

Soluzione Esercizi 01/06

Automi e Linguaggi Formali – Seconda Parte

Indecidibili

Mostra che A_{TM} non è riducibile mediante funzione a E_{TM} . In altre parole, dimostra che non esiste una funzione calcolabile che riduce A_{TM} ad E_{TM} .

Per dimostrare che l'ATM non è una mappatura riducibile all'ETM, possiamo utilizzare una prova per contraddizione. Ciò significa che assumiamo l'opposto di ciò che vogliamo dimostrare e dimostriamo che ciò porta a una contraddizione. Ecco i passi che possiamo compiere:

Assumiamo che esista una funzione computabile f che riduce ATM a ETM.

Possiamo usare il fatto che ATM è indecidibile e ETM è riconoscibile per dimostrare che questa ipotesi porta a una contraddizione.

Poiché ATM è indecidibile, non esiste una macchina di Turing che possa decidere se una data macchina accetta un dato input.

D'altra parte, ETM è riconoscibile, il che significa che esiste una macchina di Turing che accetta tutte le stringhe di input accettate da un'altra macchina di Turing.

Se esiste una funzione computabile f che riduce ATM a ETM, allora possiamo usare questa funzione per decidere se una data macchina accetta un dato input trasformando il problema nel linguaggio riconoscibile di ETM.

Ciò significa che ATM sarebbe decidibile, il che contraddice il fatto che è indecidibile.

Pertanto, la nostra ipotesi che esista una funzione computabile f che riduce ATM a ETM deve essere falsa e abbiamo dimostrato che ATM non è una mappatura riducibile a ETM.

Sia A il linguaggio che contiene solo ed unicamente la stringa s ,

$$s = \begin{cases} 0 & \text{se la vita non sarà mai trovata su Marte} \\ 1 & \text{se un giorno la vita sarà trovata su Marte} \end{cases}$$

A è un linguaggio decidibile? Giustificare la risposta. Ai fini del problema, assumere che la questione se la vita sarà trovata su Marte ammetta una risposta non ambigua Si/No.

Per determinare se il linguaggio A è decidibile, dobbiamo verificare se esiste un algoritmo che può determinare in maniera definitiva se una stringa fa parte del linguaggio A o meno.

Nel caso specifico, il linguaggio A contiene solo ed esclusivamente la stringa s , che può assumere il valore di 0 o 1, a seconda che la vita sia o non sia stata trovata su Marte. Poiché l'informazione sulla vita su Marte è definita come non ambigua (Si/No), il linguaggio A è formato da una sola stringa.

Essendo il linguaggio A composto da una sola stringa, è possibile costruire un algoritmo che verifica se una stringa data corrisponde esattamente alla stringa s . L'algoritmo può semplicemente confrontare la stringa in input con la stringa s e restituire "Si" se corrispondono e "No" altrimenti. Questo algoritmo termina sempre e determina in modo definitivo l'appartenenza o meno di una stringa al linguaggio A .

Quindi, il linguaggio A è decidibile perché esiste un algoritmo che può determinare in modo definitivo se una stringa fa parte del linguaggio o no.

- Se L_1 e L_2 sono due linguaggi, allora consideriamo

$$INTERLACE(L_1, L_2) = \{w_1v_1w_2v_2 \dots w_nv_n \mid w_1w_2 \dots w_n \in L_1, v_1v_2 \dots v_n \in L_2\}.$$

Ad esempio, se $abc \in L_1$ e $123 \in L_2$, allora $a1b2c3 \in INTERLACE(L_1, L_2)$

Dimostrare che il linguaggio è decidibile e Turing-riconoscibile.

Per dimostrare che il linguaggio $INTERLACE(L_1, L_2)$ è decidibile, dobbiamo dimostrare che esiste un algoritmo in grado di determinare, per un dato input, se la stringa in ingresso appartiene o meno a $INTERLACE(L_1, L_2)$.

La decidibilità è una proprietà più forte della riconoscibilità, quindi se possiamo dimostrare che $INTERLACE(L_1, L_2)$ è riconoscibile, implica automaticamente che è decidibile.

Per dimostrare che $INTERLACE(L_1, L_2)$ è riconoscibile, possiamo progettare una macchina di Turing che riconosca questo linguaggio. La macchina di Turing prende una stringa in ingresso e controlla se può essere divisa in due parti, dove la prima parte appartiene a L_1 e la seconda a L_2 . La macchina itera tutte le posizioni possibili per dividere l'input e verifica se entrambe le parti soddisfano i rispettivi linguaggi L_1 e L_2 . Se trova una divisione valida, accetta l'input, altrimenti continua la ricerca.

Questa macchina di Turing riconosce $INTERLACE(L_1, L_2)$ perché può potenzialmente trovare una divisione valida per qualsiasi stringa del linguaggio. Tuttavia, può funzionare all'infinito per stringhe che non appartengono al linguaggio.

Ora, per dimostrare che $INTERLACE(L_1, L_2)$ è decidibile, dobbiamo dimostrare che la stessa macchina di Turing può arrestarsi anche per input che non appartengono al linguaggio. Possiamo modificare la macchina di Turing aggiungendo un numero finito di passi per garantire che si arresti per input non appartenenti a $INTERLACE(L_1, L_2)$. Ad esempio, possiamo aggiungere un passo per rifiutare l'input se non si divide in due parti o se una delle due parti non appartiene al rispettivo linguaggio.

Modificando la macchina di Turing in modo che si arresti per tutti gli input, indipendentemente dal fatto che appartengano o meno a $INTERLACE(L_1, L_2)$, abbiamo dimostrato che il linguaggio è decidibile.

In sintesi, possiamo dimostrare che $INTERLACE(L_1, L_2)$ è riconoscibile progettando una macchina di Turing che cerca iterativamente le suddivisioni valide, e possiamo dimostrare che è decidibile modificando la macchina di Turing in modo che si arresti per tutti gli input, indipendentemente dal fatto che appartengano o meno al linguaggio.

- Questo linguaggio è decidibile, riconoscibile o irriconoscibile?

“Il linguaggio L consiste in tutte le descrizioni di macchine di Turing M , per le quali il linguaggio accettato da M è finito”.

Il linguaggio L , che consiste in tutte le descrizioni di macchine di Turing M per le quali il linguaggio accettato da M è finito, è un linguaggio decidibile.

Per dimostrare che il linguaggio L è decidibile, possiamo descrivere un algoritmo che decide l'appartenenza a L . Data una descrizione di macchina di Turing M , possiamo simulare l'esecuzione di M su tutti i possibili input e verificare se il linguaggio accettato da M è finito. A tale scopo, possiamo tenere traccia delle stringhe accettate da M durante la simulazione. Se incontriamo una stringa che non rientra nel linguaggio accettato da M , possiamo interrompere immediatamente la simulazione e rifiutare M . Se invece scopriamo che M accetta tutte le stringhe fino a una certa lunghezza, ma non di più, possiamo interrompere la simulazione e accettarla.

Poiché possiamo simulare l'esecuzione di M su tutti i possibili input e determinare se il linguaggio accettato da M è finito o meno, l'algoritmo si arresterà sempre e fornirà una risposta definitiva per qualsiasi descrizione di macchina di Turing M. Pertanto, il linguaggio L è decidibile.

Vale la pena notare che il linguaggio L è anche riconoscibile. Possiamo progettare una macchina di Turing che simuli l'esecuzione di M su tutti i possibili input e accetti se M si arresta su qualsiasi input. Questa macchina di Turing riconoscerà il linguaggio L perché si arresterà e accetterà per tutte le macchine di Turing il cui linguaggio accettato è finito. Tuttavia, funzionerà all'infinito per le macchine di Turing il cui linguaggio accettato non è finito.

In conclusione, il linguaggio L, costituito da tutte le descrizioni di macchine di Turing per le quali il linguaggio accettato è finito, è decidibile e riconoscibile.

- Sia data una funzione $f(n)$ pari a:
 - 1 se 0^n appare nella rappresentazione decimale di π
 - 0 altrimenti.

Si provi che il linguaggio è indecidibile.

Per dimostrare che il linguaggio descritto da $f(n) = 1$ se " 0^n " appare nella rappresentazione decimale di π , 0 altrimenti, è indecidibile, possiamo utilizzare un ragionamento di diagonalizzazione.

Supponiamo per assurdo che esista un algoritmo (una macchina di Turing) che può decidere se " 0^n " appare nella rappresentazione decimale di π . Questo algoritmo prenderebbe in input un numero n e restituirebbe 1 se " 0^n " appare in π , altrimenti restituirebbe 0.

Ora consideriamo un'altra macchina di Turing che utilizza l'algoritmo sopra descritto come sotto-routine. Questa macchina di Turing genera una tabella che elenca tutti i numeri naturali e, per ciascun numero n, esegue l'algoritmo per determinare se " 0^n " appare in π . Se l'algoritmo restituisce 1, la macchina di Turing scrive 0 nella cella corrispondente nella tabella; se l'algoritmo restituisce 0, la macchina di Turing scrive 1.

Ora, consideriamo la diagonale della tabella che abbiamo ottenuto. Otteniamo una sequenza infinita di bit che differisce in almeno una posizione da ciascuna delle righe della tabella. Ad esempio, se la riga i-esima della tabella ha il bit i-esimo uguale a 0, il bit i-esimo della diagonale sarà 1, e viceversa.

Ora, consideriamo il numero naturale rappresentato dalla diagonale ottenuta. Questo numero differisce da ciascun numero naturale nella tabella, perché differisce almeno in una posizione da ciascuna delle righe. Quindi, se passiamo questo numero alla sotto-routine dell'algoritmo descritto sopra, otterremo un risultato diverso da ciascuna delle righe della tabella.

Questo ci porta a una contraddizione, poiché abbiamo supposto che l'algoritmo possa decidere se " 0^n " appare nella rappresentazione decimale di π . La diagonale che abbiamo costruito dimostra che non è possibile decidere correttamente per tutti i numeri naturali se " 0^n " appare o meno in π .

Di conseguenza, possiamo concludere che il linguaggio descritto da $f(n)$ è indecidibile, poiché non esiste un algoritmo che possa decidere correttamente per tutti i numeri naturali se " 0^n " appare o meno nella rappresentazione decimale di π .

- Sia $F = \{\langle M \rangle \mid M \text{ è un TM che si ferma per ogni input in un massimo di } 50 \text{ passi}\}$.
 F può essere considerato decidibile oppure no?

Il linguaggio F, definito come l'insieme di tutte le codifiche $\langle M \rangle$ delle macchine di Turing che terminano entro 50 passi per ogni input, può essere considerato decidibile.

Per dimostrare che F è decidibile, possiamo costruire un algoritmo che prende in input la codifica $\langle M \rangle$ di una macchina di Turing M e simula l'esecuzione di M su ogni possibile input per un massimo di 50 passi. Se M termina entro 50 passi per ogni input, allora l'algoritmo restituisce "Si"; altrimenti, restituisce "No".

L'algoritmo descritto sopra termina sempre in un numero finito di passi, poiché viene effettuata una simulazione di M su tutti gli input per un massimo di 50 passi ciascuno. Inoltre, l'algoritmo fornisce una risposta definitiva per ogni input $\langle M \rangle$, indicando se M termina entro 50 passi per ogni input o meno.

Pertanto, poiché esiste un algoritmo che decide correttamente l'appartenenza di ogni input al linguaggio F , il linguaggio F è decidibile.

In conclusione, il linguaggio F , che rappresenta l'insieme delle codifiche delle macchine di Turing che terminano entro 50 passi per ogni input, è decidibile.

- Si consideri questo problema:

Dato un insieme di domino, in cui ogni domino è costituito da una stringa superiore e da una stringa inferiore, esiste un modo per disporre questi domino in una sequenza tale che la concatenazione delle stringhe superiori corrispondenti dia lo stesso risultato della concatenazione delle stringhe inferiori corrispondenti?

Ad esempio, consideriamo il seguente insieme di domino:

$$D1: \{a, abc\}$$

$$D2: \{ab, b\}$$

$$D3: \{bc, c\}$$

La domanda è: è possibile disporre questi domino in una sequenza tale che la concatenazione delle stringhe superiori ($aabbcc$) sia uguale alla concatenazione delle stringhe inferiori ($abcbcc$)?

Per dimostrare che il problema è indecidibile, utilizzeremo una riduzione di mappatura (\leq_m) dall'indecidibile Halting Problem.

Il Problema di Halting è il problema di determinare, data una macchina di Turing M e una stringa di input w , se M si ferma su w o gira all'infinito. Assumeremo che il problema di Halting sia indecidibile.

Definiamo ora la riduzione della mappatura dal Problema di Halting al problema attuale.

Data un'istanza (M, w) del Problema di Halting, costruiremo un'istanza di PCP-MOD come segue:

Costruire un insieme di domino basato sulla macchina di Turing M e sull'input w . Ogni domino rappresenterà una configurazione della macchina di Turing in uno specifico passo della sua computazione.

Per ogni configurazione della macchina di Turing, creare un domino con la stringa superiore che rappresenta lo stato della macchina di Turing e la stringa inferiore che rappresenta il contenuto del nastro in quel passo.

Inoltre, creare uno speciale domino iniziale con la stringa superiore che rappresenta lo stato iniziale della macchina di Turing e la stringa inferiore che rappresenta il contenuto iniziale del nastro.

Costruendo questo insieme di domino, abbiamo creato un'istanza di del problema che corrisponde all'istanza originale del problema di Halting.

Ora sosteniamo che la macchina di Turing originale si ferma sull'input se e solo se esiste una sequenza di domino nell'istanza di PCP-MOD che corrisponde alle stringhe superiore e inferiore.

Se la macchina di Turing si arresta in ingresso, allora esiste una sequenza di configurazioni che porta a uno stato di arresto. Questa sequenza di configurazioni corrisponde a una sequenza di domino nell'istanza PCP-MOD che può essere disposta in modo tale da far coincidere le stringhe superiore e inferiore.

Se esiste una sequenza di domino nell'istanza del nostro problema che corrisponde alle stringhe superiore e inferiore, allora questa sequenza corrisponde a una sequenza di configurazioni della macchina di Turing. Se questa sequenza porta a uno stato di arresto, significa che la macchina di Turing si arresta sull'input.

Pertanto, abbiamo dimostrato che la macchina di Turing originale si arresta in ingresso se e solo se esiste una sequenza di domino nell'istanza del nostro problema che corrisponde alle stringhe superiore e inferiore.

Poiché il problema di Halting è indecidibile e lo abbiamo ridotto al nostro, possiamo concludere che anche il nostro è indecidibile.

- Dato:

$$CF_{TM} = \{\langle M \rangle \mid M \text{ è una descrizione di una } TM \text{ e } L(M) \text{ è un linguaggio context-free}\}$$

CF_{TM} e il suo complemento sono Turing-riconoscibili oppure no?

Il linguaggio CF_TM , definito come l'insieme delle descrizioni delle macchine di Turing che riconoscono linguaggi context-free, è Turing-riconoscibile.

Per dimostrarlo, possiamo costruire una macchina di Turing che riconosce il linguaggio CF_TM . La macchina di Turing prende in input una descrizione $\langle M \rangle$ di una macchina di Turing M e procede nel seguente modo:

1. Simula l'esecuzione di M su tutte le stringhe di input possibili.
2. Se M accetta una stringa che rappresenta una grammatica context-free, la macchina di Turing accetta.
3. Altrimenti, la macchina di Turing continua a simulare l'esecuzione di M su altre stringhe di input.

Poiché ogni macchina di Turing M che riconosce un linguaggio context-free accetta almeno una descrizione corrispondente nel linguaggio CF_TM , la macchina di Turing descritta sopra riconosce il linguaggio CF_TM . Quindi, CF_TM è Turing-riconoscibile.

Per quanto riguarda il complemento di CF_TM , cioè il linguaggio che contiene tutte le descrizioni delle macchine di Turing che non riconoscono linguaggi context-free, esso non è Turing-riconoscibile.

Per dimostrarlo, possiamo utilizzare un ragionamento di riduzione. Possiamo ridurre il problema dell'arresto (Halting Problem) al complemento di CF_TM . Supponiamo per assurdo che il complemento di CF_TM sia Turing-riconoscibile. Allora potremmo utilizzare un algoritmo per riconoscere il complemento di CF_TM per decidere se una macchina di Turing M si arresta su un input w . Questo sarebbe in contraddizione con il fatto che l'Halting Problem è noto per essere non decidibile.

Quindi, possiamo concludere che il complemento di CF_TM non è Turing-riconoscibile.

In sintesi, il linguaggio CF_TM è Turing-riconoscibile, mentre il suo complemento non è Turing-riconoscibile.

Macchine di Turing ed annesse varianti

- Un enumeratore E è una TM con una stampante collegata.
 - o Il TM può inviare stringhe alla stampante.
 - o Il nastro di input è inizialmente vuoto
 - o Il linguaggio enumerato da E è l'insieme delle stringhe stampate
 - o E non può fermarsi e stampare un numero infinito di stringhe.

Dimostra che qualsiasi enumeratore può essere simulato da una macchina di Turing deterministica a nastro singolo.

Per dimostrare che qualsiasi enumeratore può essere simulato da una macchina di Turing deterministica a nastro singolo, dobbiamo costruire una macchina di Turing che emula l'enumeratore E.

L'idea principale è di utilizzare il nastro della macchina di Turing per simulare la stampante collegata all'enumeratore. Poiché la macchina di Turing può leggere e scrivere solo su un nastro singolo, dobbiamo trovare un modo per rappresentare le stringhe stampate in modo che la macchina di Turing possa leggerle e scriverle.

Possiamo utilizzare una codifica specifica per rappresentare le stringhe stampate. Ad esempio, possiamo utilizzare una sequenza di caratteri speciali come delimitatori tra le stringhe. Inizialmente, il nastro della macchina di Turing sarà vuoto.

La macchina di Turing emula l'enumeratore E nel seguente modo:

1. La macchina di Turing inizia a simulare l'enumeratore E.
2. Ogni volta che E invia una stringa alla stampante, la macchina di Turing scrive la stringa codificata sul nastro, utilizzando i delimitatori appropriati.
3. La macchina di Turing continua ad emulare E, leggendo le stringhe stampate dal nastro e reagendo di conseguenza.
4. Quando la macchina di Turing deve "stampare" una stringa, in realtà scriverà la stringa sul nastro.

Poiché la macchina di Turing può simulare l'azione dell'enumeratore E, è in grado di enumerare le stringhe come l'enumeratore originale.

La simulazione di un enumeratore da parte di una macchina di Turing deterministica a nastro singolo è possibile perché le azioni dell'enumeratore possono essere modellate come operazioni di scrittura e lettura su un nastro. Anche se l'enumeratore potrebbe non terminare, la macchina di Turing può essere progettata in modo che continui ad emulare l'enumeratore indefinitamente.

Pertanto, qualsiasi enumeratore può essere simulato da una macchina di Turing deterministica a nastro singolo mediante l'utilizzo di una codifica adeguata a rappresentare le stringhe stampate sul nastro.

- Macchina di Turing ad accesso casuale
 - o Invece di accedere ai dati sul nastro sequenziale, immaginate una TM che abbia memoria ad accesso casuale e che possa andare a qualsiasi cella del nastro in un solo passaggio. Per consentire questo, la TM dispone di registri che possono memorizzare indirizzi di memoria.
- Dimostrare che questo tipo di Turing riconosce la classe dei linguaggi riconoscibili da Turing.

Per dimostrare che una macchina di Turing ad accesso casuale riconosce la stessa classe di linguaggi riconoscibili da una macchina di Turing standard, dobbiamo dimostrare due cose:

1. Ogni linguaggio riconoscibile da una macchina di Turing standard può essere riconosciuto da una macchina di Turing ad accesso casuale.
2. Ogni linguaggio riconosciuto da una macchina di Turing ad accesso casuale può essere riconosciuto da una macchina di Turing standard.

Dimostrazione:

1. Supponiamo di avere una macchina di Turing standard M che riconosce un linguaggio L. Per dimostrare che L può essere riconosciuto da una macchina di Turing ad accesso casuale, possiamo costruire una macchina di Turing ad accesso casuale M' che simula M.

La macchina di Turing M' utilizza registri per memorizzare gli indirizzi di memoria e può accedere direttamente a qualsiasi cella del nastro in un solo passaggio. La simulazione di M' consiste nell'imitare il comportamento di M passo dopo passo, mantenendo traccia degli indirizzi di memoria necessari per accedere alle celle corrispondenti sul nastro.

Poiché M è una macchina di Turing standard che riconosce il linguaggio L, e M' simula il comportamento di M, M' riconosce anche il linguaggio L. Pertanto, ogni linguaggio riconoscibile da una macchina di Turing standard può essere riconosciuto da una macchina di Turing ad accesso casuale.

2. Supponiamo di avere una macchina di Turing ad accesso casuale M' che riconosce un linguaggio L. Per dimostrare che L può essere riconosciuto da una macchina di Turing standard, possiamo costruire una macchina di Turing standard M che simula M'.

La macchina di Turing M simula M' utilizzando un approccio sequenziale per accedere alle celle del nastro. Ogni volta che M' richiede l'accesso a una specifica cella di memoria, M scansiona sequenzialmente il nastro fino a raggiungere quella cella. Una volta raggiunta la cella, M può eseguire le operazioni richieste da M' su quella cella.

Poiché M' è una macchina di Turing ad accesso casuale che riconosce il linguaggio L, e M simula il comportamento di M', M riconosce anche il linguaggio L. Pertanto, ogni linguaggio riconosciuto da una macchina di Turing ad accesso casuale può essere riconosciuto da una macchina di Turing standard.

Pertanto, abbiamo dimostrato che una macchina di Turing ad accesso casuale riconosce la stessa classe di linguaggi riconoscibili da una macchina di Turing standard.

- Data una stringa di parentesi bilanciate, scrivere una macchina di Turing che accetti la stringa utilizzando una macchina di Turing a nastro singolo.

La simulazione deve permettere ad un compilatore di accettare il linguaggio; quindi, dalla macchina a nastro singolo, si dovrà simulare la TM che opera la gestione delle parentesi.

Per simulare una macchina di Turing a nastro singolo che accetta una stringa di parentesi bilanciate, possiamo utilizzare una macchina di Turing con due stati principali: uno stato per gestire l'apertura delle parentesi e uno stato per gestire la chiusura delle parentesi.

La macchina di Turing avrà il seguente comportamento:

Stato iniziale:

- S1 (stato di apertura)

Transizioni:

1. Se lo stato corrente è S1 (stato di apertura):

- Se il simbolo letto è una parentesi aperta "(":
 - Scrivi il simbolo "(" sul nastro e vai allo stato S1.
- Se il simbolo letto è una parentesi chiusa ")":
 - Vai allo stato S2 (stato di chiusura).

2. Se lo stato corrente è S2 (stato di chiusura):

- Se il simbolo letto è una parentesi chiusa ")":
 - Scrivi il simbolo ")" sul nastro e vai allo stato S2.
- Se il simbolo letto è una parentesi aperta "(":
 - Vai allo stato S1 (stato di apertura).

