delle piastrelle, possiamo posizionare infinite righe di piastrelle, una per ogni configurazione, e ciascuna riga sufficientemente lunga per coprire le celle necessarie della configurazione. Di conseguenza, è possibile piastrellare l'intero quadrante.

Direzione 2: T **ammette un tiling** $\implies M \in \mathbf{HALT}_\epsilon^C$ Supponiamo che T ammetta un tiling f.

- Il tiling f impone che la prima riga di piastrelle ($n = 0$) rappresenti la configurazione iniziale C_0 di M su ϵ, grazie a D_0.
- La costruzione delle piastrelle (tipi 1, 2, 3) è tale che ogni riga di piastrelle $f(m, n)$ può essere validamente posizionata sopra una riga $f(m, n-1)$ solo se $f(m, n)$ codifica una configurazione che è il risultato dell'applicazione di un passo di transizione di M alla configurazione codificata da $f(m, n-1)$.
- Crucialmente, le piastrelle di transizione sono state definite solo per stati **non terminali**. Non esistono piastrelle che possano simulare una transizione verso uno stato di accettazione o rifiuto, né che possano estendere una computazione che ha già raggiunto uno stato finale.

Se esiste un tiling che copre l'intero quadrante (infinitamente in alto), significa che M può generare una sequenza infinita di configurazioni senza mai raggiungere uno stato terminale. Pertanto, M non si arresta su ϵ.

1.4 Posizionamento di Tiling nelle Classi di Calcolabilità

Il problema del Tiling è indecidibile. In particolare, è un linguaggio in coRE.

- **Perché non è in R?** L'abbiamo appena dimostrato tramite riduzione da HALT_ϵ^C. Se Tiling fosse in R, allora HALT_ϵ^C sarebbe in R, il che è falso.
- **Perché è in coRE?** Per dimostrare che Tiling è in coRE, dobbiamo mostrare che il suo complemento (cioè, il problema di determinare se un sistema di tiling *non* ammette un tiling) è in R. Questo significa che se non è possibile piastrellare, dobbiamo essere in grado di dirlo in tempo finito. Per un dato sistema di tiling, si può esplorare un numero finito di configurazioni di piastrellamento fino ad una certa profondità e larghezza. Se non si riesce a completare il piastrellamento entro un certo limite, o se si raggiunge una configurazione che blocca ulteriori estensioni, si può dire "no". Poiché il numero di tipi di piastrelle è finito, si può sistematicamente esplorare tutte le possibilità (ad esempio, con un algoritmo di backtracking). Se l'intero spazio di ricerca (finito) viene esplorato senza trovare un tiling completo, si può rispondere "no" in tempo finito.

2 Classi di Calcolabilità: R e coRE

Finora, i problemi indecidibili che abbiamo incontrato mostrano una certa "asimmetria".

2.1 Richiamo: R e RE

Definizione 3 (Classe R)**.** *Un linguaggio L appartiene alla classe **R** (ricorsivo o decidibile) se esiste una Macchina di Turing che si arresta su ogni input e decide se l'input appartiene a L. In altre parole, possiamo rispondere "SÌ" in tempo finito se l'input è nel linguaggio, e "NO" in tempo finito se non lo è.*

Definizione 4 (Classe RE)**.** *Un linguaggio L appartiene alla classe **RE** (ricorsivamente enumerabile) se esiste una Macchina di Turing che accetta tutti gli input in L e può non arrestarsi per gli input non in L. In altre parole, possiamo rispondere "SÌ" in tempo finito se l'input è nel linguaggio, ma non abbiamo garanzie di risposta (potremmo loopare) se non lo è.*

2.2 Introduzione: coRE

Definizione 5 (Classe coRE)**.** *Un linguaggio L appartiene alla classe **coRE** se il suo complemento $\overline{L}$ appartiene a RE. Se L ∈ coRE, significa che esiste una Macchina di Turing che si arresta e rifiuta tutti gli input in L (ovvero, accetta tutti gli input non in L). In altre parole, possiamo rispondere "NO" in tempo finito se l'input non è nel linguaggio, ma non abbiamo garanzie di risposta (potremmo loopare) se lo è.*