3. Se la testina della macchina di Turing raggiunge la fine della stringa:

- Se lo stato corrente è S1 (stato di apertura), accetta la stringa.
- Altrimenti, rifiuta la stringa.

Questa macchina di Turing simula il processo di apertura e chiusura delle parentesi. Inizia nello stato S1 e, ogni volta che legge una parentesi aperta "(", la scrive sul nastro e rimane nello stato S1. Quando legge una parentesi chiusa ")", passa allo stato S2. Nello stato S2, ogni volta che legge una parentesi chiusa ")", la scrive sul nastro e rimane nello stato S2. Quando legge una parentesi aperta "(", torna allo stato S1.

Se la testina raggiunge la fine della stringa, controlla lo stato corrente. Se è nello stato di apertura S1, accetta la stringa poiché tutte le parentesi sono state correttamente chiuse. Altrimenti, se è nello stato di chiusura S2, rifiuta la stringa perché ci sono parentesi aperte senza corrispondente parentesi chiusa.

Con questa simulazione, la macchina di Turing a nastro singolo può accettare correttamente le stringhe di parentesi bilanciate.

- Una macchina di Turing non deterministica multi-nastro e multi-testina è una macchina con un nastro infinito in una sola direzione. Per un dato stato e un dato simbolo di input ha almeno una scelta per muoversi (numero finito di scelte per la mossa successiva), ogni scelta ha diverse scelte del percorso che potrebbe seguire per una data stringa di input, dato però su più nastri con diverse testine.

Dimostra che qualsiasi macchina di questo tipo può essere simulata da una macchina di Turing deterministica a nastro singolo.

Per dimostrare che qualsiasi macchina di Turing non deterministica multi-nastro e multi-testina può essere simulata da una macchina di Turing deterministica a nastro singolo, possiamo utilizzare l'approccio di simulazione delle scelte non deterministiche.

Consideriamo una macchina di Turing non deterministica multi-nastro e multi-testina con k nastri e m teste. Ogni testina può leggere e scrivere sui nastri indipendentemente dalle altre teste.

Per simulare questa macchina di Turing con una macchina di Turing deterministica a nastro singolo, utilizziamo un approccio di simulazione in cui consideriamo tutte le possibili combinazioni di scelte non deterministiche che la macchina di Turing non deterministica potrebbe fare.

La simulazione avviene nel seguente modo:

1. La macchina di Turing deterministica inizia la simulazione sulla stringa di input.

2. Per ogni passo di computazione, la macchina di Turing deterministica considera tutte le possibili combinazioni di scelte che la macchina di Turing non deterministica potrebbe fare.
3. Per ogni combinazione di scelte, la macchina di Turing deterministica simula la computazione in modo sequenziale, seguendo il percorso che corrisponde a quella specifica combinazione di scelte.
4. La macchina di Turing deterministica tiene traccia dello stato corrente e delle posizioni delle teste su ogni passo di computazione.
5. Durante la simulazione, la macchina di Turing deterministica può utilizzare una codifica speciale per rappresentare le posizioni delle teste sui nastri, ad esempio utilizzando delimitatori o simboli speciali per indicare le posizioni delle teste.
6. La simulazione continua finché tutte le possibili combinazioni di scelte sono state esplorate e termina quando una delle seguenti condizioni viene soddisfatta:
 - La simulazione raggiunge uno stato finale corrispondente a una configurazione di accettazione.
 - La simulazione termina su tutti i percorsi possibili senza raggiungere una configurazione di accettazione.

Poiché la macchina di Turing deterministica considera tutte le possibili combinazioni di scelte non deterministiche e segue i percorsi corrispondenti sequenzialmente, è in grado di simulare il comportamento della macchina di Turing non deterministica su tutte le possibili computazioni in modo deterministico.

Quindi, possiamo concludere che qualsiasi macchina di Turing non deterministica multi-nastro e multi-testina può essere simulata da una macchina di Turing deterministica a nastro singolo, utilizzando l'approccio di simulazione delle scelte non deterministiche descritto sopra.

- Una macchina di Turing viene definita distruttrice quando considerato un alfabeto Σ è in grado di decidere il suo linguaggio, considerato composto da termini finiti, ed è in grado di muovere le testine riportando sempre alla stessa posizione una volta consumato tutto l'input. Quanto descritto, logicamente, può corrispondere alla deallocazione delle variabili una volta usciti dal programma. Descrivere questa macchina con una delle varianti delle Turing machines a nastro singolo e dimostrare che qual può essere simulata da una macchina di Turing deterministica a nastro singolo.

Per descrivere una macchina di Turing distruttrice con una variante delle macchine di Turing a nastro singolo, possiamo utilizzare una macchina di Turing deterministica a nastro singolo che simula il comportamento della macchina distruttrice.

Una macchina di Turing distruttrice opera nel seguente modo:

1. Ha un alfabeto di input Σ e un linguaggio L composto da termini finiti sull'alfabeto Σ .
2. La macchina di Turing distruttrice riceve in input un termine finito w su Σ .
3. La macchina di Turing distruttrice consuma il termine finito w e prende le decisioni necessarie per determinare se w appartiene al linguaggio L .
4. Durante l'esecuzione, la macchina di Turing distruttrice può modificare il contenuto delle celle del nastro, eseguendo operazioni di cancellazione o sovrascrittura.
5. Dopo aver consumato tutto l'input, la macchina di Turing distruttrice riporta sempre le sue teste alla stessa posizione iniziale, pronta per l'elaborazione successiva di un nuovo termine finito.

Per simulare una macchina di Turing distruttrice con una macchina di Turing deterministica a nastro singolo, possiamo utilizzare l'approccio di simulazione sequenziale seguente:

1. La macchina di Turing deterministica inizia la simulazione sulla stringa di input w .
2. Durante la simulazione, la macchina di Turing deterministica mantiene un registro interno per tenere traccia della posizione delle teste e delle modifiche fatte sul nastro.
3. La macchina di Turing deterministica legge un simbolo alla volta dalla stringa di input w e prende le decisioni necessarie per simulare il comportamento della macchina distruttrice.
4. Durante l'esecuzione, la macchina di Turing deterministica può eseguire le stesse operazioni di cancellazione o sovrascrittura fatte dalla macchina distruttrice.
5. Dopo aver consumato tutto l'input w , la macchina di Turing deterministica ripristina la posizione delle sue teste e il contenuto del nastro al loro stato iniziale.
6. La simulazione continua finché tutti i termini finiti sull'alfabeto Σ sono stati processati.

Poiché la macchina di Turing deterministica simula il comportamento della macchina distruttrice sequenzialmente, ripristinando sempre la posizione delle teste e il contenuto del nastro alla loro configurazione iniziale dopo aver processato ogni termine finito, è in grado di simulare il comportamento della macchina distruttrice.

Pertanto, possiamo concludere che una macchina di Turing distruttrice può essere simulata da una macchina di Turing deterministica a nastro singolo, utilizzando l'approccio di simulazione sequenziale descritto sopra.

- Diciamo che una stringa in $\{0,1\}^*$ è "dominata a sinistra" se il suo bit più basso è grande almeno quanto il suo bit più a destra, il suo secondo bit più a sinistra è grande almeno quanto il suo bit più a destra, e così via, fino a quando non vengono confrontati i bit centrali.
- Ad esempio, le seguenti stringhe sono dominate da sinistra :
 - 10
 - 100
 - 10100101
 - 111
 - 100000
 - 1
 - 0
 - stringa vuota

mentre le seguenti non lo sono:

- 01
- 001
- 10101101

Descrivete una macchina di Turing che decida il linguaggio $L = \{a \text{ in } \{0,1\}^* : a \text{ è dominato a sinistra}\}$. Dovete fornire una descrizione ad alto livello, una descrizione a livello di implementazione e una descrizione formale della vostra macchina, dimostrando come questa possa essere simulata dalla macchina TM a nastro singolo.

Descrizione ad alto livello della macchina di Turing:

La macchina di Turing per decidere il linguaggio $L = \{a \text{ in } \{0,1\}^* : a \text{ è dominato a sinistra}\}$ funzionerà confrontando i bit da sinistra a destra per verificare se ogni bit è maggiore o uguale al bit più a destra. Se in qualsiasi punto si trova un bit che è inferiore al bit più a destra, la macchina di Turing rifiuta la stringa.

Altrimenti, se viene raggiunto il bit più a destra senza trovare alcun bit inferiore, la macchina di Turing accetta la stringa.

Descrizione a livello di implementazione della macchina di Turing:

La macchina di Turing utilizzerà una testina che scorrerà da sinistra a destra sulla stringa di input. La testina manterrà il bit più a destra e confronta ogni bit successivo con esso. Se il bit successivo è inferiore al bit più a destra, la macchina di Turing passerà allo stato di rifiuto. Se il bit successivo è maggiore o uguale al bit più a destra, la macchina di Turing passerà al bit successivo fino a raggiungere il bit più a destra. Se viene raggiunto il bit più a destra senza trovare alcun bit inferiore, la macchina di Turing passerà allo stato di accettazione.

Descrizione formale della macchina di Turing:

La macchina di Turing può essere formalmente descritta come una 7-tupla $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$, dove:

- Q è l'insieme degli stati della macchina di Turing.
- Σ è l'alfabeto di input che contiene i simboli 0 e 1.
- Γ è l'alfabeto del nastro che include Σ , oltre a un simbolo speciale per la casella vuota e altri simboli ausiliari se necessario.
- δ è la funzione di transizione che specifica come la macchina di Turing si muove tra gli stati in base ai simboli letti dal nastro.
- q_0 è lo stato iniziale.
- q_{acc} è lo stato di accettazione.
- q_{rej} è lo stato di rifiuto.

La funzione di transizione δ può essere definita come segue:

$$\delta(q, a) = (p, b, D)$$

dove q è lo stato corrente, a è il simbolo letto sulla testina, p è lo stato successivo, b è il simbolo da scrivere sulla testina e D indica la direzione in cui la testina si muove (destra D o sinistra S).

La simulazione della macchina di Turing a nastro singolo può avvenire utilizzando una codifica appropriata dell'input e delle configurazioni della macchina di Turing distrutta. È possibile utilizzare delimitatori speciali o simboli ausiliari per rappresentare le posizioni della testina sul nastro. La macchina di Turing a nastro singolo può iterare sequenzialmente attraverso le transizioni della macchina di Turing distrutta, modificando il contenuto del nastro secondo necessità.

La simulazione può avvenire seguendo un approccio di simulazione delle scelte non deterministiche, considerando tutte le possibili transizioni della macchina di Turing distrutta e seguendo il percorso corrispondente all'input. Se in qualsiasi punto viene rilevato uno stato di rifiuto, la macchina di Turing a nastro singolo passerà allo stato di rifiuto. Se viene raggiunto lo stato di accettazione, la macchina di Turing a nastro singolo passerà allo stato di accettazione.

Pertanto, la macchina di Turing per decidere il linguaggio $L = \{a \in \{0,1\}^* : a \text{ è dominato a sinistra}\}$ può essere simulata da una macchina di Turing a nastro singolo utilizzando un'adeguata codifica dell'input e delle configurazioni della macchina di Turing distrutta, seguendo l'approccio di simulazione descritto sopra.

- Progettare una macchina di Turing che sommi due numeri binari dati. Prendiamo come input l'alfabeto $\Sigma = \{0, 1\}$ e assumiamo che i numeri in ingresso siano dati come due stringhe divise da una cella vuota. Ad esempio, $|_|10|_|101|_|$ sarebbe un input per $2 + 5$.

Per progettare una macchina di Turing che sommi due numeri binari dati, possiamo seguire l'approccio seguente utilizzando una macchina di Turing a nastro singolo:

1. La macchina di Turing legge il primo numero binario fino alla cella vuota. Durante la lettura, la macchina di Turing può memorizzare il numero in una delle sue variabili interne.
2. Dopo aver letto il primo numero, la macchina di Turing si sposta alla cella successiva e legge il secondo numero binario fino alla cella vuota. Durante la lettura, la macchina di Turing può memorizzare il secondo numero in un'altra variabile interna.
3. La macchina di Turing esegue la somma dei due numeri binari. Può utilizzare variabili interne per tenere traccia del riporto durante la somma.
4. La macchina di Turing scrive il risultato della somma sul nastro, sovrascrivendo il secondo numero binario.
5. La macchina di Turing torna alla cella iniziale e ripristina la posizione della testina.
6. La macchina di Turing può continuare a leggere ulteriori numeri binari e sommarli al risultato precedente, se necessario.

La simulazione della macchina di Turing a nastro singolo può avvenire utilizzando una codifica appropriata dei numeri binari e delle configurazioni della macchina di Turing. Possiamo utilizzare delimitatori speciali o simboli ausiliari per separare i numeri binari e tenere traccia della posizione della testina sul nastro.

La macchina di Turing a nastro singolo può iterare sequenzialmente attraverso le transizioni della macchina di Turing per leggere i numeri binari, eseguire la somma e scrivere il risultato sul nastro. La macchina di Turing può utilizzare variabili interne per tenere traccia dei numeri binari e dei riporti durante la somma.

Pertanto, la macchina di Turing per sommare due numeri binari dati può essere simulata da una macchina di Turing a nastro singolo utilizzando un'adeguata codifica degli input e delle configurazioni della macchina di Turing, seguendo l'approccio di simulazione descritto sopra.

Soluzione Esercizi 23/05

Automi e Linguaggi Formali – Seconda Parte

Turing Machine

(8 punti) Una Turing machine con alfabeto binario è una macchina di Turing deterministica a singolo nastro dove l’alfabeto di input è $\Sigma = \{0, 1\}$ e l’alfabeto del nastro è $\Gamma = \{0, 1, _\}$. Questo significa che la macchina può scrivere sul nastro solo i simboli 0, 1 e blank: non può usare altri simboli né marcare i simboli sul nastro.

Dimostra che le Turing machine con alfabeto binario riconoscono tutti e soli i linguaggi Turing-riconoscibili sull’alfabeto $\{0, 1\}$.

Per risolvere l’esercizio dobbiamo dimostrare che (a) ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è Turing-riconoscibile e (b) ogni linguaggio Turing-riconoscibile sull’alfabeto $\{0, 1\}$ è riconosciuto da una Turing machine con alfabeto binario.

- Questo caso è semplice: una Turing machine con alfabeto binario è un caso speciale di Turing machine deterministica a nastro singolo. Quindi ogni linguaggio riconosciuto da una Turing machine con alfabeto binario è anche Turing-riconoscibile.
- Per dimostrare questo caso, consideriamo un linguaggio L Turing-riconoscibile, e sia M una Turing machine deterministica a nastro singolo che lo riconosce. Questa TM potrebbe avere un alfabeto del nastro Γ che contiene altri simboli oltre a 0, 1 e blank. Per esempio potrebbe contenere simboli marcati o separatori.

Per costruire una TM con alfabeto binario B che simula il comportamento di M dobbiamo come prima cosa stabilire una *codifica binaria* dei simboli nell’alfabeto del nastro Γ di M . Questa codifica è una funzione C che assegna ad ogni simbolo $a \in \Gamma$ una sequenza di k cifre binarie, dove k è un valore scelto in modo tale che ad ogni simbolo corrisponda una codifica diversa. Per esempio, se Γ contiene 4 simboli, allora $k = 2$, perché con 2 bit si rappresentano 4 valori diversi. Se Γ contiene 8 simboli, allora $k = 3$, e così via.

La TM con alfabeto binario B che simula M è definita in questo modo:

B = ”su input w :

- Sostituisce $w = w_1w_2\dots w_n$ con la codifica binaria $C(w_1)C(w_2)\dots C(w_n)$, e riporta la testina sul primo simbolo di $C(w_1)$.
- Scorre il nastro verso destra per leggere k cifre binarie: in questo modo la macchina stabilisce qual è il simbolo a presente sul nastro di M . Va a sinistra di k celle.
- Aggiorna il nastro in accordo con la funzione di transizione di M :
 - Se $\delta(r, a) = (s, b, R)$, scrive la codifica binaria di b sul nastro.
 - Se $\delta(r, a) = (s, b, L)$, scrive la codifica binaria di b sul nastro e sposta la testina a sinistra di $2k$ celle.
- Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M , allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2.”

- 1. (12 punti)** Una *Macchina di Turing con eliminazione* è una macchina di Turing deterministica a nastro singolo che può eliminare celle dal nastro. Formalmente la funzione di transizione è definita come

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, D\}$$

dove L, R indicano i normali spostamenti a sinistra e a destra della testina, e D indica l'eliminazione della cella sotto la posizione corrente della testina. Dopo una operazione di eliminazione, la testina si muove nella cella che si trovava immediatamente a destra della cella eliminata.

Dimostra che qualsiasi macchina di Turing con eliminazione può essere simulata da una macchina di Turing deterministica a nastro singolo.

Soluzione. Mostriamo come convertire una macchina di Turing con Inserimento M in una TM deterministica a nastro singolo S equivalente. La simulazione usa il simbolo speciale $\#$ per segnare le celle che vengono eliminate.

S = “Su input w :

1. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, L)$ procede come nella TM standard: S scrive b sul nastro e muove la testina di una cella a sinistra. Se lo spostamento a sinistra porta la testina sopra un $\#$, allora continua a spostare la testina a sinistra finché non si arriva in una cella che contiene un simbolo diverso dal $\#$.
2. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, R)$ procede come nella TM standard: S scrive b sul nastro e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra un $\#$, allora continua a spostare la testina a destra finché non si arriva in una cella che contiene un simbolo diverso dal $\#$.
3. Per simulare una mossa del tipo $\delta(q, a) = (r, b, D)$ la TM S scrive $\#$ nella cella corrente e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra un $\#$, allora continua a spostare la testina a destra finché non si arriva in una cella che contiene un simbolo diverso dal $\#$.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M , allora S termina con accettazione. Se in qualsiasi momento la simulazione raggiunge lo stato di rifiuto di M , allora S termina con rifiuto. Negli altri casi continua la simulazione dal punto 2.”

- 3. (9 punti)** Una *macchina di Turing ad albero binario* usa un albero binario infinito come nastro, dove ogni cella nel nastro ha un figlio sinistro e un figlio destro. Ad ogni transizione, la testina si sposta dalla cella corrente al padre, al figlio sinistro oppure al figlio destro della cella corrente. Pertanto, la funzione di transizione di una tale macchina ha la forma

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{P, L, R\},$$

dove P indica lo spostamento verso il padre, L verso il figlio sinistro e R verso il figlio destro. La stringa di input viene fornita lungo il ramo sinistro dell'albero.

Mostra che qualsiasi macchina di Turing ad albero binario può essere simulata da una macchina di Turing standard.

Per dimostrare che qualsiasi macchina di Turing ad albero binario può essere simulata da una macchina di Turing standard, dobbiamo costruire una simulazione che preserva il comportamento della macchina di Turing ad albero binario utilizzando una macchina di Turing standard.

Consideriamo una macchina di Turing ad albero binario M_1 con un albero binario infinito come nastro e la funzione di transizione $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{P, L, R\}$, come descritto nell'enunciato.

Per simulare M_1 utilizzando una macchina di Turing standard M_2 , possiamo utilizzare una codifica dell'albero binario infinito come stringa di input sulla quale M_2 opera. Una possibile codifica potrebbe essere rappresentare l'albero binario utilizzando un'appropriata notazione come parentesi annidate.

La macchina di Turing M_2 opera come segue:

1. Legge la stringa di input, che rappresenta l'albero binario, dalla posizione iniziale della sua nastro.

2. Simula il comportamento della macchina di Turing ad albero binario M1 utilizzando una serie di istruzioni della macchina di Turing standard.

3. Per ogni transizione $\delta(q, a) = (p, b, P)$ di M1, M2 simula il movimento verso il padre dell'albero binario rappresentato dalla stringa di input. Per fare ciò, M2 si sposta verso sinistra sulla stringa di input fino a raggiungere il punto in cui il nodo corrente è il padre del nodo corrente nell'albero binario.

4. Per ogni transizione $\delta(q, a) = (p, b, L)$ di M1, M2 simula il movimento verso il figlio sinistro dell'albero binario rappresentato dalla stringa di input. Per fare ciò, M2 si sposta verso destra sulla stringa di input fino a raggiungere il punto in cui il nodo corrente è il figlio sinistro del nodo corrente nell'albero binario.

5. Per ogni transizione $\delta(q, a) = (p, b, R)$ di M1, M2 simula il movimento verso il figlio destro dell'albero binario rappresentato dalla stringa di input. Per fare ciò, M2 si sposta verso destra sulla stringa di input fino a raggiungere il punto in cui il nodo corrente è il figlio destro del nodo corrente nell'albero binario.

6. M2 applica le regole di transizione di M1, sostituendo i simboli corrispondenti nella stringa di input.

7. Continua la simulazione di M1 fino a quando M2 raggiunge uno stato finale di accettazione o di rifiuto.

In questo modo, la macchina di Turing M2 simula il comportamento della macchina di Turing ad albero binario M1 utilizzando una codifica dell'albero binario come stringa di input sulla quale opera. Poiché M2 è una macchina di Turing standard, abbiamo dimostrato che qualsiasi macchina di Turing ad albero binario può essere simulata da una macchina di Turing standard.

1. **(12 punti)** Una *Macchina di Turing con stack di nastri* possiede due azioni aggiuntive che modificano il suo nastro di lavoro, oltre alle normali operazioni di scrittura di singole celle e spostamento a destra o a sinistra della testina: può salvare l'intero nastro inserendolo in uno stack (operazione di Push) e può ripristinare l'intero nastro estraendolo dallo stack (operazione di Pop). Il ripristino del nastro riporta il contenuto di ogni cella al suo contenuto quando il nastro è stato salvato. Il salvataggio e il ripristino del nastro non modificano lo stato della macchina o la posizione della sua testina. Se la macchina tenta di "ripristinare" il nastro quando lo stack è vuoto, la macchina va nello stato di rifiuto. Se ci sono più copie del nastro salvate nello stack, la macchina ripristina l'ultima copia inserita nello stack, che viene quindi rimossa dallo stack.

Mostra che qualsiasi macchina di Turing con stack di nastri può essere simulata da una macchina di Turing standard. *Suggerimento:* usa una macchina multinastro per la simulazione.

Per mostrare che qualsiasi macchina di Turing con stack di nastri può essere simulata da una macchina di Turing standard, useremo una macchina di Turing multinastro per la simulazione.

Supponiamo di avere una macchina di Turing con stack di nastri, dove ogni nastro rappresenta uno stack. Chiameremo questa macchina di Turing con stack di nastri come MTStack.

Per simulare MTStack utilizzando una macchina di Turing standard, creeremo una macchina di Turing multinastro chiamata MTMulti. MTMulti avrà un nastro per la sua memoria di lavoro e un numero arbitrario di nastri aggiuntivi per simulare gli stack.

La simulazione avviene nel modo seguente:

1. Inizialmente, la macchina di Turing MTMulti copia il contenuto del primo nastro di MTStack nel suo nastro di memoria di lavoro. Questo può essere fatto spostando la testina sui nastri di MTStack e copiando ogni simbolo nel nastro di memoria di lavoro di MTMulti.
2. Per simulare l'operazione di Push di MTStack, MTMulti aggiunge un nuovo nastro al suo insieme di nastri aggiuntivi e copia il contenuto del nastro di memoria di lavoro nel nuovo nastro aggiuntivo. In altre parole, MTMulti salva lo stato corrente del nastro di memoria di lavoro nel nuovo nastro.

3. Per simulare l'operazione di Pop di MTStack, MTMulti estraе il nastro più recentemente aggiunto dal suo insieme di nastri aggiuntivi e copia il suo contenuto nel nastro di memoria di lavoro. In altre parole, MTMulti ripristina lo stato precedente del nastro di memoria di lavoro dal nastro aggiuntivo.

4. MTMulti esegue tutte le altre operazioni di MTStack (scrittura di singole celle, spostamento a destra o sinistra della testina) direttamente sul suo nastro di memoria di lavoro.

5. Se MTStack tenta di "ripristinare" il nastro quando lo stack è vuoto, MTMulti va nello stato di rifiuto.

In questo modo, MTMulti simula l'intero comportamento di MTStack utilizzando i suoi nastri aggiuntivi per simulare gli stack. Ogni stack è rappresentato da un nastro aggiuntivo, e l'operazione di Push corrisponde alla copia del contenuto del nastro di memoria di lavoro nel nuovo nastro, mentre l'operazione di Pop corrisponde all'estrazione del nastro più recentemente aggiunto e al ripristino del suo contenuto nel nastro di memoria di lavoro.

Poiché MTMulti è una macchina di Turing standard che simula il comportamento di MTStack, dimostriamo che qualsiasi macchina di Turing con stack di nastri può essere simulata da una macchina di Turing standard.

1. Una *tag-Turing machine* è una macchina di Turing con un singolo nastro e due testine: una testina può solo leggere, l'altra può solo scrivere. All'inizio della computazione la testina di lettura si trova sopra il primo simbolo dell'input e la testina di scrittura si trova sopra la cella vuota posta immediatamente dopo la stringa di input. Ad ogni transizione, la testina di lettura può spostarsi di una cella a destra o rimanere ferma, mentre la testina di scrittura deve scrivere un simbolo nella cella corrente e spostarsi di una cella a destra. Nessuna delle due testine può spostarsi a sinistra.

Dimostra che le tag-Turing machine riconoscono la classe dei linguaggi Turing-riconoscibili.

1. Per risolvere l'esercizio dobbiamo dimostrare che (a) ogni linguaggio riconosciuto da una tag-Turing machine è Turing-riconoscibile e (b) ogni linguaggio Turing-riconoscibile è riconosciuto da una tag-Turing machine.

- (a) Mostriamo come convertire una tag-Turing machine M in una TM deterministica a nastro singolo S equivalente. S simula il comportamento di M tenendo traccia delle posizioni delle due testine marcando la cella dove si trova la testina di lettura con un pallino sopra il simbolo, e marcando la cella dove si trova la testina di scrittura con un pallino sotto il simbolo. Per simulare il comportamento di M la TM S scorre il nastro e aggiorna le posizioni delle testine ed il contenuto delle celle come indicato dalla funzione di transizione di M .

$S = \text{"Su input } w = w_1 w_2 \dots w_n:$

1. Scrivi un pallino sopra il primo simbolo di w e un pallino sotto la prima cella vuota dopo l'input, in modo che il nastro contenga

$$\overset{\bullet}{w_1} w_2 \dots w_n \underset{\bullet}{\omega}$$

2. Per simulare una transizione, S scorre il nastro per trovare la posizione della testina di lettura e determinare il simbolo letto da M . Se la funzione di transizione stabilisce che la testina di lettura deve spostarsi a destra, allora S sposta il pallino nella cella immediatamente a destra, altrimenti lo lascia dov'è. Successivamente S si sposta verso destra finché non trova la cella dove si trova la testina di scrittura, scrive il simbolo stabilito dalla funzione di transizione nella cella e sposta la marcatura nella cella immediatamente a destra.
3. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M , allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2.”

(b) Mostriamo come convertire una TM deterministica a nastro singolo S in una tag-Turing machine M equivalente. M simula il comportamento di S memorizzando sul nastro una sequenza di configurazioni di S separate dal simbolo $\#$. All'interno di ogni configurazione un pallino marca il simbolo sotto la testina di S . Per simulare il comportamento di S la tag-Turing machine M scorre la configurazione corrente e scrivendo man mano la prossima configurazione sul nastro.

M = "Su input $w = w_1 w_2 \dots w_n$:

1. Scrive il simbolo $\#$ subito dopo l'input, seguito dalla configurazione iniziale, in modo che il nastro contenga

$$w_1 w_2 \dots w_n \# w_1^* w_2 \dots w_n \omega,$$

che la testina di lettura si trovi in corrispondenza del w_1^* e quella di lettura in corrispondenza del blank dopo la configurazione iniziale. Imposta lo stato corrente della simulazione st allo stato iniziale di S e memorizza l'ultimo simbolo letto $prec = \#$. L'informazione sui valori di st e $prec$ sono codificate all'interno degli stati di M .

2. Finché il simbolo sotto la testina di lettura non è marcato, scrive il simbolo precedente $prec$ e muove a destra. Aggiorna il valore di $prec$ con il simbolo letto.
3. Quando si trova un simbolo marcato a e $\delta(st, a) = (q, b, R)$:
 - aggiorna lo stato della simulazione $st = q$;
 - scrive $prec$ seguito da b , poi muove la testina di lettura a destra;
 - scrive il simbolo sotto la testina marcandolo con un pallino.
4. Quando si trova un simbolo marcato a e $\delta(st, a) = (q, b, L)$:
 - aggiorna lo stato della simulazione $st = q$;
 - scrive $prec$; se $prec = \#$ scrive $\#\Sigma$;
 - scrive b .
5. Copia il resto della configurazione fino al $\#$ escluso. Al termine della copia la testina di lettura di trova in corrispondenza della prima cella nella configurazione corrente, e quella di lettura sulla cella vuota dopo la configurazione.
6. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di S , allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di M allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2."

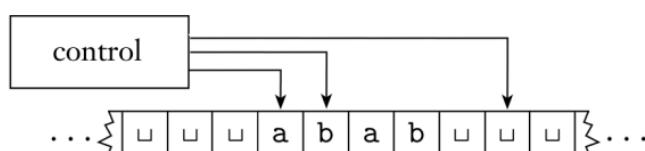
1. Una macchina di Turing a testine multiple è una macchina di Turing con un solo nastro ma con varie testine. Inizialmente, tutte le testine si trovano sopra alla prima cella dell'input. La funzione di transizione viene modificata per consentire la lettura, la scrittura e lo spostamento delle testine. Formalmente,

$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R\}^k$$

dove k è il numero delle testine. L'espressione

$$\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$$

significa che, se la macchina si trova nello stato q_i e le testine da 1 a k leggono i simboli a_1, \dots, a_k allora la macchina va nello stato q_j , scrive i simboli b_1, \dots, b_k e muove le testine a destra e a sinistra come specificato.



Un esempio di TM con tre testine.

Per dimostrare che qualsiasi macchina di Turing a testine multiple può essere simulata da una macchina di Turing deterministica a nastro singolo, utilizzeremo un'argomentazione di simulazione.

Sia M una macchina di Turing a testine multiple con k testine. Dobbiamo costruire una macchina di Turing deterministica a nastro singolo che simuli il comportamento di M .

La simulazione avviene in modo sequenziale, in cui la macchina di Turing deterministica esegue le operazioni delle testine di M una dopo l'altra.

La macchina di Turing deterministica a nastro singolo T funziona come segue:

1. T tiene traccia dello stato corrente di M e delle posizioni delle testine.
2. In ogni passo, T seleziona una delle testine di M e legge il simbolo corrispondente sulla posizione corrente della testina selezionata.
3. T determina l'azione successiva di M in base allo stato corrente di M e ai simboli letti dalle testine.
4. T aggiorna lo stato corrente di M, scrive il simbolo appropriato sulla posizione corrente e si sposta verso sinistra o verso destra sulla base dell'azione determinata.
5. T passa quindi alla testina successiva e ripete i passaggi 2-4 fino a quando tutte le testine sono state considerate.
6. T continua ad eseguire i passaggi 2-5 finché M non raggiunge uno stato di arresto.

La chiave per la simulazione è che la macchina di Turing deterministica T, anche se lavora su un nastro singolo, tiene traccia delle posizioni delle diverse testine di M e riproduce le loro operazioni una dopo l'altra.

Poiché la macchina di Turing deterministica T esegue le stesse operazioni di M in sequenza, T simula efficacemente il comportamento di M. Entrambe le macchine accettano o rifiutano lo stesso linguaggio e producono lo stesso risultato per ogni input.

Pertanto, qualsiasi macchina di Turing a testine multiple può essere simulata da una macchina di Turing deterministica a nastro singolo.

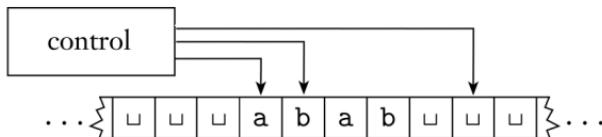
1. Una macchina di Turing a testine multiple è una macchina di Turing con un solo nastro ma con varie testine. Inizialmente, tutte le testine si trovano sopra alla prima cella dell'input. La funzione di transizione viene modificata per consentire la lettura, la scrittura e lo spostamento delle testine. Formalmente,

$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R\}^k$$

dove k è il numero delle testine. L'espressione

$$\delta(q_i, a_1, \dots, a_k) = (q_j, b_1, \dots, b_k, L, R, \dots, L)$$

significa che, se la macchina si trova nello stato q_i e le testine da 1 a k leggono i simboli a_1, \dots, a_k allora la macchina va nello stato q_j , scrive i simboli b_1, \dots, b_k e muove le testine a destra e a sinistra come specificato.



Un esempio di TM con tre testine.

Una *macchina di Turing con reset a sinistra* è una variante delle comuni macchine di Turing, dove la funzione di transizione ha la forma:

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{R, \text{RESET}\}.$$

Se $\delta(q, a) = (r, b, \text{RESET})$, quando la macchina si trova nello stato q e legge a , la testina scrive b sul nastro, salta all'estremità sinistra del nastro ed entra nello stato r . Per sapere su quale cella saltare la macchina usa il simbolo speciale \triangleright per identificare l'estremità di sinistra del nastro. Questo simbolo si può trovare solo in una cella del nastro, e non può essere sovrascritto o cancellato. La computazione di una macchina di Turing con reset a sinistra sulla parola w inizia con $\triangleright w$ sul nastro. Si noti che queste macchine non hanno la solita capacità di muovere la testina di una cella a sinistra.

Mostrare che le macchine di Turing con reset a sinistra riconoscono la classe dei linguaggi Turing-riconoscibili.

- (a) Mostriamo come convertire una TM con reset a sinistra M in una TM standard S equivalente. S simula il comportamento di M nel modo seguente. Se la mossa da simulare prevede uno spostamento a destra, allora S esegue direttamente la mossa. Se la mossa prevede un *RESET*, allora S scrive il nuovo simbolo sul nastro, poi scorre il nastro a sinistra finché non trova il simbolo \triangleright , e riprende la simulazione dall'inizio del nastro. Per ogni stato q di M , S possiede uno stato q_{RESET} che serve per simulare il reset e riprendere la simulazione dallo stato corretto.

S = “Su input w :

1. scrive il simbolo \triangleright subito prima dell'input, in modo che il nastro contenga $\triangleright w$.
2. Se la mossa da simulare è $\delta(q, a) = (r, b, R)$, allora S la esegue direttamente: scrive b sul nastro, muove la testina a destra e va nello stato r .
3. Se la mossa da simulare è $\delta(q, a) = (r, b, \text{RESET})$, allora S esegue le seguenti operazioni: scrive b sul nastro, poi muove la testina a sinistra e va nello stato r_{RESET} . La macchina rimane nello stato r_{RESET} e continua a muovere la testina a sinistra finché non trova il simbolo \triangleright . A quel punto la macchina sposta la testina un'ultima volta a sinistra, poi di una cella a destra per tornare sopra al simbolo di fine nastro. La computazione riprende dallo stato r .
4. Se non sei nello stato di accettazione o di rifiuto, ripeti da 2.”

- (b) Mostriamo come convertire una TM standard S in una TM con reset a sinistra M equivalente. M simula il comportamento di S nel modo seguente. Se la mossa da simulare prevede uno spostamento a destra, allora M può eseguire direttamente la mossa. Se la mossa da simulare prevede uno spostamento a sinistra, allora M simula la mossa come descritto dall'algoritmo seguente. L'algoritmo usa un nuovo simbolo \triangleleft per identificare la fine della porzione di nastro usata fino a quel momento, e può marcare le celle del nastro ponendo un punto al di sopra di un simbolo.

M = "Su input w :

1. Scrive il simbolo \triangleleft subito dopo l'input, per marcare la fine della porzione di nastro utilizzata. Il nastro contiene $\triangleright w \triangleleft$.
 2. Simula il comportamento di S . Se la mossa da simulare è $\delta(q, a) = (r, b, R)$, allora M la esegue direttamente: scrive b sul nastro, muove la testina a destra e va nello stato r . Se muovendosi a destra la macchina si sposta sulla cella che contiene \triangleleft , allora questo significa che S ha spostato la testina sulla parte di nastro vuota non usata in precedenza. Quindi M scrive un simbolo blank marcato su questa cella, sposta \triangleleft di una cella a destra, e fa un reset a sinistra. Dopo il reset si muove a destra fino al blank marcato, e prosegue con la simulazione mossa successiva.
 3. Se la mossa da simulare è $\delta(q, a) = (r, b, L)$, allora S esegue le seguenti operazioni:
 - 3.1 scrive b sul nastro, marcandolo con un punto, poi fa un reset a sinistra
 - 3.2 Se il simbolo subito dopo \triangleright è già marcato, allora vuol dire che S ha spostato la testina sulla parte vuota di sinistra del nastro. Quindi M scrive un blank e sposta il contenuto del nastro di una cella a destra finché non trova il simbolo di fine nastro \triangleleft . Fa un reset a sinistra e prosegue con la simulazione della prossima mossa dal nuovo blank posto subito dopo l'inizio del nastro. Se il simbolo subito dopo \triangleright non è marcato, lo marca, resetta a sinistra e prosegue con i passi successivi.
 - 3.3 Si muove a destra fino al primo simbolo marcato, e poi a destra di nuovo.
 - 3.4 se la cella in cui si trova è marcata, allora è la cella da cui è partita la simulazione. Toglie la marcatura e resetta. Si muove a destra finché non trova una cella marcata. Questa cella è quella immediatamente precedente la cella di partenza, e la simulazione della mossa è terminata
 - 3.5 se la cella in cui si trova non è marcata, la marca, resetta, si muove a destra finché non trova una marcatura, cancella la marcatura e riprende da 3.3.
4. Se non sei nello stato di accettazione o di rifiuto, ripeti da 2."

Indecidibili

2. Dimostra che il seguente linguaggio è indecidibile:

$$A_{1010} = \{\langle M \rangle \mid M \text{ è una TM tale che } 1010 \in L(M)\}.$$

Dimostriamo che A_{1010} è un linguaggio indecidibile mostrando che A_{TM} è riducibile ad A_{1010} . La funzione di riduzione f è calcolata dalla seguente macchina di Turing:

F = "su input $\langle M, w \rangle$, dove M è una TM e w una stringa:

1. Costruisci la seguente macchina M_w :

M_w = "su input x :

1. Se $x \neq 1010$, rifiuta.
2. Se $x = 1010$, esegue M su input w .
3. Se M accetta, *accetta*.
4. Se M rifiuta, *rifiuta*."

2. Restituisci $\langle M_w \rangle$."

Dimostriamo che f è una funzione di riduzione da A_{TM} ad A_{1010} .

- Se $\langle M, w \rangle \in A_{TM}$ allora la TM M accetta w . Di conseguenza la macchina M_w costruita dalla funzione accetta la parola 1010. Quindi $f(\langle M, w \rangle) = \langle M_w \rangle \in A_{1010}$.
- Viceversa, se $\langle M, w \rangle \notin A_{TM}$ allora la computazione di M su w non termina o termina con rifiuto. Di conseguenza la macchina M_w rifiuta 1010 e $f(\langle M, w \rangle) = \langle M_w \rangle \notin A_{1010}$.

Per concludere, siccome abbiamo dimostrato che $A_{TM} \leq_m A_{1010}$ e sappiamo che A_{TM} è indecidibile, allora possiamo concludere che A_{1010} è indecidibile.

2. (12 punti) Data una Turing Machine M , definiamo

$$\text{HALTS}(M) = \{w \mid M \text{ termina la computazione su } w\}.$$

Considera il linguaggio

$$I = \{\langle M \rangle \mid \text{HALTS}(M) \text{ è un insieme infinito}\}.$$

Dimostra che I è indecidibile.

Soluzione. La seguente macchina F calcola una riduzione mediante funzione $A_{TM} \leq_m I$:

F = "su input $\langle M, w \rangle$, dove M è una TM e w una stringa:

1. Costruisci la seguente macchina M' :

M' = "Su input x :

1. Esegue M su input w .
2. Se M accetta, *accetta*.
3. Se M rifiuta, va in loop."

2. Ritorna $\langle M' \rangle$.

Mostriamo che F calcola una funzione di riduzione f da A_{TM} a I , cioè una funzione tale che

$$\langle M, w \rangle \in A_{TM} \text{ se e solo se } M' \in I.$$

- Se $\langle M, w \rangle \in A_{TM}$ allora la macchina M accetta w . In questo caso la macchina M' accetta tutte le parole, quindi $\text{HALTS}(M') = \Sigma^*$ che è un insieme infinito. Di conseguenza $M' \in I$.
- Viceversa, se $\langle M, w \rangle \notin A_{TM}$, allora la macchina M su input w rifiuta oppure va in loop. In entrambi i casi la macchina M' va in loop su tutte le stringhe, quindi $\text{HALTS}(M') = \emptyset$ che è un insieme finito. Di conseguenza $M' \notin I$.

2. (12 punti) Considera il linguaggio

$\text{FORTY-TWO} = \{\langle M, w \rangle \mid M \text{ termina la computazione su } w \text{ avendo solo 42 sul nastro}\}$.

Dimostra che FORTY-TWO è indecidibile.

Per dimostrare che il linguaggio Forty-Two è indecidibile, utilizzeremo l'argomento della riduzione al problema dell'arresto (Halting Problem).

Supponiamo per assurdo che Forty-Two sia decidibile, il che significa che esiste una macchina di Turing (MT) che può determinare se un'altra macchina di Turing M termina la computazione su una stringa w, avendo solo 42 sul nastro.

Possiamo utilizzare questa MT per risolvere il problema dell'arresto, che è noto essere indecidibile. Dato un input $\langle M', w \rangle$ per il problema dell'arresto, possiamo costruire un input $\langle M, w \rangle$ per il linguaggio FORTY-Two come segue:

1. La macchina di Turing M che rappresenta l'input $\langle M', w \rangle$ per il problema dell'arresto può essere modificata per accettare solo 42 sul nastro. Questa modifica può essere fatta semplicemente sostituendo tutti i simboli diversi da 42 con un'istruzione che va in uno stato di rifiuto.
2. La stringa w può essere scelta in modo arbitrario.

Ora, se utilizziamo la MT che decide Forty-Two sull'input $\langle M, w \rangle$, essa determinerà se M termina la computazione su w, avendo solo 42 sul nastro. Ma questo è equivalente a risolvere il problema dell'arresto sull'input $\langle M', w \rangle$, il che è impossibile poiché il problema dell'arresto è indecidibile.

Di conseguenza, abbiamo ottenuto una contraddizione, il che dimostra che Forty-Two è indecidibile, poiché supporre il contrario avrebbe implicazioni che contraddicono la indecidibilità del problema dell'arresto.

2. Considera il linguaggio $\text{ALWAYSDIVERGE} = \{\langle M \rangle \mid M \text{ è una TM tale che per ogni } w \in \Sigma^* \text{ la computazione di } M \text{ su input } w \text{ non termina}\}$.

- (a) Dimostra che il ALWAYSDIVERGE è indecidibile.
- (b) ALWAYSDIVERGE è un linguaggio Turing-riconoscibile, coTuring-riconoscibile oppure né Turing-riconoscibile né coTuring-riconoscibile? Giustifica la tua risposta.

2. (a) Dimostriamo che ALWAYSDIVERGE è un linguaggio indecidibile mostrando che $\overline{A_{TM}}$ è riducibile ad ALWAYSDIVERGE . La funzione di riduzione f è calcolata dalla seguente macchina di Turing:

$F = \text{"su input } \langle M, w \rangle, \text{ dove } M \text{ è una TM e } w \text{ una stringa:}$

1. Costruisci la seguente macchina M' :

$M' = \text{"su input } x:$

1. Se $x \neq w$, vai in loop.
2. Se $x = w$, esegue M su input w .
3. Se M accetta, *accetta*.
4. Se M rifiuta, vai in loop."

2. Restituisci $\langle M' \rangle$.

Dimostriamo che f è una funzione di riduzione da $\overline{A_{TM}}$ ad ALWAYSDIVERGE .

- Se $\langle M, w \rangle \in \overline{A_{TM}}$ allora la computazione di M su w non termina oppure termina rifiutando. Di conseguenza la macchina M' costruita dalla funzione non termina mai la computazione per qualsiasi input. Quindi $f(\langle M, w \rangle) = \langle M' \rangle \in \text{ALWAYSDIVERGE}$.
- Viceversa, se $\langle M, w \rangle \notin \overline{A_{TM}}$ allora la computazione di M su w termina con accettazione. Di conseguenza la macchina M' termina la computazione sull'input w e $f(\langle M, w \rangle) = \langle M' \rangle \notin \text{ALWAYSDIVERGE}$.

Per concludere, siccome abbiamo dimostrato che $\overline{A_{TM}} \leq_m \text{ALWAYSDIVERGE}$ e sappiamo che $\overline{A_{TM}}$ è indecidibile, allora possiamo concludere che ALWAYSDIVERGE è indecidibile.

(b) ALWAYSDIVERGE è un linguaggio coTuring-riconoscibile. Dato un ordinamento s_1, s_2, \dots di tutte le stringhe in Σ^* , la seguente TM riconosce il complementare $\overline{\text{ALWAYSDIVERGE}}$:

$D = \text{"su input } \langle M \rangle, \text{ dove } M \text{ è una TM:}$

1. Ripeti per $i = 1, 2, 3, \dots$:
2. Esegue M per i passi di computazione su ogni input $s_1, s_2, \dots s_i$.
3. Se M termina la computazione su almeno uno, *accetta*. Altrimenti continua."