Nota 1. *Molti dei problemi indecidibili che abbiamo visto finora (ad esempio, $\overline{L_U}$, $\overline{L_{NE}}$, $\overline{HALT}$, $\overline{HALT_\epsilon}$, Tiling) si trovano in coRE ma non in RE.*

- $L_U \in RE$, $\overline{L_U} \in coRE$.
- $L_{NE} \in RE$, $\overline{L_{NE}} \in coRE$.
- $HALT \in RE$, $\overline{HALT} \in coRE$.
- $HALT_\epsilon \in RE$, $\overline{HALT_\epsilon} \in coRE$.
- *Tiling ∈ coRE (come discusso sopra).*

Questi linguaggi sono "sbilanciati": hanno un "lato facile" (o il SÌ o il NO può essere confermato in tempo finito).

2.3 Relazioni tra le Classi

Teorema 2. *$RE \cap coRE = R$.*

Dimostrazione. **Parte 1: RE ∩ coRE ⊆ R** Sia L un linguaggio tale che $L \in \text{RE} \cap \text{coRE}$. Poiché $L \in \text{RE}$, esiste una MT M_1 che accetta L. Per ogni $w \in L$, $M_1(w)$ si arresta e accetta. Per $w \notin L$, $M_1(w)$ o si arresta e rifiuta o entra in loop. Poiché $L \in \text{coRE}$, per definizione $\overline{L} \in \text{RE}$. Quindi esiste una MT M_2 che accetta $\overline{L}$. Per ogni $w \notin L$ (cioè $w \in \overline{L}$), $M_2(w)$ si arresta e accetta. Per $w \in L$, $M_2(w)$ o si arresta e rifiuta o entra in loop.

Per dimostrare che $L \in \text{R}$, costruiamo una nuova MT $M_{decider}$ che decide L:

```
M_decider(w):
  Simula M_1(w) e M_2(w) in parallelo.
  Se M_1(w) accetta, allora M_decider accetta.
  Se M_2(w) accetta, allora M_decider rifiuta.
```

Poiché w deve essere o in L o in $\overline{L}$, una delle due simulazioni si arresterà e accetterà. $M_{decider}$ si arresterà sempre e darà una risposta (accetta se $w \in L$, rifiuta se $w \notin L$). Dunque $L \in \text{R}$.

Parte 2: R ⊆ RE ∩ coRE Sia $L \in \text{R}$. Poiché $L \in \text{R}$, esiste una MT M che decide L. Questa MT si arresta sempre. Dunque M è anche una macchina che accetta L, quindi $L \in \text{RE}$. Inoltre, se M decide L, possiamo costruire una macchina M' che decide $\overline{L}$ semplicemente invertendo le risposte di M. Poiché M' decide $\overline{L}$, $\overline{L} \in \text{R}$. E dato che $\text{R} \subseteq \text{RE}$, allora $\overline{L} \in \text{RE}$. Per definizione, questo significa che $L \in \text{coRE}$. Quindi, se $L \in \text{R}$, allora $L \in \text{RE}$ e $L \in \text{coRE}$, da cui $L \in \text{RE} \cap \text{coRE}$. □

Teorema 3. *$R \neq RE$ e $R \neq coRE$.*

Dimostrazione. Sappiamo che $\text{HALT} \in \text{RE}$ ma $\text{HALT} \notin \text{R}$. Questo dimostra che $\text{RE} \neq \text{R}$. Inoltre, sappiamo che $\overline{\text{HALT}} \in \text{coRE}$ ma $\overline{\text{HALT}} \notin \text{R}$. Questo dimostra che $\text{coRE} \neq \text{R}$. □