2. Dimostra che il seguente linguaggio è indecidibile:

$$L_2 = \{\langle M, w \rangle \mid M \text{ accetta la stringa } ww^R\}.$$

Per dimostrare che il linguaggio $L_2 = \{(M, w) \mid M \text{ accetta la stringa } ww^R\}$ è indecidibile, utilizzeremo un'argomentazione di riduzione da un altro problema indecidibile noto, come il problema dell'arresto (Halting Problem).

Supponiamo per assurdo che esista una macchina di Turing (TM) chiamata H che decide il linguaggio L_2 . La macchina di Turing H accetta una coppia (M, w) come input e restituisce "accetta" se M accetta la stringa ww^R e "rifiuta" altrimenti.

Costruiremo una nuova macchina di Turing TM' che prende in input un codice di TM M' e produce una coppia (M, w) come input per H , in modo che TM' accetta se e solo se M' non si arresta su w .

La costruzione di TM' è la seguente:

1. TM' prende in input un codice di TM M' e una stringa w .

2. TM' costruisce una nuova TM M come segue:

- M si comporta come M' su w .

- Inserisce w^R nel nastro dopo la stringa di input.

- M inizia a leggere la stringa di input e la copia sulla parte destra del nastro.
 - Una volta raggiunto il simbolo di fine input sulla parte sinistra del nastro, M inizia a leggere e confrontare i simboli sulla parte destra del nastro con quelli sulla parte sinistra.
 - Se i simboli corrispondono, M continua la computazione come M' su w . Se non corrispondono, M entra in uno stato di rifiuto.
3. TM' passa la coppia (M, w) a H come input.
4. Se H accetta la coppia (M, w) , cioè se M accetta la stringa ww^R , allora TM' entra in uno stato di rifiuto.
5. Se H rifiuta la coppia (M, w) , cioè se M rifiuta la stringa ww^R o entra in un loop infinito, allora TM' accetta.

Ora, supponiamo che TM' decidesse il problema dell'arresto. In altre parole, supponiamo che TM' possa determinare se una macchina di Turing M' si arresta su una determinata stringa w .

Consideriamo il caso in cui M' non si arresta su w . In questo caso, TM' accetta la coppia (M, w) e H rifiuta, poiché M non accetta la stringa ww^R . Allo stesso modo, se M' si arresta su w , TM' entra in uno stato di rifiuto e H accetta.

Tuttavia, questo porta a una contraddizione. Se TM' decidesse il problema dell'arresto, potremmo utilizzare TM' per risolvere il problema dell'arresto, che è noto essere indecidibile. Pertanto, la nostra suposizione iniziale che TM' decidesse il problema dell'arresto è falsa.

Di conseguenza, il linguaggio $L_2 = \{(M, w) \mid M \text{ accetta la stringa } ww^R\}$ è indecidibile.

3. (8 punti) Una *Turing machine con alfabeto ternario* è una macchina di Turing deterministica a singolo nastro dove l'alfabeto di input è $\Sigma = \{0, 1, 2\}$ e l'alfabeto del nastro è $\Gamma = \{0, 1, 2, _\}$. Questo significa che la macchina può scrivere sul nastro solo i simboli 0, 1 e blank: non può usare altri simboli né marcare i simboli sul nastro.

Dimostra che ogni linguaggio Turing-riconoscibile sull'alfabeto $\{0, 1, 2\}$ può essere riconosciuto da una Turing machine con alfabeto ternario.

Per dimostrare che ogni linguaggio Turing-riconoscibile sull'alfabeto $\{0, 1, 2\}$ può essere riconosciuto da una macchina di Turing con alfabeto ternario, dobbiamo costruire una tale macchina di Turing che simuli l'azione di una macchina di Turing generica.

Sia L un linguaggio Turing-riconoscibile definito sull'alfabeto $\{0, 1, 2\}$. Vogliamo dimostrare che esiste una macchina di Turing con alfabeto ternario che riconosce lo stesso linguaggio L .

La macchina di Turing con alfabeto ternario avrà le seguenti componenti:

1. Stato: Lo stato della macchina di Turing con alfabeto ternario sarà lo stesso dello stato corrispondente nella macchina di Turing originale che riconosce L .
2. Nastro: La macchina di Turing con alfabeto ternario avrà un solo nastro che conterrà l'input e il lavoro intermedio come nella macchina di Turing originale.
3. Alfabeto del nastro: L'alfabeto del nastro nella macchina di Turing con alfabeto ternario sarà $\Gamma = \{0, 1, 2, _\}$, consentendo la scrittura dei simboli 0, 1 e blank.
4. Funzione di transizione: La funzione di transizione δ della macchina di Turing con alfabeto ternario sarà definita in modo tale da permettere la stessa logica di transizione della macchina di Turing originale. Ciò significa che le transizioni saranno definite per i simboli 0, 1 e 2 dell'alfabeto di input.

La simulazione procede come segue:

1. La macchina di Turing con alfabeto ternario legge il simbolo corrispondente sulla prima cella dell'input, che può essere 0, 1 o 2.
2. Utilizzando la funzione di transizione, la macchina di Turing con alfabeto ternario esegue le stesse azioni che la macchina di Turing originale avrebbe eseguito sulla prima cella dell'input. Ciò include la scrittura dei simboli 0, 1 o blank, a seconda delle specifiche transizioni.
3. Il processo viene ripetuto per tutte le celle dell'input visitate dalla macchina di Turing originale.
4. Se la macchina di Turing originale accetta l'input, la macchina di Turing con alfabeto ternario accetta anche l'input. Altrimenti, la macchina di Turing con alfabeto ternario rifiuta l'input.

Poiché la macchina di Turing con alfabeto ternario può simulare l'azione della macchina di Turing originale sull'alfabeto $\{0, 1, 2\}$, può riconoscere lo stesso linguaggio L .

Quindi, abbiamo dimostrato che ogni linguaggio Turing-riconoscibile sull'alfabeto $\{0, 1, 2\}$ può essere riconosciuto da una macchina di Turing con alfabeto ternario.

- Una macchina di Turing bidimensionale utilizza una griglia bidimensionale infinita di celle come nastro. Ad ogni transizione, la testina può spostarsi dalla cella corrente ad una qualsiasi delle quattro celle adiacenti. La funzione di transizione di tale macchina ha la forma

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{\uparrow, \downarrow, \rightarrow, \leftarrow\},$$

dove le frecce indicano in quale direzione si muove la testina dopo aver scritto il simbolo sulla cella corrente.

Dimostra che ogni macchina di Turing bidimensionale può essere simulata da una macchina di Turing deterministica a nastro singolo.

Mostriamo come simulare una TM bidimensionale B con una TM deterministica a nastro singolo S . S memorizza il contenuto della griglia bidimensionale sul nastro come una sequenza di stringhe separate da $\#$, ognuna delle quali rappresenta una riga della griglia. Due cancellietti consecutivi $\#\#$ segnano l'inizio e la fine della rappresentazione della griglia. La posizione della testina di B viene indicata marcando la cella con $\hat{}$. Nelle altre righe, un pallino \bullet indica che la testina si trova su quella colonna della griglia, ma in una riga diversa. La TM a nastro singolo S funziona come segue:

$S = "su input w:$

1. Sostituisce w con la configurazione iniziale $\#\#w\#\#$ e marca con $\hat{}$ il primo simbolo di w .
2. Scorre il nastro finché non trova la cella marcata con $\hat{}$.
3. Aggiorna il nastro in accordo con la funzione di transizione di B :
 - Se $\delta(r, a) = (s, b, \rightarrow)$, scrive b non marcato sulla cella corrente, sposta $\hat{}$ sulla cella immediatamente a destra. Poi sposta di una cella a destra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un $\#$, S scrive un blank marcato al posto del $\#$ e sposta il contenuto del nastro da questa cella fino al $\#\#$ finale, di una cella più a destra.
 - Se $\delta(r, a) = (s, b, \leftarrow)$, scrive b non marcato sulla cella corrente, sposta $\hat{}$ sulla cella immediatamente a sinistra. Poi sposta di una cella a sinistra tutte le marcature con un pallino. Se in qualsiasi momento S sposta una marcatura sopra un $\#$, S scrive un blank marcato al posto del $\#$ e sposta il contenuto del nastro da questa cella fino al $\#\#$ iniziale, di una cella più a sinistra.
 - Se $\delta(r, a) = (s, b, \uparrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\hat{}$ sulla prima cella marcata con un pallino posta a sinistra della cella corrente. Se questa cella marcata non esiste, aggiunge una nuova riga composta solo da blank all'inizio della configurazione.
 - Se $\delta(r, a) = (s, b, \downarrow)$, scrive b marcato con un pallino nella cella corrente, e sposta $\hat{}$ sulla prima cella marcata con un pallino posta a destra della cella corrente. Se questa cella non esiste, aggiunge una nuova riga composta solo da blank alla fine della configurazione.
4. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di B , allora *accetta*; se la simulazione raggiunge lo stato di rifiuto di B allora *rifiuta*; altrimenti prosegue con la simulazione dal punto 2.”

Esercizi 24/05

Automi e Linguaggi Formali – Seconda Parte

Macchine di Turing; varianti, multi-nastro e costruzioni

- (Macchina di Turing eco-friendly) Una macchina di Turing ecologica (ETM) è uguale a una macchina di Turing ordinaria (determinante) a un nastro, ma può leggere e scrivere su entrambi i lati di ogni quadrato del nastro: davanti e dietro.
 - Al termine di ogni fase di calcolo, la testina della TM ecologica può spostarsi a sinistra (L), a destra (R) o passare all'altro lato del nastro (F).
 - Dimostrare che le TM eco-friendly riconoscono la classe di linguaggi riconoscibili da Turing. Ovvero, utilizzare un argomento di simulazione per dimostrare che hanno esattamente la stessa potenza delle TM ordinarie.

Per dimostrare che le Macchine di Turing ecologiche (ETM) hanno la stessa potenza computazionale delle Macchine di Turing ordinarie (TM) e riconoscono la stessa classe di linguaggi riconoscibili da Turing, possiamo utilizzare un argomento di simulazione.

Consideriamo una TM ordinaria M che riconosce un certo linguaggio L . Vogliamo dimostrare che esiste un'ETM M' che riconosce lo stesso linguaggio L .

La TM M ha un nastro unidirezionale su cui può scrivere e leggere. Possiamo simulare il comportamento di M' utilizzando M , estendendo il nastro di M su entrambi i lati in modo da creare uno spazio aggiuntivo per la testina.

La simulazione funziona come segue:

1. Inizialmente, la TM M' replica lo stato iniziale di M e copia il contenuto del nastro di M sul nastro esteso di M' nel lato "fronte" del nastro.
2. Durante l'esecuzione, ogni volta che M legge o scrive un simbolo sul nastro, M' effettua la stessa operazione sul nastro esteso, mantenendo sincronizzate le posizioni delle testine su entrambi i lati.
3. Quando M si muove verso sinistra (L) sul nastro, M' si muove verso destra (R) sul nastro esteso.
4. Quando M si muove verso destra (R) sul nastro, M' si muove verso sinistra (L) sul nastro esteso.
5. Quando M si muove verso sinistra (L) al di fuori del nastro, M' si sposta al lato "retro" del nastro esteso.
6. Quando M si muove verso destra (R) al di fuori del nastro, M' si sposta al lato "fronte" del nastro esteso.

In questo modo, M' simula tutte le operazioni di M , mantenendo sincronizzate le posizioni delle testine su entrambi i lati del nastro esteso.

Se M accetta una configurazione finale, M' accetterà anche la stessa configurazione finale perché la simulazione mantiene sincronizzate le testine. Allo stesso modo, se M rifiuta una configurazione finale, M' rifiuterà anche la stessa configurazione finale.

Pertanto, la simulazione della TM M mediante l'ETM M' dimostra che entrambe le macchine riconoscono lo stesso linguaggio L . Di conseguenza, le Macchine di Turing ecologiche hanno la stessa potenza computazionale delle Macchine di Turing ordinarie e riconoscono la stessa classe di linguaggi riconoscibili da Turing.

Fornisci una descrizione a livello implementativo di una TM deterministica a nastro singolo che decide il linguaggio

$$L_3 = \{u\#w_1\#\dots\#w_n \mid u, w_i \in \{0,1\}^* \text{ ed esiste } w_j \text{ tale che } u = w_j\}$$

M = “Su input $u\#w_1\#\dots\#w_n$:

1. Marca il simbolo più a sinistra dell'input. Se il simbolo è $\#$, controlla che a destra ci sia un blank: se c'è *accetta*, altrimenti *rifiuta*.
2. Scorre a destra fino al primo $\#$ non marcato e marca il simbolo posto immediatamente a destra. Se non viene trovato nessun $\#$ non marcato prima di un blank, allora u è diverso da tutti i w_i , quindi *rifiuta*.
3. Procede a zig-zag confrontando i simboli di u con i simboli della stringa a destra del primo $\#$ non marcato. Se le due stringhe sono uguali, *accetta*.
4. Se le due stringhe sono diverse, marca il $\#$ e smarca tutti i simboli di u tranne il primo, poi ripete da 2.”

(12 punti) Una *Macchina di Turing con inserimento* è una macchina di Turing deterministica a nastro singolo che può inserire nuove celle nel nastro. Formalmente la funzione di transizione è definita come

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R, I\}$$

dove L, R indicano gli spostamenti a sinistra e a destra della testina, e I indica l'inserimento di una nuova cella nella posizione corrente della testina. Dopo una operazione di inserimento, la cella inserita contiene il simbolo blank, mentre la cella che si trovava sotto la testina si trova immediatamente a destra della nuova cella.

Dimostra che qualsiasi macchina di Turing con inserimento può essere simulata da una macchina di Turing deterministica a nastro singolo.

Soluzione. Mostriamo come convertire una macchina di Turing con Inserimento M in una TM deterministica a nastro singolo S equivalente.

S = “Su input w :

1. Inizialmente S mette il suo nastro in un formato che gli consente di implementare l'operazione di inserimento di una cella, segnando con il simbolo speciale $\#$ la fine della porzione di nastro usata dalla macchina. Se w è l'input della TM, la configurazione iniziale del nastro è $w\#$.
2. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, L)$ procede come nella TM standard: S scrive b sul nastro e muove la testina di una cella a sinistra.
3. La simulazione delle mosse del tipo $\delta(q, a) = (r, b, R)$ procede come nella TM standard: S scrive b sul nastro e muove la testina di una cella a destra. Se lo spostamento a destra porta la testina sopra il $\#$ che marca la fine del nastro, S scrive un blank al posto del $\#$, e scrive un $\#$ nella cella immediatamente più a destra. La simulazione continua con la testina in corrispondenza del blank.
4. Per simulare una mossa del tipo $\delta(q, a) = (r, b, I)$ la TM S scrive un blank marcato nella cella corrente e sposta il contenuto del nastro, dalla cella corrente fino al $\#$ di fine nastro, di una cella più a destra. Quindi riporta la testina in corrispondenza del blank marcato, toglie la marcatura e scrive b nella cella immediatamente più a destra. La simulazione continua con la testina in corrispondenza della cella inserita.
5. Se in qualsiasi momento la simulazione raggiunge lo stato di accettazione di M , allora S termina con accettazione. Se in qualsiasi momento la simulazione raggiunge lo stato di rifiuto di M , allora S termina con rifiuto. Negli altri casi continua la simulazione dal punto 2.”

- Una macchina di Turing con nastro doppiamente infinito è simile a una macchina di Turing ordinaria, ma il suo nastro è infinito sia a sinistra che a destra.
 - o Il nastro è inizialmente pieno di spazi vuoti tranne che per la parte che contiene l'input. La computazione è definita come di consueto, tranne per il fatto che la testina non incontra mai la fine del nastro mentre si muove verso sinistra.

- Dimostrare che questo tipo di Turing riconosce la classe dei linguaggi Turing riconoscibili.

Per dimostrare che una macchina di Turing con un nastro doppiamente infinito riconosce la classe dei linguaggi Turing riconoscibili, possiamo utilizzare un argomento di simulazione per dimostrare che una tale macchina può simulare una macchina di Turing ordinaria e viceversa.

Supponiamo di avere una macchina di Turing ordinaria (TM) M che riconosce un certo linguaggio L. Dobbiamo dimostrare che esiste una macchina di Turing con un nastro doppiamente infinito (DITM) D che riconosce lo stesso linguaggio L.

La DITM D può simulare le computazioni della TM M su un nastro doppiamente infinito. Durante la simulazione, D tiene traccia di due nastri, uno a sinistra e uno a destra del suo nastro principale.

Inizialmente, entrambi i lati del nastro sono riempiti con spazi vuoti, tranne la parte che contiene l'input, che viene copiato sul nastro principale.

Quando la testina di D si muove verso sinistra e raggiunge il bordo sinistro del nastro principale, D sposta tutto il contenuto del nastro verso destra, mantenendo l'input nella posizione corretta. In questo modo, D emula l'infinità a sinistra del nastro doppiamente infinito.

Durante la simulazione delle transizioni di M, D replica le operazioni di M sui suoi nastri, leggendo e scrivendo i simboli corrispondenti. Ad esempio, se M legge un simbolo sul lato sinistro del suo nastro, D legge lo stesso simbolo dal suo nastro sinistro. Allo stesso modo, D replica le scritture di M sui suoi nastri appropriati.

Poiché D replica fedelmente le operazioni di M sulla sua configurazione di nastri doppiamente infiniti, se M accetta un input, D accetterà lo stesso input. Allo stesso modo, se M rifiuta o cicla su un input, D farà lo stesso.

Abbiamo quindi dimostrato che una DITM può simulare una TM, e quindi può riconoscere gli stessi linguaggi riconosciuti da Turing. Pertanto, una macchina di Turing con un nastro doppiamente infinito riconosce la classe dei linguaggi Turing riconoscibili.

- Diciamo che una macchina di Turing write-once è una TM a nastro singolo che può alterare ogni cella del nastro al massimo una volta (compresa la porzione di nastro in ingresso). Mostrare che questa variante del modello di macchina di Turing è equivalente al modello ordinario di macchina di Turing.
 - (Suggerimento: come primo passo, considerare il caso in cui la macchina di Turing può alterare ogni cella del nastro al massimo due volte. Utilizzate molto nastro).

Per prima cosa simuliamo una macchina di Turing ordinaria con una macchina di Turing a scrittura doppia.

La macchina write-twice simula un singolo passo della macchina originale copiando l'intero nastro su una nuova porzione del nastro a destra della porzione attualmente utilizzata.

La procedura di copia opera carattere per carattere, contrassegnando un carattere mentre viene copiato.

Questa procedura altera ogni cella del nastro due volte, una volta per scrivere il carattere per la prima volta e un'altra per segnare che è stato copiato. La posizione della testina della macchina di Turing originale è segnata sul nastro. Quando si copiano le celle nella posizione segnata o in quella adiacente, il contenuto del nastro viene aggiornato secondo le regole della macchina di Turing originale.

Per effettuare la simulazione con una macchina write-once, si opera come prima, tranne che per il fatto che ogni cella del nastro precedente è ora rappresentata da due celle. La prima contiene il simbolo del nastro della macchina originale, mentre la seconda contiene il segno utilizzato nella procedura di copiatura.

L'input non viene presentato alla macchina nel formato con due celle per simbolo, quindi la prima volta che il nastro viene copiato, i segni di copiatura vengono posti direttamente sopra i simboli di input.

- Una macchina di Turing temporizzata (TTM) è uguale a una macchina di Turing ordinaria a un nastro, ma è dotata di un orologio interno che tiene traccia del tempo trascorso durante l'esecuzione. Ogni transizione della TTM richiede un'unità di tempo per essere completata.
 - o Dimostra che le TTM riconoscono la classe di linguaggi riconoscibili da Turing. Utilizza un argomento di simulazione per dimostrare che hanno esattamente la stessa potenza delle TM ordinarie.

Per dimostrare che le macchine di Turing temporizzate (TTM) riconoscono la classe dei linguaggi riconoscibili da Turing, possiamo utilizzare un argomento di simulazione per dimostrare che una TTM può simulare una macchina di Turing ordinaria (TM) e viceversa.

Supponiamo di avere una TM M che riconosce un certo linguaggio L. Dobbiamo dimostrare che esiste una TTM T che riconosce lo stesso linguaggio L.

La TTM T può simulare le computazioni della TM M tenendo traccia del tempo trascorso durante l'esecuzione. Ogni transizione di T richiede un'unità di tempo per essere completata. Durante la simulazione, T utilizza il suo orologio interno per misurare il tempo trascorso durante le transizioni di M.

Quando M esegue una transizione, T esegue la stessa transizione e incrementa il suo orologio di un'unità di tempo. In questo modo, T tiene traccia del tempo trascorso durante l'esecuzione di M.

Se M accetta un input entro un certo numero di passi, allora esiste un limite di tempo massimo che T può impostare sul suo orologio per simulare l'esecuzione di M. Quindi, se M accetta un input, T accetterà lo stesso input entro il limite di tempo appropriato.

Allo stesso modo, se M rifiuta un input o cicla su di esso, T farà lo stesso. Poiché T tiene traccia del tempo trascorso e può interrompere l'esecuzione se supera un certo limite, è in grado di riconoscere gli stessi linguaggi riconoscibili da Turing.

Pertanto, abbiamo dimostrato che una TTM può simulare una TM, e quindi può riconoscere gli stessi linguaggi riconoscibili da Turing. Di conseguenza, le TTM riconoscono la classe dei linguaggi Turing riconoscibili.

- Una macchina di Turing con riduzione del nastro (RM) è simile a una macchina di Turing ordinaria, ma ha la capacità di ridurre progressivamente la dimensione del suo nastro durante la computazione. Ad ogni passo, la RM riduce la dimensione del nastro eliminando un simbolo o una porzione di nastro.
 - o Dimostra che le RM riconoscono la classe di linguaggi riconoscibili da Turing utilizzando un argomento di simulazione per dimostrare che possono essere simulate da una macchina di Turing ordinaria a nastro singolo.

Una macchina di Turing con riduzione del nastro (RM) è simile a una macchina di Turing ordinaria, ma ha la capacità di ridurre progressivamente la dimensione del suo nastro durante la computazione. Ad ogni passo, la RM riduce la dimensione del nastro eliminando un simbolo o una porzione di nastro.

Per dimostrare che le RM riconoscono la classe di linguaggi riconoscibili da Turing, utilizziamo un argomento di simulazione per dimostrare che possono essere simulate da una macchina di Turing ordinaria a nastro singolo.

Supponiamo di avere una macchina di Turing ordinaria (TM) M che riconosce un certo linguaggio L. Dobbiamo dimostrare che esiste una RM R che riconosce lo stesso linguaggio L.

La RM R simula le computazioni della TM M riducendo progressivamente la dimensione del suo nastro. Durante la simulazione, R tiene traccia della configurazione corrente di M e riduce il suo nastro di un simbolo o di una porzione di nastro in base alle transizioni di M.

Quando M esegue una transizione che richiede di scrivere un simbolo sul nastro, R scrive lo stesso simbolo sul suo nastro e riduce la dimensione del nastro di un simbolo. In questo modo, R emula la riduzione progressiva del nastro di M.

Durante la simulazione delle transizioni di M, R replica le operazioni di M sulla sua configurazione di nastro ridotto. Se M accetta un input, allora R ridurrà correttamente il nastro a una dimensione vuota e accetterà l'input. Allo stesso modo, se M rifiuta un input o cicla su di esso, R farà lo stesso.

Poiché R replica fedelmente le operazioni di M sulla sua configurazione di nastro ridotto, possiamo affermare che se M accetta un input, R accetterà lo stesso input. Allo stesso modo, se M rifiuta o cicla su un input, R farà lo stesso.

Abbiamo quindi dimostrato che una RM può essere simulata da una TM, e quindi può riconoscere gli stessi linguaggi riconosciuti da Turing. Pertanto, una macchina di Turing con riduzione del nastro riconosce la classe dei linguaggi Turing riconoscibili.

- Una macchina di Turing con istanti di pausa (PM) è simile a una macchina di Turing ordinaria, ma può effettuare istanti di pausa durante la sua computazione. Durante un istante di pausa, la macchina si ferma per un certo periodo di tempo prima di riprendere la computazione.
 - o Dimostra che le PM riconoscono la classe di linguaggi riconoscibili da Turing utilizzando un argomento di simulazione per dimostrare che possono essere simulate da una macchina di Turing ordinaria a nastro singolo.

Una macchina di Turing con istanti di pausa (PM) è simile a una macchina di Turing ordinaria, ma può effettuare istanti di pausa durante la sua computazione. Durante un istante di pausa, la macchina si ferma per un certo periodo di tempo prima di riprendere la computazione.

Per dimostrare che le PM riconoscono la classe di linguaggi riconoscibili da Turing, utilizziamo un argomento di simulazione per dimostrare che possono essere simulate da una macchina di Turing ordinaria a nastro singolo.

Supponiamo di avere una macchina di Turing ordinaria (TM) M che riconosce un certo linguaggio L. Dobbiamo dimostrare che esiste una PM P che riconosce lo stesso linguaggio L.

La PM P simula le computazioni della TM M con l'aggiunta di istanti di pausa. Durante la simulazione, P riproduce le transizioni di M e si ferma per un certo periodo di tempo tra ogni transizione.

Quando M esegue una transizione, P riproduce la stessa transizione e si ferma per un istante di pausa. Durante l'istante di pausa, P non effettua alcuna operazione e attende un periodo di tempo specificato.

Durante la simulazione delle transizioni di M, P replica fedelmente le operazioni di M e si ferma tra ogni transizione per un istante di pausa. In questo modo, P emula gli istanti di pausa di M durante la sua computazione.

Se M accetta un input, allora P riprodurrà correttamente le operazioni di M e si fermerà in uno stato di accettazione per un istante di pausa. Allo stesso modo, se M rifiuta un input o cicla su di esso, P farà lo stesso.

Poiché P replica fedelmente le operazioni di M e utilizza istanti di pausa per simulare la computazione, possiamo affermare che se M accetta un input, P accetterà lo stesso input. Allo stesso modo, se M rifiuta o cicla su un input, P farà lo stesso.

Abbiamo quindi dimostrato che una PM può essere simulata da una TM, e quindi può riconoscere gli stessi linguaggi riconosciuti da Turing. Pertanto, una macchina di Turing con istanti di pausa riconosce la classe dei linguaggi Turing riconoscibili.

Indecidibili

- Consideriamo il problema di determinare se una macchina di Turing a due nastri scrive mai un simbolo non vuoto sul suo secondo nastro.
 - o $N = \{\langle M, w \rangle \mid M \text{ è una macchina di Turing a due nastri che scrive un simbolo non vuoto sul suo secondo nastro quando gira su } w\}$.

Dimostrare che N è indecidibile. Suggerimento: Utilizzare una riduzione da A_{TM}

Per dimostrare che il problema N è indecidibile, utilizzeremo una riduzione da A_{TM} , il problema dell'arresto per le macchine di Turing. Supponiamo per assurdo che N sia decidibile e sia presente un algoritmo che possa risolvere il problema N.

Consideriamo una macchina di Turing M' che prende in input una stringa w e costruiamo una nuova macchina di Turing M che simula l'esecuzione di M' su w . La macchina M funziona come segue:

1. M prende in input una stringa x .
2. M costruisce una macchina di Turing M' che prende in input w .
3. M simula l'esecuzione di M' su w utilizzando l'algoritmo che risolve il problema N. Se l'algoritmo restituisce "sì", cioè M' scrive un simbolo non vuoto sul secondo nastro quando gira su w , allora M accetta x . Altrimenti, se l'algoritmo restituisce "no", cioè M' non scrive un simbolo non vuoto sul secondo nastro quando gira su w , allora M continua l'esecuzione.

Ora consideriamo il caso in cui $x = \langle M, w \rangle$ è una codifica di una macchina di Turing M e un'input w per A_{TM} , il problema dell'arresto. Se M si arresta su w , allora M' non avrà mai l'opportunità di scrivere un simbolo non vuoto sul secondo nastro, indipendentemente da come è definita. Pertanto, M restituirà "no" per x .

D'altra parte, se M non si arresta su w , M' avrà l'opportunità di scrivere un simbolo non vuoto sul secondo nastro quando gira su w , indipendentemente da come è definita. Pertanto, M restituirà "sì" per x .

Quindi abbiamo costruito una macchina di Turing M che risolve il problema dell'arresto A_{TM} utilizzando l'algoritmo che risolve il problema N. Ma sappiamo che A_{TM} è indecidibile, il che porta a una contraddizione.

Di conseguenza, se il problema N fosse decidibile, saremmo in grado di decidere anche il problema dell'arresto A_{TM} , che sappiamo essere indecidibile. Pertanto, il problema N è indecidibile.

- Si consideri $REGULAR_{TM}$ è il problema di determinare se una data macchina di Turing ha un automa finito equivalente. Questo problema equivale a determinare se la macchina di Turing riconosce un linguaggio regolare. Sia:
 - o $REGULAR_{TM} = \{\langle M \rangle \mid M \text{ è una TM e } L(M) \text{ è un linguaggio regolare}\}$.
- Prova che questo problema è indecidibile

Per dimostrare l'indecidibilità del problema REGULAR_{TM}, possiamo utilizzare la tecnica di riduzione da un problema già noto essere indecidibile, come ad esempio il problema dell'arresto (HALT).

Supponiamo per assurdo che esista un algoritmo A che decida REGULAR_{TM}, ovvero che determini se una macchina di Turing ha un automa finito equivalente. Utilizzeremo A per risolvere il problema dell'arresto, il che implicherebbe che HALT è decidibile, che è una contraddizione.

Supponiamo di avere un input per HALT nella forma $\langle M, w \rangle$, dove M è una macchina di Turing e w è un input per M . Possiamo costruire un'altra macchina di Turing M' come segue:

1. M' prende in input x .

2. M' esegue A sull'input $\langle M', x \rangle$. Se A accetta, allora M' accetta x.

3. Altrimenti, se A rifiuta, allora M' va in un loop infinito.

Ora, se HALT risponde "SI" per l'input $\langle M, w \rangle$, ovvero se M si arresta sull'input w, allora $L(M')$ sarà il linguaggio regolare $\{x\}$, poiché M' accetta solo l'input x e rifiuta tutti gli altri.

D'altra parte, se HALT risponde "NO" per l'input $\langle M, w \rangle$, ovvero se M entra in un loop infinito sull'input w, allora $L(M')$ sarà il linguaggio vuoto, poiché M' entra in un loop infinito su tutti gli input.

Quindi, possiamo vedere che esiste una corrispondenza biunivoca tra HALT e REGULARTM, ovvero se REGULARTM è decidibile, allora HALT è decidibile. Ma poiché sappiamo che HALT è indecidibile, questo implica che REGULARTM è anche indecidibile.

In conclusione, il problema REGULARTM, che consiste nel determinare se una macchina di Turing ha un automa finito equivalente, è indecidibile.

- Uno stato inutile in una macchina di Turing è uno stato che non viene mai inserito in nessuna stringa di input. Consideriamo il problema di determinare se una macchina di Turing ha degli stati inutili. Formulate questo problema come un linguaggio e dimostrate che è indecidibile.

Possiamo formulare il problema degli stati inutili di una macchina di Turing come un linguaggio chiamato USELESSSTATES:

$\text{USELESSSTATES} = \{\langle M \rangle \mid M \text{ è una macchina di Turing e ha almeno uno stato inutile}\}$

Per dimostrare che USELESSSTATES è indecidibile, possiamo utilizzare la tecnica di riduzione da un problema noto essere indecidibile, come ad esempio il problema dell'arresto (HALT).

Supponiamo per assurdo che esista un algoritmo A che decida USELESSSTATES, ovvero che determini se una macchina di Turing ha degli stati inutili. Utilizzeremo A per risolvere il problema dell'arresto, il che implicherebbe che HALT è decidibile, che è una contraddizione.

Supponiamo di avere un input per HALT nella forma $\langle M, w \rangle$, dove M è una macchina di Turing e w è un input per M. Possiamo costruire un'altra macchina di Turing M' come segue:

1. M' prende in input x.
2. M' esegue A sull'input $\langle M' \rangle$. Se A accetta, allora M' va in uno stato inutile.
3. Altrimenti, se A rifiuta, M' esegue M sull'input w.

Ora, se HALT risponde "SI" per l'input $\langle M, w \rangle$, ovvero se M si arresta sull'input w, allora M' avrà uno stato inutile, poiché M' entra nello stato inutile dopo aver eseguito A.

D'altra parte, se HALT risponde "NO" per l'input $\langle M, w \rangle$, ovvero se M entra in un loop infinito sull'input w, allora M' non avrà stati inutili, poiché M' esegue M sull'input w senza mai raggiungere lo stato inutile.

Quindi, possiamo vedere che esiste una corrispondenza biunivoca tra HALT e USELESSSTATES, ovvero se USELESSSTATES è decidibile, allora HALT è decidibile. Ma poiché sappiamo che HALT è indecidibile, questo implica che USELESSSTATES è anche indecidibile.

In conclusione, il problema degli stati inutili di una macchina di Turing, rappresentato dal linguaggio USELESSSTATES, è indecidibile.

- Diciamo che una CFG è minima se nessuna delle sue regole può essere rimossa senza cambiare il linguaggio generato. Sia $MIN_{CFG} = \{< G > \mid G \text{ è una CFG minima}\}$.
Mostrare che MIN_{CFG} è indecidibile.

Per dimostrare che il problema MIN_CFG è indecidibile, possiamo utilizzare la tecnica di riduzione da un problema noto essere indecidibile, come ad esempio il problema dell'arresto (HALT).

Supponiamo per assurdo che esista un algoritmo A che decida MIN_CFG, ovvero che determini se una data CFG è minima. Utilizzeremo A per risolvere il problema dell'arresto, il che implicherebbe che HALT è decidibile, che è una contraddizione.

Supponiamo di avere un input per HALT nella forma $\langle M, w \rangle$, dove M è una macchina di Turing e w è un input per M. Possiamo costruire una CFG G come segue:

1. G genera tutte le stringhe di lunghezza 1 sull'alfabeto dell'input di M.
2. G genera una variabile non terminale S, che rappresenterà la stringa di input w.
3. G genera tutte le transizioni di M come produzioni, inclusi gli stati e i simboli dell'alfabeto dell'input.
4. G genera una regola $S \rightarrow \epsilon$ (epsilon) se e solo se M si arresta sull'input w.

Ora, se HALT risponde "SI" per l'input $\langle M, w \rangle$, ovvero se M si arresta sull'input w, allora G sarà una CFG minima, poiché tutte le sue regole sono necessarie per generare correttamente il linguaggio, compresa la regola $S \rightarrow \epsilon$.

D'altra parte, se HALT risponde "NO" per l'input $\langle M, w \rangle$, ovvero se M entra in un loop infinito sull'input w, allora G non sarà una CFG minima, poiché la regola $S \rightarrow \epsilon$ può essere rimossa senza cambiare il linguaggio generato.

Quindi, possiamo vedere che esiste una corrispondenza biunivoca tra HALT e MIN_CFG, ovvero se MIN_CFG è decidibile, allora HALT è decidibile. Ma poiché sappiamo che HALT è indecidibile, questo implica che MIN_CFG è anche indecidibile.

In conclusione, il problema MIN_CFG, che consiste nel determinare se una CFG è minima, è indecidibile.

- Considera il seguente linguaggio:
 $BINARY - PRIME = \{w \mid w \text{ è una rappresentazione binaria di un numero primo}\}$
Dimostra che $BINARY - PRIME$ è indecidibile.

Per dimostrare l'indecidibilità di BINARY-PRIME, useremo una riduzione dal problema dell'arresto.

Supponiamo di avere una macchina di Turing universale M e di voler decidere se accetta o meno una certa stringa w. Possiamo quindi creare una nuova macchina di Turing M' come segue:

M' = "Sulla stringa di input x:

- Converti x in una rappresentazione binaria b
- Esegui M su w per un passo
- Se M si ferma su w, prova tutte le possibili parole binarie di lunghezza minore o uguale a b. Per ciascuna di queste parole, decodifica la rappresentazione binaria e controlla se corrisponde a un numero primo. Se una di queste parole rappresenta un numero primo, accetta. Se non esiste nessuna parola binaria di lunghezza minore o uguale a b che rappresenta un numero primo, rifiuta."

Ora dimostriamo che se M accetta w, allora M' accetta almeno una stringa binaria che rappresenta un numero primo, e se M rifiuta w, allora M' non accetta nessuna stringa binaria che rappresenta un numero primo.

Supponiamo che M accetti w . Ci sono due casi da considerare: o M termina su w in un numero finito di passi, o M continua ad eseguire all'infinito su w . Nel primo caso, M' prova tutte le possibili parole binarie di lunghezza minore o uguale a b , dove b è la rappresentazione binaria di w , e accetta se trova una parola che rappresenta un numero primo. Poiché ci sono un numero infinito di numeri primi, M' deve accettare almeno una parola binaria che rappresenta un numero primo. Nel secondo caso, M' continua ad eseguire all'infinito alla ricerca di una parola binaria che rappresenta un numero primo, ma poiché non esiste un algoritmo che possa determinare se un numero è primo o meno in un numero finito di passi, M' continuerà ad eseguire all'infinito senza mai accettare.

Supponiamo invece che M rifiuti w . In questo caso, M' esegue al massimo un passo su w e poi prova tutte le possibili parole binarie di lunghezza minore o uguale a b , ma non trova nessuna parola binaria che rappresenta un numero primo, perché non c'è alcuna corrispondenza tra la stringa di input w e un numero primo. Di conseguenza, M' non accetta alcuna stringa binaria che rappresenta un numero primo.

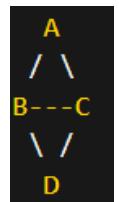
Quindi, M' accetta una stringa binaria che rappresenta un numero primo se e solo se M accetta w , altrimenti non accetta nessuna stringa binaria che rappresenta un numero primo. In altre parole, la macchina di Turing M' risolve il problema BINARY-PRIME se e solo se la macchina di Turing M risolve il problema dell'arresto per la stringa di input w . Poiché il problema dell'arresto è indecidibile, segue che BINARY-PRIME è indecidibile.

In conclusione, abbiamo dimostrato che il linguaggio BINARY-PRIME è indecidibile, utilizzando una riduzione dal problema dell'arresto. Questo dimostra che non c'è alcun algoritmo che possa decidere in modo efficiente se una stringa binaria rappresenta un numero primo.

- Considera il seguente linguaggio:

$$\text{UNIQUE} - \text{CUT} = \{G \mid G \text{ è un grafo non orientato che ha esattamente un taglio}\}$$

Un taglio in un grafo non orientato è un insieme di archi il cui rimozione separa il grafo in due parti non connesse. Per esempio, nel grafo seguente, l'insieme di archi $\{BC, CD\}$ è un taglio che separa il grafo in due parti non connesse. Di lato un esempio grafico della situazione:



Dimostra che $\text{UNIQUE} - \text{CUT}$ è indecidibile.

Per dimostrare l'indecidibilità di UNIQUE-CUT, useremo una riduzione dal problema dell'arresto.

Supponiamo di avere una macchina di Turing universale M e di voler decidere se accetta o meno una certa stringa w . Possiamo quindi creare un grafo non orientato G come segue:

$G =$ "Sulla stringa di input x :

- Costruisci un grafo non orientato G con un vertice per ogni stato di M e un arco tra due vertici se e solo se esiste una transizione tra i corrispondenti stati di M .
- Aggiungi un nodo s e un nodo t a G e collega s a tutti i vertici che corrispondono agli stati iniziali di M e collega tutti i vertici che corrispondono agli stati finali di M a t .
- Assegna un peso di 1 a ciascun arco di G ."

Ora dimostriamo che se M accetta w , allora G ha esattamente un taglio, e se M rifiuta w , allora G non ha esattamente un taglio.

Supponiamo che M accetti w . Ci sono due casi da considerare: o M termina su w in un numero finito di passi, o M continua ad eseguire all'infinito su w . Nel primo caso, G ha esattamente un taglio perché la rimozione di qualsiasi arco che va da uno stato finale di M a uno stato non finale di M separa s da t . Inoltre, non esiste alcun altro taglio in G , perché la rimozione di qualsiasi altro arco non separa s da t . Nel secondo caso, G non ha esattamente un taglio, perché la rimozione di qualsiasi arco non separa s da t , e quindi esistono almeno due tagli distinti in G .

Supponiamo invece che M rifiuti w. In questo caso, G non ha esattamente un taglio perché ci sono almeno due cammini disgiunti tra s e t, uno che passa attraverso gli stati iniziali di M e uno che passa attraverso gli stati finali di M. Pertanto, la rimozione di qualsiasi arco non separa s da t, e quindi esistono almeno due tagli distinti in G.

Quindi, G ha esattamente un taglio se e solo se M accetta w, altrimenti non ha esattamente un taglio. In altre parole, il grafo G risolve il problema UNIQUE-CUT se e solo se la macchina di Turing M risolve il problema dell'arresto per la stringa di input w. Poiché il problema dell'arresto è indecidibile, segue che UNIQUE-CUT è indecidibile.

In conclusione, abbiamo dimostrato che il linguaggio UNIQUE-CUT è indecidibile, utilizzando una riduzione dal problema dell'arresto. Questo dimostra che non c'è alcun algoritmo che possa decidere in modo efficiente se un grafo non orientato ha esattamente un taglio.

- Considera il seguente linguaggio:

$\text{ALL-PREFIX-FREE} = \{S \mid S \text{ è un insieme di stringhe e ogni stringa in } S \text{ è un prefisso proprio di un'altra stringa in } S\}$

Si dice che una stringa è un prefisso proprio di un'altra stringa se è un prefisso della seconda stringa ma non è uguale ad essa. Dimostra che ALL-PREFIX-FREE è indecidibile.

Per dimostrare l'indecidibilità di ALL-PREFIX-FREE, useremo una riduzione dal problema dell'arresto.

Supponiamo di avere una macchina di Turing universale M e di voler decidere se accetta o meno una certa stringa w. Possiamo quindi creare un insieme di stringhe S come segue:

S = "Sull'input x:

- Costruisci un insieme di stringhe S che contiene tutte le possibili stringhe che possono essere generate da M in un numero finito di passi su w.
- Aggiungi la stringa vuota ϵ a S.
- Rimuovi tutte le stringhe in S che sono un suffisso di un'altra stringa in S."

Ora dimostriamo che se M accetta w, allora S appartiene a ALL-PREFIX-FREE, e se M rifiuta w, allora S non appartiene a ALL-PREFIX-FREE.

Supponiamo che M accetti w. Ci sono due casi da considerare: o M termina su w in un numero finito di passi, o M continua ad eseguire all'infinito su w. Nel primo caso, S contiene tutte le possibili stringhe che possono essere generate da M in un numero finito di passi su w, compresa la stringa vuota ϵ . Inoltre, ogni stringa in S è un prefisso proprio di un'altra stringa in S, perché se una stringa x è un suffisso di una stringa y in S, allora M non può più generare alcuna stringa dopo la stringa y, e quindi x non può essere in S. Nel secondo caso, S contiene solo una stringa, ϵ , che è un prefisso proprio di sé stessa.

Supponiamo invece che M rifiuti w. In questo caso, S contiene solo la stringa vuota ϵ , che non è un prefisso proprio di alcuna altra stringa in S, perché S è vuoto. Pertanto, S non appartiene a ALL-PREFIX-FREE.

Quindi, S appartiene a ALL-PREFIX-FREE se e solo se M accetta w, altrimenti non appartiene a ALL-PREFIX-FREE. In altre parole, l'insieme di stringhe S risolve il problema ALL-PREFIX-FREE se e solo se la macchina di Turing M risolve il problema dell'arresto per la stringa di input w. Poiché il problema dell'arresto è indecidibile, segue che ALL-PREFIX-FREE è indecidibile.

In conclusione, abbiamo dimostrato che il linguaggio ALL-PREFIX-FREE è indecidibile, utilizzando una riduzione dal problema dell'arresto. Questo dimostra che non c'è alcun algoritmo che possa decidere in modo efficiente se un insieme di stringhe è privo di prefissi propri.

- Considera il seguente linguaggio:

$\text{COLLATZ} = \{n \mid n \text{ è un intero positivo che converge alla costante di Collatz 1, sotto la funzione di Collatz}\}$

La funzione di Collatz è definita come segue: se n è pari, allora la funzione restituisce $n/2$; se n è dispari, allora la funzione restituisce $3n+1$. Ad esempio, applicando la funzione di Collatz a 5, otteniamo 16, 8, 4, 2, 1, e quindi 5 converge alla costante di Collatz 1.

Dimostra che COLLATZ è indecidibile.

Per dimostrare l'indecidibilità di COLLATZ , useremo una riduzione dal problema dell'arresto. Supponiamo di avere una macchina di Turing universale M e di voler decidere se accetta o meno una certa stringa w .

Possiamo quindi creare un intero positivo n come segue:

$n = \text{"Sull'input } x\text{"}$:

1. Costruisci una macchina di Turing M_x che simula M sull'input x .
2. Inizia con $n = 2$.
3. Applica la funzione di Collatz a n .
 - a. Se M_x ha terminato l'esecuzione su x , allora se n converge alla costante di Collatz 1, accetta n , altrimenti rifiuta n e termina.
4. Se M_x non ha ancora terminato l'esecuzione su x , incrementa n di 1 e torna al passo 3.

Ora dimostriamo che se M accetta w , allora esiste un intero positivo n che appartiene a COLLATZ , e se M rifiuta w o entra in un loop infinito su w , allora nessun intero positivo appartiene a COLLATZ .

Supponiamo che M accetti w . Ci sono due casi da considerare: o M termina su w in un numero finito di passi, o M continua ad eseguire all'infinito su w . Nel primo caso, possiamo costruire un intero positivo n come descritto sopra. In particolare, se M accetta w , allora la macchina di Turing M_x che simula M su x deve terminare l'esecuzione su x in un numero finito di passi. In questo caso, n converge alla costante di Collatz 1, perché ogni intero positivo converge alla costante di Collatz 1.