Teorema 4. *$RE \neq coRE$.*

Dimostrazione. Supponiamo per assurdo che RE = coRE. Allora, poiché HALT $\in$ RE (e per l'assunto RE = coRE), avremmo HALT $\in$ coRE. Se HALT $\in$ coRE, allora per definizione $\overline{\text{HALT}} \in$ RE. Ma HALT $\in$ RE e $\overline{\text{HALT}} \in$ RE implica, per il teorema precedente, che HALT $\in$ RE $\cap$ coRE = R. Questo è una contraddizione, poiché sappiamo che HALT $\notin$ R. Dunque, l'assunto RE = coRE è falso, e RE $\neq$ coRE. □

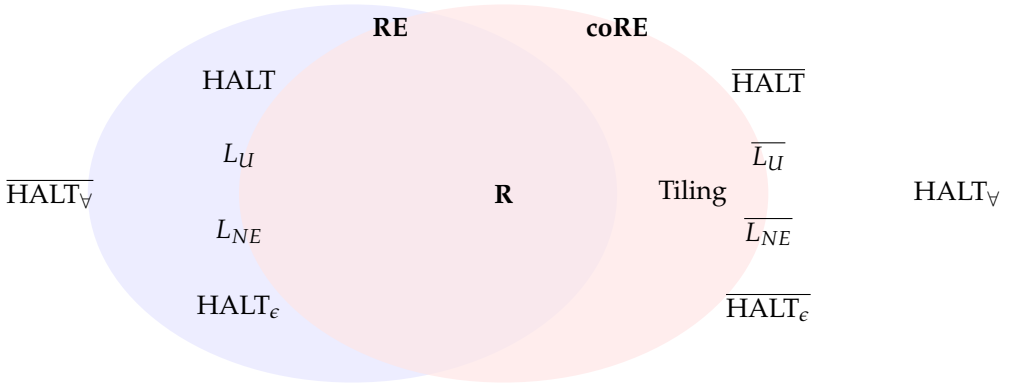


Figura 1: Diagramma delle classi di calcolabilità RE, coRE e R.

3 Problemi Esterni a RE e coRE

I problemi indecidibili che abbiamo visto finora appartengono a RE o a coRE (ovvero, hanno un "lato facile" per cui si può dire SÌ o NO in tempo finito). La domanda è: esistono problemi che non appartengono né a RE né a coRE? Ovvero, problemi per i quali non siamo in grado di dare garanzie di risposta (SÌ o NO) in tempo finito?

Definizione 6 ($\text{HALT}_\forall$)**.** *Sia* $HALT_\forall = \{\langle M \rangle \mid M$ *si arresta su ogni input* $w\}$.

Nota 2. *Intuitivamente:*

- ***Perché*** $\boldsymbol{HALT_\forall \notin RE?}$ *Per determinare se una macchina si arresta su *ogni* input, dovremmo simulare la macchina su tutti gli input possibili, che sono infiniti. Se la macchina si arresta su tutti, non potremmo mai finire di verificarlo.*
- ***Perché*** $\boldsymbol{HALT_\forall \notin}$ ***coRE?*** *Se* $HALT_\forall \in coRE$*, allora* $\overline{HALT_\forall} \in RE$*. Ma* $\overline{HALT_\forall} = \{\langle M \rangle \mid M$ *non si arresta su almeno un input* $w\}$*. Per determinare se M non si arresta su *almeno un* input, dovremmo cercare questo input. Se M non si arresta su un certo w, la simulazione di M su w non terminerebbe, e quindi non potremmo dire "SÌ" in tempo finito.*

Quindi, $HALT_\forall$ *sembra essere un candidato a non stare né in RE né in coRE.*
Per dimostrarlo formalmente, useremo il seguente Lemma sulle riduzioni:

Lemma 1. *Siano A e B due linguaggi.*

- *Se* $A \leq_m B$ *e* $B \in R$*, allora* $A \in R$*. (Già dimostrato)*
- *Se* $A \leq_m B$ *e* $A \notin RE$*, allora* $B \notin RE$*. (Già dimostrato)*