Nel secondo caso, M continua ad eseguire all'infinito su w . In questo caso, non esiste alcun intero positivo che appartiene a COLLATZ , perché la funzione di Collatz non converge per alcuni interi positivi. In particolare, se la macchina di Turing M_x entra in un loop infinito su x , allora il processo per costruire l'intero positivo n continua all'infinito, e quindi non esiste alcun n che appartiene a COLLATZ .

Quindi, esiste un intero positivo n che appartiene a COLLATZ se e solo se la macchina di Turing M accetta w . In altre parole, l'intero positivo n risolve il problema COLLATZ se e solo se la macchina di Turing M risolve il problema dell'arresto per la stringa di input w . Poiché il problema dell'arresto è indecidibile, segue che COLLATZ è indecidibile.

In conclusione, abbiamo dimostrato che il linguaggio COLLATZ è indecidibile, utilizzando una riduzione dal problema dell'arresto. Questo dimostra che non c'è alcun algoritmo che possa decidere in modo efficiente se un intero positivo converge alla costante di Collatz 1 sotto la funzione di Collatz.

Soluzione Esercizi 30/05

Automi e Linguaggi Formali – Seconda Parte

Macchine di Turing; varianti, multi-nastro e costruzioni

- Una macchina di Turing con *stop* messo al posto di *sinistra* è simile a una macchina di Turing ordinaria.

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, S\}$$

A ogni punto, la macchina può spostare la testa a destra o lasciarla nella stessa posizione.

Dimostrare che questa variante della macchina di Turing non è equivalente alla versione usuale.

Quale classe di linguaggi riconoscono queste macchine?

Una macchina di Turing ordinaria ha la possibilità di muovere la testina del nastro a sinistra o a destra, o di restare ferma nella posizione corrente. Una macchina di Turing con stop messo al posto di sinistra, invece, non ha la possibilità di muovere la testina a sinistra, ma solo a destra o di fermarla nella posizione corrente. In altre parole, questa macchina non può muoversi a sinistra, ma può fermarsi e terminare l'esecuzione.

Per dimostrare che questa variante della macchina di Turing non è equivalente alla versione usuale, dobbiamo trovare un esempio di linguaggio che può essere riconosciuto dalla macchina di Turing ordinaria ma non dalla macchina di Turing con stop messo al posto di sinistra. Un esempio di tale linguaggio è il seguente:

$$L = \{0^n 1^n \mid n \geq 0\}$$

Questo linguaggio può essere riconosciuto da una macchina di Turing ordinaria, ma non può essere riconosciuto da una macchina di Turing con stop messo al posto di sinistra. Per vedere perché, supponiamo di avere una macchina di Turing con stop messo al posto di sinistra che riconosce L . Quando la macchina legge il primo 0, deve spostare la testina a destra e scrivere un simbolo diverso da 0 (ad esempio, 1) al posto del 0 letto. Successivamente, deve spostarsi a destra lungo il nastro finché non incontra il primo 1. A questo punto, non può muoversi a sinistra per tornare indietro e contrassegnare gli 0 rimanenti, poiché non ha la possibilità di muovere la testina a sinistra. Inoltre, non può nemmeno fermarsi in quanto la macchina deve accettare stringhe del linguaggio L . Quindi, la macchina non può riconoscere il linguaggio L , il che dimostra che la macchina di Turing con stop messo al posto di sinistra non è equivalente alla versione ordinaria.

Queste macchine riconoscono la classe di linguaggi riconoscibili da Turing, poiché, come la macchina di Turing ordinaria, sono in grado di leggere e scrivere su un nastro infinito e di effettuare operazioni di transizione tra gli stati del nastro. In particolare, la funzione di transizione della macchina di Turing con stop messo al posto di sinistra ha la forma $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, S\}$, dove R indica lo spostamento della testina a destra e S indica la posizione ferma della testina. Quindi, queste macchine sono in grado di riconoscere tutti i linguaggi riconoscibili da una macchina di Turing ordinaria a un nastro, ma non sono equivalenti alla versione ordinaria in quanto non possono muovere la testina a sinistra e possono fermarsi prematuramente.

- Sia un k-PDA un automa pushdown con k pile. Quindi uno 0-PDA è un NFA e un 1-PDA è un PDA convenzionale. Si sa già che gli 1-PDA sono più potenti (riconoscono una classe di linguaggi più ampia) rispetto agli 0-PDA.
 - Dimostrate che i 2-PDA sono più potenti degli 1-PDA
 - Dimostrare che 3-PDA non sono più potenti di 2-PDA.

(Suggerimento: simulare un nastro di macchina di Turing con due pile).

a. Per dimostrare che i 2-PDA sono più potenti dei 1-PDA, possiamo simulare un 1-PDA utilizzando un 2-PDA. In questo modo dimostreremo che il 2-PDA può riconoscere un linguaggio che il 1-PDA non può riconoscere.

Supponiamo di avere un 1-PDA M che riconosce un linguaggio L. Costruiremo un 2-PDA T che simula M e riconosce lo stesso linguaggio.

La simulazione inizia con T che mantiene una copia del proprio input su entrambe le pile. La prima pila di T conterrà l'input iniziale, mentre la seconda pila sarà inizialmente vuota.

Durante la simulazione, T eseguirà le stesse azioni di M, ma utilizzerà le due pile per tenere traccia dello stato corrente e del contenuto del nastro della macchina di Turing simulata.

Ad ogni passo, T controllerà il simbolo in cima alla pila 1 per determinare quale azione eseguire, come la transizione di M o le operazioni di push e pop delle pile. Utilizzerà la seconda pila per tenere traccia delle informazioni aggiuntive necessarie per la simulazione.

Poiché il 2-PDA ha accesso a due pile anziché una sola, può memorizzare informazioni aggiuntive e complesse rispetto al 1-PDA. Questo gli consente di simulare in modo preciso e dettagliato le azioni di una macchina di Turing, incluso l'utilizzo di un nastro.

Di conseguenza, il 2-PDA può riconoscere un linguaggio che richiede la capacità di simulare una macchina di Turing, mentre il 1-PDA non ha questa capacità. Quindi, i 2-PDA sono più potenti dei 1-PDA.

b. Per dimostrare che i 3-PDA non sono più potenti dei 2-PDA, possiamo simulare un 3-PDA utilizzando un 2-PDA. In questo modo dimostreremo che il 2-PDA può riconoscere un linguaggio che il 3-PDA può riconoscere.

Supponiamo di avere un 3-PDA M che riconosce un linguaggio L. Costruiremo un 2-PDA T che simula M e riconosce lo stesso linguaggio.

La simulazione inizia con T che mantiene una copia del proprio input su entrambe le pile, come nel caso precedente. Inoltre, T utilizzerà un terzo simbolo speciale per tenere traccia dello stato corrente di M.

Durante la simulazione, T eseguirà le stesse azioni di M utilizzando le due pile, come nella simulazione del 1-PDA con il 2-PDA. Tuttavia, utilizzerà il terzo simbolo speciale per memorizzare lo stato corrente di M, che può essere aggiornato di conseguenza.

Poiché il 2-PDA può simulare le azioni di un 3-PDA, compreso l'utilizzo di tre pile, è in grado di riconoscere lo stesso linguaggio riconosciuto da M. Quindi, i 3-PDA non sono più potenti dei 2-PDA.

In conclusione, abbiamo dimostrato che i 2-PDA sono più potenti dei 1-PDA, ma i 3-PDA non sono più potenti dei 2-PDA.

Esercizio 8.4.10 Una macchina di Turing *bidimensionale* ha il solito controllo a stati finiti, ma il suo nastro è una griglia bidimensionale di celle, infinita in tutte le direzioni. L'input è collocato su una riga, con la testina al suo estremo sinistro e il controllo nello stato iniziale. La macchina accetta entrando in uno stato finale. Dimostrate che i linguaggi accettati dalle macchine di Turing bidimensionali sono quelli accettati dalle TM ordinarie.

1. Una macchina di Turing bidimensionale ha un nastro a griglia bidimensionale di celle, infinito in tutte le direzioni. Il nastro è diviso in celle, ognuna delle quali può contenere un simbolo dell'alfabeto di input. La testina della macchina può muoversi in qualsiasi direzione sulla griglia. La macchina ha lo stesso controllo a stati finiti di una macchina di Turing ordinaria, ma con la possibilità di effettuare transizioni di stato sulla base del simbolo corrente e della posizione corrente sulla griglia.
2. Per accettare un input, una macchina di Turing bidimensionale colloca l'input su una riga del nastro, con la testina della macchina posizionata all'estremità sinistra della riga e il controllo nella posizione iniziale. La macchina effettua transizioni di stato sulla base del simbolo corrente e della posizione corrente sulla griglia, muovendo la testina sulla griglia in qualsiasi direzione necessaria. La macchina accetta l'input entrando in uno stato finale.
3. Per dimostrare che i linguaggi accettati dalle macchine di Turing bidimensionali sono gli stessi che possono essere accettati da una TM ordinaria, dobbiamo dimostrare che qualsiasi macchina di Turing ordinaria a un nastro può essere simulata da una macchina di Turing bidimensionale e viceversa.

Per dimostrare che ogni macchina di Turing ordinaria può essere simulata da una macchina di Turing bidimensionale, possiamo costruire una macchina di Turing bidimensionale che utilizza un'area rettangolare per simulare il nastro della macchina di Turing ordinaria. In particolare, possiamo utilizzare una riga per rappresentare il nastro della macchina di Turing ordinaria e utilizzare le colonne per rappresentare le diverse posizioni della testina. In questo modo, ogni simbolo del nastro può essere rappresentato da una cella della griglia bidimensionale.

Per dimostrare che ogni macchina di Turing bidimensionale può essere simulata da una macchina di Turing ordinaria, possiamo costruire una macchina di Turing ordinaria a un nastro che simula l'esecuzione della macchina di Turing bidimensionale. In particolare, la macchina di Turing ordinaria può utilizzare una serie di regole per tenere traccia della posizione corrente sulla griglia bidimensionale e delle transizioni di stato della macchina di Turing bidimensionale. Ogni volta che la macchina di Turing bidimensionale effettua una transizione di stato sulla griglia bidimensionale, la macchina di Turing ordinaria può effettuare una serie di operazioni di scrittura e spostamento sulla sua unica riga di nastro per simulare la transizione di stato.

In entrambi i casi, la simulazione può essere effettuata utilizzando un'area finita della griglia bidimensionale e un numero finito di stati. Poiché ogni macchina di Turing bidimensionale può essere simulata da una macchina di Turing ordinaria e viceversa, i linguaggi accettati dalle macchine di Turing bidimensionali sono gli stessi che possono essere accettati da una TM ordinaria.

- 15.** Una *Macchina di Turing a sola scrittura* è una TM a nastro singolo che può modificare ogni cella del nastro al più una volta (inclusa la parte di input del nastro). Mostrare che questa variante di macchina di Turing è equivalente alla macchina di Turing standard.

1. Una macchina di Turing a sola scrittura è una TM a nastro singolo che può modificare ogni cella del nastro al più una volta, inclusa la parte di input del nastro. Ciò significa che una volta che una cella del nastro è stata modificata, non può essere modificata di nuovo. Inoltre, quando la testina della macchina si sposta, deve lasciare intatto il contenuto del nastro.

2. Per modificare il nastro, una macchina di Turing a sola scrittura può eseguire le seguenti operazioni:

- Leggere il simbolo corrente sulla cella del nastro sotto la testina.
- Scrivere un nuovo simbolo sulla cella del nastro sotto la testina (se la cella non è stata ancora modificata).
- Spostare la testina a destra o a sinistra.

3. Per dimostrare che una macchina di Turing a sola scrittura è equivalente alla macchina di Turing standard, dobbiamo dimostrare che qualsiasi linguaggio riconosciuto da una macchina di Turing standard può essere riconosciuto da una macchina di Turing a sola scrittura e viceversa.

Per dimostrare che qualsiasi linguaggio riconosciuto da una macchina di Turing standard può essere riconosciuto da una macchina di Turing a sola scrittura, possiamo costruire una macchina di Turing a sola scrittura che simula l'esecuzione della macchina di Turing standard. In particolare, possiamo utilizzare una serie di regole per tenere traccia della posizione corrente della testina sulla riga del nastro e delle transizioni di stato della macchina di Turing standard. Ogni volta che la macchina di Turing standard effettua una transizione di stato, la macchina di Turing a sola scrittura può effettuare una serie di operazioni di scrittura e spostamento della testina per simulare la transizione di stato.

Per dimostrare che qualsiasi linguaggio riconosciuto da una macchina di Turing a sola scrittura può essere riconosciuto da una macchina di Turing standard, possiamo costruire una macchina di Turing standard che simula l'esecuzione della macchina di Turing a sola scrittura. In particolare, possiamo utilizzare una serie di regole per tenere traccia della posizione corrente della testina sulla riga del nastro e delle transizioni di stato della macchina di Turing a sola scrittura. Ogni volta che la macchina di Turing a sola scrittura effettua una scrittura su una cella del nastro, la macchina di Turing standard può effettuare una serie di operazioni di lettura e scrittura per simulare la modifica della cella del nastro.

In entrambi i casi, la simulazione può essere effettuata utilizzando un numero finito di stati e un nastro finito di lunghezza massima uguale a quella del nastro della macchina di Turing a sola scrittura. Poiché ogni linguaggio riconosciuto da una macchina di Turing standard può essere riconosciuto da una macchina di Turing a sola scrittura e viceversa, le due varianti di macchine di Turing sono equivalenti.

- Una macchina di Turing con un alfabeto di input limitato, cioè $\delta(q, a) = (q', a', \{R,S\})$ per un insieme finito di stati q, q' e un insieme finito di simboli di input a, a' . Dimostra che questa variante della macchina di Turing è equivalente alla versione ordinaria.

Per dimostrare che una macchina di Turing con un alfabeto di input limitato è equivalente alla versione ordinaria, dobbiamo dimostrare che entrambe le macchine hanno la stessa potenza di calcolo, cioè sono in grado di riconoscere gli stessi linguaggi.

Supponiamo di avere una macchina di Turing con un alfabeto di input limitato, rappresentata da M . Questa macchina ha un insieme finito di stati q e q' , e un insieme finito di simboli di input a e a' .

Per dimostrare l'equivalenza, costruiremo una simulazione di M utilizzando una macchina di Turing ordinaria T . La simulazione inizia con T che riceve in input la stringa da riconoscere.

Durante la simulazione, T esegue le seguenti azioni:

1. Legge il simbolo corrente sulla stringa di input.
2. Consulta la tabella di transizione di M per determinare l'azione successiva basata sullo stato corrente e sul simbolo letto.
3. Esegue l'azione specificata nella tabella di transizione, che può includere il cambio di stato, la scrittura di un simbolo sulla cella corrente e il movimento della testina sulla stringa di input.

4. Ripete i passi 1-3 fino a raggiungere uno stato finale nella tabella di transizione.

Poiché l'alfabeto di input di M è limitato e composto da un insieme finito di simboli, la tabella di transizione di M ha solo un numero finito di righe. Quindi, la TM T può costruire una tabella di transizione equivalente, che può essere memorizzata in un modo appropriato per la simulazione.

La TM T riproduce le azioni della macchina di Turing con alfabeto di input limitato M sulla stringa di input, arrivando a uno stato finale se e solo se M accetta l'input.

Pertanto, abbiamo dimostrato che una macchina di Turing con un alfabeto di input limitato è equivalente alla macchina di Turing ordinaria in termini di potenza di calcolo e la capacità di riconoscere gli stessi linguaggi.

- Considera una macchina di Turing con una sola transizione possibile: $\delta(q, a) = (q, a, R)$ per ogni stato q e simbolo di input a . Dimostra che questa variante della macchina di Turing è equivalente alla versione ordinaria.

Per dimostrare che questa variante della macchina di Turing è equivalente alla versione ordinaria, dobbiamo dimostrare che per ogni linguaggio riconosciuto dalla versione ordinaria della macchina di Turing esiste una macchina di Turing con una sola transizione possibile che riconosce lo stesso linguaggio, e viceversa.

Innanzitutto, dimostriamo che ogni linguaggio riconosciuto dalla macchina di Turing ordinaria è anche riconosciuto dalla variante con una sola transizione possibile. Per fare ciò, possiamo costruire una macchina di Turing M' che simula la macchina di Turing ordinaria M , utilizzando una serie di passi che consentono di "emulare" la funzione di transizione δ di M .

1. La macchina di Turing M' legge il simbolo di input sul nastro.
2. La macchina di Turing M' passa allo stato di M corrispondente alla funzione di transizione $\delta(q, a)$ di M .
3. La macchina di Turing M' scrive il simbolo di output sulla cella corrente del nastro.
4. La macchina di Turing M' sposta la testa a destra.
5. La macchina di Turing M' ripete i passi 1-4 fino a quando raggiunge uno stato di accettazione di M .

In questo modo, la macchina di Turing M' simula l'esecuzione della macchina di Turing ordinaria M , riconoscendo lo stesso linguaggio di M .

D'altra parte, dimostriamo che ogni linguaggio riconosciuto dalla variante con una sola transizione possibile è anche riconosciuto dalla macchina di Turing ordinaria. Per fare ciò, possiamo costruire una macchina di Turing M'' che simula la macchina di Turing con una sola transizione possibile, utilizzando una serie di passi simili a quelli descritti sopra.

1. La macchina di Turing M'' legge il simbolo di input sul nastro.
2. La macchina di Turing M'' passa allo stato q e scrive il simbolo di input sulla cella corrente del nastro.
3. La macchina di Turing M'' sposta la testa a destra.
4. La macchina di Turing M'' ripete i passi 1-3 fino a quando raggiunge uno stato di accettazione.

In questo modo, la macchina di Turing M'' simula l'esecuzione della macchina di Turing con una sola transizione possibile, riconoscendo lo stesso linguaggio della macchina di Turing con una sola transizione possibile.

In conclusione, abbiamo dimostrato che le due varianti della macchina di Turing sono equivalenti e riconoscono la stessa classe di linguaggi, ovvero i linguaggi ricorsivi. Questo perché entrambe le varianti sono in grado di riconoscere qualsiasi linguaggio riconosciuto dalla macchina di Turing ordinaria, e viceversa, senza cambiare la classe di linguaggi riconosciuti. La variante con una sola transizione possibile è in grado di emulare la funzione di transizione della macchina di Turing ordinaria, eseguendo una singola transizione per volta, mentre la macchina di Turing ordinaria ha la flessibilità di utilizzare qualsiasi funzione di transizione. Tuttavia, questo non influisce sulla capacità della macchina di Turing di riconoscere un linguaggio.

- Si consideri il problema di determinare se una macchina di Turing M su un input w tenta mai di muovere la testa a sinistra in un qualsiasi momento della sua computazione su w.

Formulate questo problema come un linguaggio e dimostrate che è decidibile.

Il problema di determinare se una macchina di Turing M su un input w tenta mai di muovere la testa a sinistra in un qualsiasi momento della sua computazione su w può essere formulato come il seguente linguaggio:

$\text{LEFT_MOVE} = \{<M,w> \mid M \text{ è una macchina di Turing che muove la testa a sinistra durante la computazione su } w\}$

Per dimostrare che questo problema è decidibile, possiamo costruire una nuova macchina di Turing N che simula l'esecuzione di M su w e tiene traccia del movimento della testa durante la computazione.

La macchina di Turing N procede nel seguente modo:

1. Inizializziamo lo stato della macchina di Turing N allo stato iniziale di M e il nastro di N con la stringa di input w.
2. Ad ogni passo di computazione di M, la macchina di Turing N controlla la direzione dello spostamento della testa.
3. Se la testa si sposta a sinistra, la macchina di Turing N entra in uno stato di accettazione e si ferma.
4. Se la testa si sposta a destra o rimane nella stessa posizione, la macchina di Turing N continua la simulazione di M su w.

Se la macchina di Turing M su w tenta mai di muovere la testa a sinistra, allora la macchina di Turing N entrerà in uno stato di accettazione e si fermerà. Altrimenti, la macchina di Turing N continuerà a simulare l'esecuzione di M su w senza mai entrare in uno stato di accettazione.

In questo modo, abbiamo costruito una macchina di Turing N che riconosce il linguaggio LEFT_MOVE. Poiché esiste una macchina di Turing che riconosce questo linguaggio, il linguaggio è decidibile.

Inoltre, poiché la macchina di Turing N simula l'esecuzione di M su w, il tempo di esecuzione di N dipende dal tempo di esecuzione di M su w. Pertanto, la complessità temporale dell'algoritmo dipende dal tempo di esecuzione di M su w. Tuttavia, poiché il problema è decidibile, esiste sempre una soluzione, anche se il tempo di esecuzione può essere molto grande per alcuni input.

- Consideriamo il problema di determinare se un dato DFA e una data espressione regolare sono equivalenti (cioè riconoscono lo stesso linguaggio).
 - a) Scrivete questo problema come linguaggio.
 - b) Dimostrare che questo problema è decidibile.
- a) Il problema di determinare se un dato DFA e una data espressione regolare sono equivalenti può essere scritto come il seguente linguaggio:

$\text{EQ_DFA_ER} = \{(A, E) \mid A \text{ è un DFA e } E \text{ è un'espressione regolare, e } A \text{ ed } E \text{ riconoscono lo stesso linguaggio}\}$

b) Il problema di determinare se un DFA e un'espressione regolare sono equivalenti è decidibile. Possiamo utilizzare l'algoritmo di equivalenza tra DFA e espressioni regolari per risolvere questo problema.

L'algoritmo di equivalenza tra DFA ed espressioni regolari funziona nel seguente modo:

1. Data un'espressione regolare E ed un DFA A , costruiamo un NFA B che riconosce la stessa lingua di E .
2. Costruiamo il DFA C equivalente a B utilizzando l'algoritmo di determinizzazione di sottoinsiemi.
3. Confrontiamo i DFA A e C per verificare se riconoscono lo stesso linguaggio.

Se i DFA A e C riconoscono lo stesso linguaggio, allora l'espressione regolare E e il DFA A sono equivalenti. In caso contrario, l'espressione regolare E ed il DFA A non sono equivalenti.

L'algoritmo di equivalenza tra DFA ed espressioni regolari è un algoritmo finito e deterministico che termina in un numero finito di passi. Quindi, il problema di determinare se un DFA e un'espressione regolare sono equivalenti è decidibile.

Inoltre, esiste un algoritmo più efficiente per risolvere questo problema, noto come l'algoritmo di Hopcroft-Karp. Questo algoritmo ha una complessità temporale di $O(n \log n)$, dove n è il numero di stati del DFA, ed è quindi più efficiente dell'algoritmo di equivalenza tra DFA ed espressioni regolari.

In conclusione, il problema di determinare se un DFA e un'espressione regolare sono equivalenti è un problema decidibile.

Indecidibili

- 21.** Sia $E_{TM} = \{\langle M \rangle \mid G \text{ è una TM tale che } L(M) = \emptyset\}$. Mostrare che $\overline{E_{TM}}$, il complemento di E_{TM} , è Turing-riconoscibile.

Per dimostrare che il complemento di ETM è Turing-riconoscibile, dobbiamo costruire una macchina di Turing che riconosce il complemento di ETM. In altre parole, dobbiamo costruire una macchina di Turing che accetta tutte le stringhe che non sono della forma $\langle M \rangle$, dove M è una TM e $L(M) = \emptyset$.

La costruzione della macchina di Turing può essere effettuata in modo simile alla costruzione della macchina di Turing per ETM. In particolare, la macchina di Turing può procedere come segue:

1. Leggere l'input w.
2. Se w non ha la forma $\langle M \rangle$ per qualche TM M, accettare w.
3. Se w ha la forma $\langle M \rangle$ per qualche TM M, simulare l'esecuzione di M sull'input vuoto.
4. Se la simulazione termina in uno stato di accettazione, rifiutare w.
5. Se la simulazione non termina o termina in uno stato di rifiuto, accettare w.

In altre parole, la macchina di Turing verifica se l'input è della forma $\langle M \rangle$, dove M è una TM. Se l'input non ha questa forma, la macchina di Turing accetta l'input. Se l'input ha questa forma, la macchina di Turing simula l'esecuzione di M sull'input vuoto. Se la simulazione termina in uno stato di accettazione, la macchina di Turing rifiuta l'input, perché abbiamo trovato una TM M che accetta il linguaggio e quindi l'input non appartiene al complemento di ETM. Se la simulazione non termina o termina in uno stato di rifiuto, la macchina di Turing accetta l'input, perché non abbiamo trovato una TM M che accetta il linguaggio e quindi l'input appartiene al complemento di ETM.

Questa macchina di Turing riconosce il complemento di ETM, perché accetta tutte le stringhe che non sono della forma $\langle M \rangle$, dove M è una TM e $L(M) = \emptyset$, e rifiuta tutte le stringhe che sono della forma $\langle M \rangle$, dove M è una TM e $L(M) \neq \emptyset$. Poiché il complemento di ETM è Turing-riconoscibile, segue che ETM deve essere Turing-riconoscibile, perché il complemento di un linguaggio Turing-riconoscibile è sempre Turing-riconoscibile.

- 22.** Mostrare che se A è Turing-riconoscibile e $A \leq_m \overline{A}$, allora A è decidibile.

Per dimostrare che se A è Turing-riconoscibile e $A \leq_m \overline{A}$, allora A è decidibile, possiamo utilizzare il seguente ragionamento:

1. Poiché A è Turing-riconoscibile, esiste una macchina di Turing M che accetta tutti e solo gli elementi di A.
2. Poiché $A \leq_m \overline{A}$, esiste una funzione calcolabile f tale che $x \in A \iff f(x) \in \overline{A}$.
3. Possiamo utilizzare f per costruire una nuova macchina di Turing N che decide A.
4. In particolare, la macchina di Turing N può essere costruita come segue:
 - Leggi l'input x.
 - Calcola f(x).
 - Utilizza la macchina di Turing M per decidere se f(x) appartiene ad A.
 - Se f(x) appartiene ad A, accetta x. Altrimenti, rifiuta x.

5. Poiché f è calcolabile e M è una macchina di Turing, la macchina di Turing N è ben definita e calcola la funzione caratteristica di A .

6. Pertanto, A è decidibile.

In sostanza, la dimostrazione si basa sull'idea che se esiste una riduzione mappante A su A , allora possiamo utilizzare questa riduzione per costruire una nuova macchina di Turing che decide A . In particolare, la riduzione ci consente di trasformare la decisione di A in una decisione su un altro linguaggio (in questo caso, A stessa), che è già Turing-riconoscibile. Utilizzando questa trasformazione, possiamo costruire una nuova macchina di Turing che decide A in modo efficiente.

Questa dimostrazione mostra l'importanza delle riduzioni mappanti nella teoria della computabilità. Le riduzioni mappanti sono strumenti utili per dimostrare la decidibilità o l'indecidibilità di un linguaggio. In particolare, se esiste una riduzione mappante tra due linguaggi, allora la decidibilità o l'indecidibilità di uno di questi linguaggi può essere dedotta dalla decidibilità o indecidibilità dell'altro.

23. Sia A un linguaggio. Dimostrare che A è Turing-riconoscibile *se e solo se* esiste un linguaggio decidibile B tale che $A = \{x \mid \text{esiste } y \text{ tale che } \langle x, y \rangle \in B\}$.

Per dimostrare che A è Turing-riconoscibile se e solo se esiste un linguaggio decidibile B tale che $A = \{x \mid \text{esiste } y \text{ tale che } \langle x, y \rangle \in B\}$, possiamo utilizzare il seguente ragionamento:

1. Supponiamo che A sia Turing-riconoscibile. Allora esiste una macchina di Turing M che accetta tutti e solo gli elementi di A .

2. Costruiamo il linguaggio B come segue:

- $B = \{\langle x, y \rangle \mid x \text{ appartiene ad } A \text{ e la macchina di Turing } M \text{ accetta } \langle x, y \rangle\}$.

3. Poiché M è una macchina di Turing, la verifica se $\langle x, y \rangle$ appartiene a B è decidibile in modo efficiente. Pertanto, B è un linguaggio decidibile.

4. Dimostriamo che $A = \{x \mid \text{esiste } y \text{ tale che } \langle x, y \rangle \in B\}$.

- Se x appartiene ad A , allora esiste una stringa y tale che $\langle x, y \rangle$ appartiene a B , perché la macchina di Turing M accetta $\langle x, y \rangle$.

- Se x non appartiene ad A , allora non esiste una stringa y tale che $\langle x, y \rangle$ appartiene a B , perché la macchina di Turing M non accetta $\langle x, y \rangle$.

- Pertanto, A è uguale all'insieme dei primi elementi di tutte le coppie $\langle x, y \rangle$ che appartengono a B . In altre parole, A è uguale all'insieme dei valori di x per cui esiste una coppia $\langle x, y \rangle$ che appartiene a B .

5. Quindi, abbiamo dimostrato che A è uguale all'insieme dei valori di x per cui esiste una coppia $\langle x, y \rangle$ che appartiene a un linguaggio decidibile B . Pertanto, A può essere espresso come l'insieme di tutti i valori di x per cui esiste un y tale che $\langle x, y \rangle$ appartiene a B .

6. Viceversa, supponiamo che esista un linguaggio decidibile B tale che $A = \{x \mid \text{esiste } y \text{ tale che } \langle x, y \rangle \in B\}$.

7. Costruiamo una macchina di Turing M che riconosce A come segue:

- Leggi l'input x .

- Per ogni stringa y , verifica se $\langle x, y \rangle$ appartiene a B .

- Se sì, accetta x .

- Se non hai ancora accettato x e hai esaminato tutte le stringhe y , rifiuta x .

8. Poiché B è decidibile, la verifica di ogni coppia $\langle x, y \rangle$ può essere eseguita in modo efficiente. Inoltre, se x appartiene ad A , allora esiste almeno una coppia $\langle x, y \rangle$ che appartiene a B , e quindi la macchina di Turing M accetta x . D'altra parte, se x non appartiene ad A , allora per ogni y , $\langle x, y \rangle$ non appartiene a B , e quindi la macchina di Turing M rifiuta x .

9. Pertanto, la macchina di Turing M riconosce A .

10. Quindi, abbiamo dimostrato che se esiste un linguaggio decidibile B tale che $A = \{x \mid \text{esiste } y \text{ tale che } \langle x, y \rangle \in B\}$, allora A è Turing-riconoscibile.

In sintesi, abbiamo dimostrato che A è Turing-riconoscibile se e solo se esiste un linguaggio decidibile B tale che $A = \{x \mid \text{esiste } y \text{ tale che } \langle x, y \rangle \in B\}$. Questo risultato mostra un'importante connessione tra i linguaggi decidibili e i linguaggi Turing-riconoscibili e suggerisce che i linguaggi Turing-riconoscibili possono essere espressi come l'insieme di tutte le coppie di stringhe che appartengono a un linguaggio decidibile.

24. $A \leq_m B$ e B è un linguaggio regolare implica che A è un linguaggio regolare? Perché sì o perché no?

La relazione di riducibilità mappante non preserva la regolarità dei linguaggi. Pertanto, non possiamo concludere che se $A \leq_m B$ e B è un linguaggio regolare, allora A è un linguaggio regolare.

Infatti, la riduzione mappante $A \leq_m B$ implica solo che esiste una funzione calcolabile f che trasforma ogni istanza di A in un'istanza di B in modo che x appartiene ad A se e solo se $f(x)$ appartiene a B . Tuttavia, questa riduzione non garantisce che la struttura del linguaggio A sia simile a quella del linguaggio B . In altre parole, anche se possiamo trasformare ogni istanza di A in un'istanza di B , questo non implica che A e B abbiano la stessa struttura o che A sia regolare se B è regolare.

Ad esempio, consideriamo il linguaggio $A = \{0^n 1^n \mid n \geq 0\}$ e il linguaggio $B = \{0^n \mid n \geq 0\}$. Il linguaggio B è un linguaggio regolare, in quanto può essere descritto dalla seguente espressione regolare: 0^* . Il linguaggio A può essere ridotto a B come segue: definiamo la funzione $f: A \rightarrow B$ come $f(0^n 1^n) = 0^n$. È facile verificare che f è calcolabile e che x appartiene ad A se e solo se $f(x)$ appartiene a B . Tuttavia, non possiamo concludere che A è un linguaggio regolare, perché il linguaggio A non può essere descritto da un'espressione regolare. Infatti, il linguaggio A è un classico esempio di un linguaggio non regolare.

In sintesi, la relazione di riducibilità mappante non preserva la regolarità dei linguaggi. Anche se possiamo ridurre un linguaggio non regolare ad un linguaggio regolare, questo non implica che il linguaggio originale sia regolare. Pertanto, non possiamo concludere che se $A \leq_m B$ e B è un linguaggio regolare, allora A è un linguaggio regolare.

26. Sia $J = \{w \mid w = 0x \text{ per qualche } x \in A_{TM} \text{ oppure } w = 1y \text{ per qualche } y \in \overline{A_{TM}}\}$. Mostrare che sia J che \overline{J} non sono Turing-riconoscibili.

Per dimostrare che il linguaggio J non è Turing-riconoscibile, possiamo utilizzare il metodo di diagonalizzazione di Cantor. Supponiamo per assurdo che J sia Turing-riconoscibile. Allora esiste una macchina di Turing M che riconosce J .

Costruiamo un nuovo linguaggio L come segue:

$$L = \{0^n \mid n \text{ non appartiene ad } A_{TM}\} \cup \{1^n \mid n \text{ appartiene ad } A_{TM}\}$$

In altre parole, il linguaggio L contiene tutte le stringhe di zeri il cui valore in notazione binaria non appartiene ad A_{TM} , insieme a tutte le stringhe di uni il cui valore in notazione binaria appartiene ad A_{TM} .

Ora, possiamo costruire una nuova macchina di Turing N che riconosce L utilizzando la macchina di Turing M come un filtro. La macchina di Turing N procede come segue:

1. Leggi l'input x .
2. Se x è della forma 0^n , dove n non appartiene ad ATM, accetta x .
3. Se x è della forma 1^n , dove n appartiene ad ATM, simula l'esecuzione della macchina di Turing M sull'input 0^n . Se M accetta 0^n , accetta x . Altrimenti, rifiuta x .
4. Se x non è della forma 0^n o 1^n , rifiuta x .

In altre parole, la macchina di Turing N accetta tutte le stringhe di zeri il cui valore in notazione binaria non appartiene ad ATM. Inoltre, se x è una stringa di uni il cui valore in notazione binaria appartiene ad ATM, la macchina di Turing N simula l'esecuzione della macchina di Turing M sull'input 0^n , dove n è il valore in notazione binaria di x , e accetta x se e solo se M accetta 0^n .

Ora, consideriamo il caso in cui applichiamo la macchina di Turing N sull'input $w = 1^n$, dove n non appartiene ad ATM. In questo caso, la macchina di Turing N accetta w , perché w è della forma 1^n e n non appartiene ad ATM. D'altra parte, se applichiamo la macchina di Turing N sull'input $w = 0^n$, dove n appartiene ad ATM, la macchina di Turing N simula l'esecuzione della macchina di Turing M sull'input 0^n . Poiché M riconosce J , la macchina di Turing M deve accettare 0^n se e solo se 0^n appartiene ad ATM. Ma questo significa che la macchina di Turing N accetta anche l'input $w = 0^n$, il che contraddice il fatto che w non appartiene a J .

Pertanto, abbiamo dimostrato che la macchina di Turing N non può riconoscere il linguaggio L , il che implica che il linguaggio L non è Turing-riconoscibile. Ma poiché J contiene L , questo significa che J non può essere Turing-riconoscibile.

In sintesi, abbiamo dimostrato che il linguaggio J non è Turing-riconoscibile utilizzando il metodo di diagonalizzazione di Cantor. L'idea di base della dimostrazione è quella di costruire un linguaggio che contiene informazioni sia su ATM che su un complemento di ATM e di mostrare che nessuna macchina di Turing può riconoscere questo linguaggio. Questo dimostra che il linguaggio J non può essere Turing-riconoscibile. Lo stesso ragionamento può essere applicato anche al linguaggio J per dimostrare che anche J non è Turing-riconoscibile.

- Il seguente linguaggio L è indecidibile?

$$L = \{M \mid M \text{ è una descrizione di una macchina di Turing ed esiste un ingresso } x \text{ di lunghezza } k \text{ tale che } M \text{ si ferma dopo al massimo } k \text{ passi}\}.$$

Il linguaggio L è indecidibile. Infatti, possiamo utilizzare il metodo di riduzione dell'halting per dimostrare che L è indecidibile.

Supponiamo per assurdo che esista una macchina di Turing H che decide L . Costruiamo una nuova macchina di Turing D che utilizza H per decidere il problema dell'halting, che sappiamo essere indecidibile. La macchina di Turing D procede come segue:

1. Dato in input una coppia $\langle M, x \rangle$, dove M è una descrizione di una macchina di Turing e x è una stringa di input, D costruisce una nuova macchina di Turing M' che simula l'esecuzione di M su x per al massimo k passi, dove k è la lunghezza di x .
2. Lascia che H decida se M' si ferma dopo al massimo k passi.
3. Se H accetta, allora M si ferma su x e la macchina di Turing D accetta $\langle M, x \rangle$. Altrimenti, se H rifiuta, allora M non si ferma su x e la macchina di Turing D rifiuta $\langle M, x \rangle$.

In altre parole, la macchina di Turing D simula l'esecuzione di M su x per al massimo k passi, costruendo una nuova macchina di Turing M' che limita l'esecuzione di M . Successivamente, la macchina di Turing D utilizza

H per decidere se M' si ferma dopo al massimo k passi. Se H accetta, allora la macchina di Turing D accetta $\langle M, x \rangle$, altrimenti rifiuta $\langle M, x \rangle$.

Ora, supponiamo che esista una macchina di Turing T che risolve il problema dell'halting, ovvero che T accetta $\langle M, x \rangle$ se e solo se M si ferma su x . Possiamo utilizzare la macchina di Turing T per costruire una nuova macchina di Turing H' che decide il linguaggio L . La macchina di Turing H' procede come segue:

1. Dato in input una descrizione di una macchina di Turing M , la macchina di Turing H' costruisce tutte le coppie $\langle M, x \rangle$ in cui x è una stringa di lunghezza k , dove k è la lunghezza della descrizione di M .
2. Per ogni coppia $\langle M, x \rangle$ costruita al passo precedente, la macchina di Turing H' utilizza T per decidere se M si ferma su x .
3. Se T accetta $\langle M, x \rangle$ per almeno una coppia $\langle M, x \rangle$, allora la macchina di Turing H' accetta la descrizione di M . Altrimenti, se T rifiuta per tutte le coppie $\langle M, x \rangle$, la macchina di Turing H' rifiuta la descrizione di M .

In altre parole, la macchina di Turing H' costruisce tutte le coppie $\langle M, x \rangle$ in cui x è una stringa di lunghezza k , e utilizza la macchina di Turing T per verificare se M si ferma su ogni coppia $\langle M, x \rangle$. Se M si ferma su almeno una coppia, allora la macchina di Turing H' accetta la descrizione di M , altrimenti la rifiuta.

Ma sappiamo che il problema dell'halting è indecidibile, il che significa che non esiste una macchina di Turing T che risolve il problema dell'halting. Pertanto, non possiamo costruire la macchina di Turing H' che decide L , il che implica che il linguaggio L è indecidibile.

In sintesi, abbiamo dimostrato che il linguaggio L è indecidibile utilizzando la riduzione dell'halting. L'idea di base della dimostrazione è quella di utilizzare il problema dell'halting, che sappiamo essere indecidibile, per dimostrare che il linguaggio L è indecidibile.

- Si consideri il linguaggio

$$A\varepsilon TM = \{ \langle M \rangle \mid M \text{ è un TM che accetta } \varepsilon \}.$$

Mostrare che $A\varepsilon TM$ è indecidibile.

Per dimostrare che il linguaggio $A\varepsilon TM$ è indecidibile, possiamo utilizzare il metodo di riduzione dell'halting.

Supponiamo per assurdo che esista una macchina di Turing H che decide $A\varepsilon TM$. Costruiamo una nuova macchina di Turing D che utilizza H per decidere il problema dell'halting, che sappiamo essere indecidibile. La macchina di Turing D procede come segue:

1. Dato in input una coppia $\langle M, x \rangle$, dove M è una descrizione di una macchina di Turing e x è una stringa di input, D costruisce una nuova macchina di Turing M' che simula l'esecuzione di M su x .
2. Lascia che H decida se M' accetta ε .
3. Se H accetta, allora M si ferma su x e la macchina di Turing D accetta $\langle M, x \rangle$. Altrimenti, se H rifiuta, allora M non si ferma su x e la macchina di Turing D rifiuta $\langle M, x \rangle$.

In altre parole, la macchina di Turing D costruisce una nuova macchina di Turing M' che simula l'esecuzione di M su x . Successivamente, la macchina di Turing D utilizza H per decidere se M' accetta ε . Se H accetta, allora la macchina di Turing D accetta $\langle M, x \rangle$, altrimenti rifiuta $\langle M, x \rangle$.

Ora, supponiamo che esista una macchina di Turing T che risolve il problema dell'halting, ovvero che T accetta $\langle M, x \rangle$ se e solo se M si ferma su x . Possiamo utilizzare la macchina di Turing T per costruire una nuova macchina di Turing H' che decide il linguaggio $A\varepsilon TM$. La macchina di Turing H' procede come segue:

1. Dato in input una descrizione di una macchina di Turing M, la macchina di Turing H' costruisce una nuova macchina di Turing M' che simula l'esecuzione di M su ϵ .

2. Utilizza la macchina di Turing T per decidere se M si ferma su ϵ .

3. Se T accetta, allora la macchina di Turing H' accetta la descrizione di M. Altrimenti, la macchina di Turing H' rifiuta la descrizione di M.

In altre parole, la macchina di Turing H' costruisce una nuova macchina di Turing M' che simula l'esecuzione di M su ϵ , e utilizza la macchina di Turing T per verificare se M si ferma su ϵ . Se M si ferma su ϵ , allora la macchina di Turing H' accetta la descrizione di M, altrimenti la rifiuta.

Ma sappiamo che il problema dell'halting è indecidibile, il che significa che non esiste una macchina di Turing T che risolve il problema dell'halting. Pertanto, non possiamo costruire la macchina di Turing H' che decide $A\epsilon$ TM, il che implica che il linguaggio $A\epsilon$ TM è indecidibile.

In sintesi, abbiamo dimostrato che il linguaggio $A\epsilon$ TM è indecidibile utilizzando la riduzione dell'halting. L'idea di base della dimostrazione è quella di utilizzare il problema dell'halting, che sappiamo essere indecidibile, per dimostrare che il linguaggio $A\epsilon$ TM è indecidibile.

- Consideriamo il problema della vacuità delle macchine di Turing:

$$ETM = \{ \langle M \rangle \mid M \text{ è una macchina di Turing con } L(M) = \emptyset \}.$$

Mostrare che ETM è co-Turing-riconoscibile. (Un linguaggio L è co-Turing-riconoscibile se il suo complemento L^c è Turing-riconoscibile). Si noti che il complemento di ETM è $ETM^c = \{ \langle M \rangle \mid M \text{ è una macchina di Turing con } L(M) \neq \emptyset \}$.

Per dimostrare che ETM è co-Turing-riconoscibile, dobbiamo dimostrare che il suo complemento ETM^c è Turing-riconoscibile. In altre parole, dobbiamo dimostrare che esiste una macchina di Turing che riconosce il linguaggio ETM^c .

Possiamo costruire una macchina di Turing M' che riconosce ETM^c come segue:

1. Dato in input una descrizione di una macchina di Turing M, la macchina di Turing M' simula l'esecuzione di M su tutte le stringhe di input di lunghezza crescente finché non trova una stringa w tale che M accetta w.

2. Se la macchina di Turing M' trova una stringa w tale che M accetta w, allora la macchina di Turing M' accetta la descrizione di M. Altrimenti, se la macchina di Turing M' esaurisce tutte le stringhe di input di lunghezza crescente e non trova una stringa che M accetta, allora la macchina di Turing M' rifiuta la descrizione di M.

In altre parole, la macchina di Turing M' simula l'esecuzione di M su tutte le stringhe di input di lunghezza crescente finché non trova una stringa w tale che M accetta w. Se M accetta una qualsiasi stringa di input, allora la macchina di Turing M' accetta la descrizione di M. Altrimenti, se M non accetta nessuna stringa di input, allora la macchina di Turing M' esaurisce tutte le stringhe di input di lunghezza crescente e rifiuta la descrizione di M.

Ora, dimostriamo che la macchina di Turing M' riconosce il complemento di ETM. Infatti, se la descrizione di una macchina di Turing M appartiene al linguaggio ETM^c , allora $L(M)$ è vuoto, il che significa che M non accetta alcuna stringa di input. In questo caso, la macchina di Turing M' esaurisce tutte le stringhe di input di lunghezza crescente e rifiuta la descrizione di M.

D'altra parte, se la descrizione di una macchina di Turing M non appartiene al linguaggio ETM^c , allora $L(M)$ non è vuoto, il che significa che esiste almeno una stringa di input w tale che M accetta w. In questo caso, la

macchina di Turing M' simula l'esecuzione di M su tutte le stringhe di input di lunghezza crescente e trova la stringa w che M accetta. Successivamente, la macchina di Turing M' accetta la descrizione di M .