- *Se $A \leq_m B$ e $A \notin$ coRE, allora $B \notin$ coRE.*

Dimostrazione (ultimo punto). Supponiamo per assurdo che $A \leq_m B$ e $B \in$ coRE, ma $A \notin$ coRE. Se $B \in$ coRE, allora $\overline{B} \in$ RE. Poiché $A \leq_m B$, esiste una funzione calcolabile f tale che $w \in A \iff f(w) \in B$. Questo implica che $w \notin A \iff f(w) \notin B$, ovvero $w \in \overline{A} \iff f(w) \in \overline{B}$. Quindi $\overline{A} \leq_m \overline{B}$. Dato che $\overline{B} \in$ RE e $\overline{A} \leq_m \overline{B}$, per la seconda parte del Lemma ($A \leq_m B$ e $B \in$ RE $\implies A \in$ RE), ne segue che $\overline{A} \in$ RE. Se $\overline{A} \in$ RE, per definizione $A \in$ coRE. Questo contraddice la nostra assunzione $A \notin$ coRE. Quindi l'assunzione è falsa, e se $A \leq_m B$ e $A \notin$ coRE, allora $B \notin$ coRE. □

3.0.1 Dimostrare $\text{HALT}_\forall \notin$ coRE

Per dimostrare $\text{HALT}_\forall \notin$ coRE, useremo la riduzione $\text{HALT}_\epsilon \leq_m \text{HALT}_\forall$. Sappiamo che $\text{HALT}_\epsilon \in$ RE ma $\text{HALT}_\epsilon \notin$ coRE (infatti $\overline{\text{HALT}_\epsilon} \in$ RE).

Sia f la funzione di riduzione che prende una MT M e produce una MT $N = f(M)$.

```
# Pseudocodice per la macchina N = f(M)
def N(input_w):
    # N ignora completamente il proprio input_w
    # e si concentra solo sulla simulazione di M su stringa vuota.

    # Simula M sulla stringa vuota (epsilon)
    # Si assume che una simulazione di M su epsilon possa essere eseguita
    # e che N possa osservarne il comportamento.
    if M.halts_on_epsilon():
        # Se M si arresta su epsilon, N si arresta (ad esempio, accetta)
        return "Accetta"
    else:
        # Se M non si arresta su epsilon, N non si arresta
        # N entra in un loop infinito
        while True:
            pass # Loop infinito
```

Dimostrazione. Dobbiamo mostrare che $M \in \text{HALT}_\epsilon \iff N \in \text{HALT}_\forall$.

Direzione 1: $M \in \textbf{HALT}_\epsilon \implies N \in \textbf{HALT}_\forall$ Se $M \in \text{HALT}_\epsilon$, significa che M si arresta sulla stringa vuota ϵ. Per costruzione, la macchina N, indipendentemente dal suo input w, simula M su ϵ. Poiché M si arresta su ϵ, la simulazione di M su ϵ termina in un numero finito di passi, e N si arresta. Questo è vero per ogni w dato a N. Dunque, N si arresta su ogni input w. Per definizione, $N \in \text{HALT}_\forall$.

Direzione 2: $M \notin \textbf{HALT}_\epsilon \implies N \notin \textbf{HALT}_\forall$ Se $M \notin \text{HALT}_\epsilon$, significa che M non si arresta sulla stringa vuota ϵ. Per costruzione, la macchina N, indipendentemente dal suo input w, simula M su ϵ. Poiché M non si arresta su ϵ, la simulazione di M su ϵ non termina mai, e N entra in un loop infinito. Questo è vero per ogni w dato a N. Dunque, N non si arresta su nessun input w. Per definizione, $N \notin \text{HALT}_\forall$.

La riduzione $\text{HALT}_\epsilon \leq_m \text{HALT}_\forall$ è valida. Poiché $\text{HALT}_\epsilon \notin$ coRE, per il Lemma, segue che $\text{HALT}_\forall \notin$ coRE. □

3.0.2 Dimostrare $\text{HALT}_\forall \notin$ RE

Per dimostrare $\text{HALT}_\forall \notin$ RE, useremo la riduzione $\overline{\text{HALT}_\epsilon} \leq_m \text{HALT}_\forall$. Sappiamo che $\overline{\text{HALT}_\epsilon} \notin$ RE.

Sia f la funzione di riduzione che prende una MT M e produce una MT $N = f(M)$.

```python
# Pseudocodice per la macchina N = f(M)
def N(input_w):
    # Usa la lunghezza di input_w come un contatore di passi.
    t = len(input_w)

    # Simula M sulla stringa vuota (epsilon) per un massimo di t passi.
    simulazione_m_terminata = M.simulate_on_epsilon(max_steps=t)

    if simulazione_m_terminata:
        # Se M si arresta su epsilon entro t passi, N entra in un loop infinito.
        while True:
            pass # Loop infinito
    else:
        # Se M non si arresta su epsilon entro t passi, N si arresta (ad esempio, rifiuta).
        return "Rifiuta"
```

Dimostrazione. Dobbiamo mostrare che $M \in \overline{\text{HALT}_\epsilon} \iff N \in \text{HALT}_\forall$.

Direzione 1: $M \in \overline{\textbf{HALT}_\epsilon} \implies N \in \textbf{HALT}_\forall$ Se $M \in \overline{\text{HALT}_\epsilon}$, significa che M non si arresta sulla stringa vuota ϵ. Poiché M non si arresta mai su ϵ, la simulazione di M su ϵ per un numero finito di passi $t = |w|$ (la lunghezza dell'input di N) non raggiungerà mai uno stato di arresto. Di conseguenza, per qualsiasi input w dato a N, la condizione `simulazione_m_terminata` sarà sempre falsa. Quindi N si arresterà sempre (rifiutando). Dunque, N si arresta su ogni input w. Per definizione, $N \in \text{HALT}_\forall$.

Direzione 2: $M \notin \overline{\textbf{HALT}_\epsilon} \implies N \notin \textbf{HALT}_\forall$ Se $M \notin \overline{\text{HALT}_\epsilon}$, significa che M si arresta sulla stringa vuota ϵ. Sia k il numero di passi dopo i quali M si arresta su ϵ.

- Se N riceve un input w tale che $|w| < k$: N simulerà M su ϵ per $|w|$ passi. Poiché $|w| < k$, M non si sarà ancora arrestata. La condizione `simulazione_m_terminata` sarà falsa, e N si arresterà (rifiutando).
- Se N riceve un input w tale che $|w| \geq k$: N simulerà M su ϵ per $|w|$ passi. Poiché $|w| \geq k$, M si arresterà entro k passi. La condizione `simulazione_m_terminata` sarà vera, e N entrerà in un loop infinito.

Poiché N si arresta su alcuni input (quelli con lunghezza $< k$) e non si arresta su altri input (quelli con lunghezza $\geq k$), N non si arresta su *ogni* input. Dunque, $N \notin \text{HALT}_\forall$.

La riduzione $\overline{\text{HALT}_\epsilon} \leq_m \text{HALT}_\forall$ è valida. Poiché $\overline{\text{HALT}_\epsilon} \notin \text{RE}$, per il Lemma, segue che $\text{HALT}_\forall \notin \text{RE}$. □

3.1 Conclusione: $\text{HALT}_\forall$ è Esterno a RE e coRE

Abbiamo dimostrato che $\text{HALT}_\forall \notin \text{RE}$ e $\text{HALT}_\forall \notin \text{coRE}$. Ciò significa che $\text{HALT}_\forall$ è un problema ancora più "difficile" dei problemi indecidibili che abbiamo visto finora. Non esiste un algoritmo che possa confermare l'appartenenza di una macchina a $\text{HALT}_\forall$ in tempo finito (perché non sta in RE), né un algoritmo che possa confermare la non appartenenza in tempo finito (perché non sta in coRE). Il suo complemento $\overline{\text{HALT}_\forall}$ si trova anch'esso fuori da RE e coRE.