In conclusione, abbiamo dimostrato che il complemento del linguaggio ETM è riconoscibile da una macchina di Turing, ovvero che ETM è co-Turing-riconoscibile. La macchina di Turing M' che abbiamo costruito riconosce il complemento di ETM, ovvero il linguaggio $\{ \langle M \rangle \mid M \text{ è una macchina di Turing con } L(M) \neq \emptyset \}$. In questo modo, abbiamo dimostrato che il linguaggio ETM è co-Turing-riconoscibile.

Soluzione Esercizi 06/06

Automi e Linguaggi Formali – Seconda Parte

Indecidibili, decidibili e varie

- Dimostrare che il seguente linguaggio è indecidibile:

$$L = \{\langle M, q, w \rangle \mid M \text{ è una macchina di Turing, } q \text{ è uno stato di } M \text{ e } M \text{ raggiunge } q \text{ nella sua computazione su } w\}.$$

Per dimostrare che il linguaggio $L = \{\langle M, q, w \rangle \mid M \text{ è una macchina di Turing, } q \text{ è uno stato di } M \text{ e } M \text{ raggiunge } q \text{ nella sua computazione su } w\}$ è indecidibile, possiamo utilizzare il metodo di riduzione. In particolare, dimostreremo che se esistesse un algoritmo che decidesse L , allora sarebbe possibile risolvere il problema dell'arresto, che è noto essere indecidibile.

Supponiamo per assurdo che esista un algoritmo A che decidesse L , cioè A prende in input $\langle M, q, w \rangle$ e restituisce "SI" se M raggiunge lo stato q durante la computazione su w , altrimenti restituisce "NO".

Ora costruiamo un nuovo algoritmo B per risolvere il problema dell'arresto utilizzando A come sottoprocedura:

B prende in input $\langle M, w \rangle$ (dove M è una macchina di Turing e w è una stringa)

1. Per ogni stato q di M :

- Esegui A sull'input $\langle M, q, w \rangle$.
- Se A restituisce "SI", allora B restituisce "SI" e termina.

2. Se il ciclo termina senza restituire "SI" per nessuno stato q di M , allora B restituisce "NO".

Ora analizziamo il comportamento di B . Se B restituisce "SI", significa che M raggiunge uno stato di arresto durante la sua computazione su w . Se B restituisce "NO", significa che M non raggiunge uno stato di arresto durante la sua computazione su w .

Quindi, se assumiamo che A decidesse L , allora B risolverebbe correttamente il problema dell'arresto. Ma sappiamo che il problema dell'arresto è indecidibile, quindi abbiamo raggiunto una contraddizione. Pertanto, la nostra ipotesi iniziale che A decidesse L deve essere falsa.

Di conseguenza, il linguaggio $L = \{\langle M, q, w \rangle \mid M \text{ è una macchina di Turing, } q \text{ è uno stato di } M \text{ e } M \text{ raggiunge } q \text{ nella sua computazione su } w\}$ è indecidibile.

- Dimostrare che il seguente linguaggio è indecidibile:

$$L = \{\langle M_1, M_2 \rangle \mid M_1 \text{ e } M_2 \text{ sono macchine di Turing e } L(M_1) \subseteq L(M_2)\}.$$

Per dimostrare che il linguaggio $L = \{\langle M_1, M_2 \rangle \mid M_1 \text{ e } M_2 \text{ sono macchine di Turing e } L(M_1) \subseteq L(M_2)\}$ è indecidibile, possiamo utilizzare il metodo di riduzione dal problema dell'arresto, che è noto essere indecidibile.

Supponiamo per assurdo che esista un algoritmo A che decidesse L , cioè A prende in input $\langle M_1, M_2 \rangle$ e restituisce "SI" se $L(M_1) \subseteq L(M_2)$, altrimenti restituisce "NO".

Ora costruiamo un nuovo algoritmo B per risolvere il problema dell'arresto utilizzando A come sottoprocedura:

B prende in input $\langle M, w \rangle$ (dove M è una macchina di Turing e w è una stringa)

1. Costruisci una nuova macchina di Turing M1 che fa quanto segue:

- Simula la computazione di M su w.
- Se M raggiunge uno stato di arresto, M1 accetta.
- Altrimenti, M1 entra in un loop infinito.

2. Esegui A sull'input $\langle M_1, M \rangle$.

3. Se A restituisce "SI", allora B restituisce "SI" e termina.

4. Se A restituisce "NO", allora B restituisce "NO" e termina.

Ora analizziamo il comportamento di B. Se B restituisce "SI", significa che $L(M_1) \subseteq L(M)$ e quindi M raggiunge uno stato di arresto durante la sua computazione su w. Se B restituisce "NO", significa che $L(M_1)$ non è un sottoinsieme di $L(M)$ e quindi M non raggiunge uno stato di arresto durante la sua computazione su w.

Quindi, se assumiamo che A decidesse L, allora B risolverebbe correttamente il problema dell'arresto. Ma sappiamo che il problema dell'arresto è indecidibile, quindi abbiamo raggiunto una contraddizione.

Pertanto, la nostra ipotesi iniziale che A decidesse L deve essere falsa.

Di conseguenza, il linguaggio $L = \{\langle M_1, M_2 \rangle \mid M_1 \text{ e } M_2 \text{ sono macchine di Turing e } L(M_1) \subseteq L(M_2)\}$ è indecidibile.

- Mostrare che esiste un linguaggio indecidibile contenuto in 1^*

Per dimostrare che esiste un linguaggio indecidibile contenuto in 1^* , possiamo utilizzare l'insieme di halting [Fonte 0]. In particolare, possiamo definire il linguaggio $L = \{1^k \mid = \langle i, j \rangle \wedge (i, j) \in H\}$, dove H è l'insieme di halting.

Il problema dell'halting è indecidibile, così come $\$H\$$, quindi ne consegue che anche $\$L\$ \subseteq 1^*$ è indecidibile. Questo perché $\$L\$$ è un sottoinsieme di 1^* , e se possiamo decidere se $\$L\$$ è riconosciuto da una macchina di Turing, possiamo anche decidere se $\$H\$$ è riconosciuto da una macchina di Turing, il che è impossibile.

Vale la pena notare che ci sono molti linguaggi indecidibili contenuti in 1^* , e che ci sono un numero incalcolabile di linguaggi sull'alfabeto $\{1\}$. Ciò significa che non esiste un algoritmo generale per risolvere il problema della decisione per tutti i linguaggi possibili.

Un altro modo di pensare ai linguaggi indecidibili è quello di utilizzare la diagonalizzazione. Per esempio, possiamo definire un linguaggio specifico L come $L = \{s_i \in \Sigma^* \mid s_i \in L_i\}$ dove L_i è l'i-esimo linguaggio sull'alfabeto Σ . Se $L = L_k$ per qualche $k \geq 1$, o $s_k \in L \setminus L$ entrambi i casi portano a una contraddizione. Pertanto, non è possibile elencare i linguaggi su Σ e quindi l'insieme dei linguaggi su Σ non è conteggiabile. Questo metodo di prova è chiamato diagonalizzazione.

È importante notare che la relazione di contenimento tra i linguaggi non preserva la durezza, il che significa che i linguaggi più piccoli non sono necessariamente più difficili di quelli più grandi. Pertanto, non possiamo assumere che un linguaggio sia decidibile solo perché è un sottoinsieme di un linguaggio decidibile. Ad esempio, l'insieme vuoto è un sottoinsieme di ogni linguaggio, ma è anche decidibile.

Per riassumere, ci sono molti linguaggi indecidibili contenuti in 1^* , e un modo per dimostrarlo è usare l'insieme di halting. Possiamo anche usare la diagonalizzazione per dimostrare l'esistenza di linguaggi indecidibili e non possiamo assumere che un linguaggio sia decidibile solo perché è un sottoinsieme di un linguaggio decidibile.

- Sia $B = \{\langle M_1 \rangle, \langle M_2 \rangle, \dots\}$ un linguaggio riconoscibile da Turing costituito da descrizioni TM descrizioni. Dimostrare che esiste un linguaggio decidibile C costituito da descrizioni di TM tali che ogni macchina descritta in B ha una macchina equivalente in C e viceversa

Per dimostrare che esiste un linguaggio decidibile C costituito da descrizioni di TM tali che ogni macchina descritta in B ha una macchina equivalente in C e viceversa, possiamo costruire tale linguaggio C come segue:

1. Per una data descrizione di macchina di Turing $\langle M \rangle$ in B , possiamo simulare M su tutti i possibili ingressi entro un limite di tempo finito. Se M si ferma e accetta su qualsiasi ingresso, aggiungiamo la descrizione $\langle M \rangle$ a C .
2. Contemporaneamente, possiamo simulare tutte le macchine di Turing in C su tutti i possibili ingressi entro un limite di tempo finito. Se una macchina in C si arresta e accetta su un qualsiasi ingresso, rimuovere la sua descrizione da C .
3. Ripetere i passi 1 e 2 all'infinito, iterando su tutte le possibili macchine di Turing in B e C .

Il linguaggio C consisterà in descrizioni di macchine di Turing che si arrestano e accettano su qualche input entro un limite di tempo finito. È decidibile perché il processo di simulazione termina per ogni macchina entro un limite di tempo finito.

Dimostriamo ora che ogni macchina descritta in B ha una macchina equivalente in C e viceversa:

1. Se una macchina di Turing M è descritta in B , significa che M è riconoscibile da Turing. Ciò implica che M si ferma e accetta su alcuni input. Per costruzione, aggiungiamo la descrizione $\langle M \rangle$ a C quando M si ferma e accetta su qualsiasi ingresso.
2. Se una macchina di Turing M è descritta in C , significa che M si arresta e accetta su qualche ingresso entro un limite di tempo finito. Per costruzione, eliminiamo la descrizione di M da C quando si arresta e accetta su qualsiasi ingresso.

Pertanto, ogni macchina descritta in B ha una macchina equivalente in C , poiché le macchine descritte in B che si fermano e accettano su un qualsiasi ingresso sono aggiunte a C , e ogni macchina descritta in C si ferma e accetta su qualche ingresso.

Allo stesso modo, viceversa, ogni macchina descritta in C ha una macchina equivalente in B , poiché le macchine descritte in C si fermano e accettano su qualche ingresso e le macchine che si fermano e accettano su qualsiasi ingresso vengono aggiunte a C .

Quindi, il linguaggio C costituito da descrizioni di TM può essere costruito in modo da soddisfare le condizioni date.

- Supponiamo di avere due funzioni F e G e di essere interessati a determinare se

$$F(x) = \int G(x) dx$$

Supponiamo che le mie funzioni siano composte da funzioni elementari (polinomi, esponenziali, log e funzioni trigonometriche), ma non, ad esempio, da serie di Taylor. Questo problema è decidibile?

Determinare se una data equazione integrale $F(x) = \int G(x) dx$ vale per funzioni elementari arbitrarie F e G è generalmente indecidibile. Ciò significa che non esiste un algoritmo che possa sempre determinare se l'equazione è vera o falsa per tutte le possibili scelte di F e G .

L'indecidibilità di questo problema deriva dal fatto che l'integrazione di funzioni elementari può portare a un'ampia gamma di funzioni diverse, comprese quelle che non possono essere espresse in termini di funzioni elementari. Esiste infatti un risultato ben noto, chiamato algoritmo di Risch, che fornisce una

procedura decisionale per determinare se una data funzione elementare ha un'antiderivata elementare. Tuttavia, anche con questo algoritmo, non è possibile determinare se due funzioni elementari date sono uguali o meno in generale.

Per illustrare questo punto, si consideri il seguente esempio:

Sia $F(x) = e^{x^2}$ e $G(x) = 2xe^{x^2}$.

Sebbene $G(x)$ sia la derivata di $F(x)$, non è possibile esprimere $F(x)$ in termini di funzioni elementari. Pertanto, l'equazione $F(x) = \int G(x)dx$ non è valida in termini di funzioni elementari. Tuttavia, senza conoscere le funzioni specifiche F e G , è generalmente impossibile determinare la validità dell'equazione.

In sintesi, determinare se $F(x) = \int G(x)dx$ vale per funzioni elementari arbitrarie F e G è un problema indecidibile a causa della complessità dell'integrazione e dell'ampia gamma di possibili funzioni che ne possono derivare.

- Si consideri il seguente problema, noto come Polynomial Halting Problem (PHP): Dato un polinomio con coefficienti interi e un ingresso intero, determinare se il polinomio si arresta (cioè raggiunge lo zero) quando viene valutato con l'ingresso dato. Il problema dell'Halting polinomiale è decidibile o indecidibile? Dimostrate la vostra risposta.

Il Polynomial Halting Problem è indecidibile, cioè non esiste un algoritmo che possa sempre determinare correttamente se un dato polinomio si ferma per un dato input.

Per dimostrare l'indecidibilità del PHP, lo ridurremo al problema di Halting, che è un noto problema indecidibile. Questa riduzione mostrerà che se disponessimo di un algoritmo per risolvere il PHP, potremmo usarlo per risolvere il Problema di Halting, il che contraddice la sua indecidibilità.

Supponiamo di avere un algoritmo A che risolve il PHP. Data una macchina di Turing M e una stringa di input w, vogliamo determinare se M si ferma sull'input w (cioè risolvere il Problema di Halting). Costruiremo un polinomio P con coefficienti interi tale che P si arresta se e solo se M si arresta su w.

Innanzitutto, codifichiamo il calcolo di M su w in un polinomio. Possiamo rappresentare la configurazione di M a ogni passo come una tupla (q, i, t) , dove q è lo stato corrente, i è la posizione corrente sul nastro e t è il contenuto corrente del nastro. Codifichiamo ogni tupla come un numero intero unico utilizzando la funzione di accoppiamento di Cantor.

Successivamente, costruiamo un polinomio che simuli l'esecuzione di M sull'input w. Inizializziamo una variabile x, che rappresenta il numero di passi che M ha eseguito finora. A ogni passo, valutiamo il polinomio per verificare la configurazione corrente e aggiornarla di conseguenza. Continuiamo questo processo fino a quando M si ferma o x raggiunge un certo valore (ad esempio, 2^n , dove n è la lunghezza della codifica di M e w).

Infine, definiamo il polinomio $P(x)$ come il polinomio ottenuto dalla codifica della configurazione finale di M. $P(x)$ si arresta se e solo se M si arresta sull'input w.

Supponiamo ora di avere un algoritmo A in grado di risolvere il PHP. Dato il polinomio P e l'ingresso x, possiamo usare A per determinare se P si arresta. Se A determina che P si arresta, significa che M si arresta sull'input w. Se A determina che P non si arresta, significa che M non si arresta sull'input w. Pertanto, abbiamo ridotto il Problema di Halting al PHP, dimostrando che il PHP è indecidibile.

Poiché il Problema di Halting è indecidibile e il PHP può essere ridotto ad esso, concludiamo che anche il Problema di Halting Polinomiale è indecidibile.

- Dimostrare che l'equivalenza di due query di aggregazione di MongoDB (strumento di base di dati non relazionale) è indecidibile riducendola all'Halting Problem, noto problema indecidibile.
- Consideriamo il seguente problema, noto come MongoDB Query Equivalence Problem (MQEP): Date due query di aggregazione di MongoDB, determinare se producono lo stesso risultato su una data collezione. Dimostrare che il MQEP è indecidibile fornendo una riduzione dall'Halting Problem.

Per dimostrare l'indecidibilità del MQEP, lo ridurremo al problema di Halting. Questa riduzione dimostrerà che se avessimo un algoritmo per decidere il MQEP, potremmo usarlo per decidere il problema di Halting, il che contraddice la sua indecidibilità.

Supponiamo di avere un algoritmo A in grado di decidere il MQEP. Data una macchina di Turing M e una stringa di input w, vogliamo determinare se M si ferma sull'input w (cioè risolvere il Problema di Halting). Costruiremo due query di aggregazione MongoDB, Q1 e Q2, tali che Q1 e Q2 producano lo stesso risultato se e solo se M si ferma sull'input w.

Costruzione delle query Q1 e Q2 di MongoDB:

Query Q1:

Costruire una pipeline di aggregazione MongoDB che includa le fasi necessarie per simulare l'esecuzione di M sull'input w.

Codificare ogni fase di calcolo della macchina di Turing come uno stadio di aggregazione corrispondente.

Includere ulteriori stadi per tracciare lo stato finale della macchina di Turing e produrre un risultato basato sul suo comportamento di arresto.

Query Q2:

Costruire una pipeline di aggregazione MongoDB che includa gli stadi necessari per simulare l'esecuzione di M sull'input w.

Codificare ogni fase di calcolo della macchina di Turing nell'ordine inverso rispetto a Q1.

Includere ulteriori fasi per tracciare lo stato finale della macchina di Turing e produrre un risultato basato sul suo comportamento di arresto.

Ora, se M si arresta sull'input w, sia Q1 che Q2 produrranno lo stesso risultato perché la simulazione di M seguirà gli stessi passi in entrambe le query. Al contrario, se M non si arresta sull'ingresso w, Q1 e Q2 produrranno risultati diversi perché la simulazione divergerà a un certo punto.

Supponiamo di avere un algoritmo A in grado di decidere il MQEP. Dati Q1 e Q2, possiamo usare A per determinare se sono equivalenti. Se A determina che Q1 e Q2 producono lo stesso risultato, significa che M si arresta sull'ingresso w. Se A determina che Q1 e Q2 producono risultati diversi, significa che M non si arresta sull'ingresso w. Pertanto, abbiamo ridotto l'Halting Problem al MQEP, dimostrando che il MQEP è indecidibile.

Poiché l'Halting Problem è indecidibile e il MQEP può essere ridotto ad esso, concludiamo che anche il MongoDB Query Equivalence Problem è indecidibile.

Macchine di Turing ed annesse varianti

- Una macchina di Turing con oracolo è una variante in cui la macchina di Turing ha accesso a un nastro "oracolo" aggiuntivo che può fornire risposte a domande specifiche. Dimostrare che le macchine di Turing con un oracolo riconoscono la classe dei linguaggi riconoscibili da Turing.

Per dimostrare che le macchine di Turing con un oracolo riconoscono la classe dei linguaggi riconoscibili da Turing, dobbiamo mostrare che, data una macchina di Turing con un oracolo, possiamo costruire una macchina di Turing equivalente che riconosce lo stesso linguaggio.

Supponiamo di avere una macchina di Turing M_O con un oracolo che può fornire risposte a domande specifiche. Vogliamo costruire una macchina di Turing M che riconosce lo stesso linguaggio di M_O .

La macchina di Turing M può essere costruita come segue:

1. M simula passo per passo l'esecuzione di M_O su un input specifico.
2. Quando M_O richiede una risposta dall'oracolo, M consulta l'oracolo per ottenere la risposta corrispondente.
3. M continua a simulare l'esecuzione di M_O fino a quando M_O termina l'esecuzione o entra in uno stato di accettazione.
4. Se M_O entra in uno stato di accettazione, M accetta l'input; altrimenti, M lo rifiuta.

La macchina di Turing M simula l'esecuzione di M_O , utilizzando l'oracolo quando necessario per ottenere risposte alle domande specifiche. Poiché M_O è una macchina di Turing con un oracolo, l'oracolo fornisce le risposte corrette, consentendo a M di riconoscere lo stesso linguaggio di M_O .

Di conseguenza, abbiamo dimostrato che le macchine di Turing con un oracolo riconoscono la classe dei linguaggi riconoscibili da Turing, poiché possiamo costruire una macchina di Turing equivalente che riconosce lo stesso linguaggio di una data macchina di Turing con un oracolo.

- Una macchina di Turing probabilistica è una variante in cui la funzione di transizione ha probabilità associate a ogni possibile transizione. La macchina sceglie casualmente una transizione in base alle probabilità a ogni passo. Dimostrare che le macchine di Turing probabilistiche riconoscono un sottoinsieme della classe dei linguaggi riconoscibili da Turing.

Per dimostrare che le macchine di Turing probabilistiche riconoscono un sottoinsieme della classe dei linguaggi riconoscibili da Turing, dobbiamo dimostrare che esiste almeno un linguaggio riconoscibile da una macchina di Turing tradizionale che non può essere riconosciuto da una macchina di Turing probabilistica.

Supponiamo che esista una macchina di Turing probabilistica P che riconosce tutti i linguaggi riconoscibili da una macchina di Turing tradizionale. Consideriamo il seguente linguaggio:

$$L = \{<M, w> \mid \text{la macchina di Turing } M \text{ accetta l'input } w \text{ in al massimo } n \text{ passi}\}$$

Dove n è un numero fisso. Il linguaggio L rappresenta tutti gli input w che possono essere accettati dalla macchina di Turing M in al massimo n passi.

Costruiamo una macchina di Turing tradizionale T che riconosce il linguaggio L . La macchina di Turing T simula passo per passo l'esecuzione di M su w , contando il numero di passi eseguiti. Se M accetta w entro n passi, allora T accetta l'input.

Ora consideriamo la macchina di Turing probabilistica P . Poiché P riconosce tutti i linguaggi riconoscibili da una macchina di Turing tradizionale, dovrebbe riconoscere anche il linguaggio L . Tuttavia, non esiste un

algoritmo probabilistico che possa contare il numero di passi eseguiti da una macchina di Turing in modo deterministico e corretto. Pertanto, non esiste una macchina di Turing probabilistica che possa riconoscere il linguaggio L.

Di conseguenza, abbiamo dimostrato che le macchine di Turing probabilistiche riconoscono un sottoinsieme della classe dei linguaggi riconoscibili da Turing, poiché esiste almeno un linguaggio riconoscibile da una macchina di Turing tradizionale che non può essere riconosciuto da una macchina di Turing probabilistica.

- Una macchina di Turing alternata è una variante che può passare tra due modalità di calcolo: una modalità esistenziale e una modalità universale. Nella modalità esistenziale, la macchina cerca di trovare un percorso di calcolo accettabile, mentre nella modalità universale verifica se tutti i percorsi di calcolo portano all'accettazione. Dimostrare che le macchine di Turing alternate riconoscono la classe dei linguaggi riconoscibili da Turing.

Per dimostrare che le macchine di Turing alternate riconoscono la classe dei linguaggi riconoscibili da Turing, dobbiamo dimostrare che per ogni linguaggio riconoscibile da Turing esiste una macchina di Turing alternata equivalente che lo riconosce.

Sia L un linguaggio riconoscibile da Turing. Ciò significa che esiste una macchina di Turing M che accetta ogni stringa in L e rifiuta o fa un loop sulle stringhe non presenti in L. Costruiremo una macchina di Turing alternata equivalente A che riconosce lo stesso linguaggio.

La macchina di Turing alternata A funziona come segue:

- A inizia in modalità esistenziale, cercando un percorso di calcolo accettabile.
- In modalità esistenziale, A sceglie in modo non deterministico una possibile transizione e simula M su tutti i rami possibili.
- Se uno qualsiasi dei rami raggiunge uno stato accettante, A accetta l'input.
- Se tutti i rami raggiungono uno stato di rifiuto o di loop, A passa alla modalità universale.
- Nella modalità universale, A simula M su tutti i rami possibili simultaneamente, verificando se ogni ramo alla fine raggiunge uno stato di accettazione.
- Se un ramo raggiunge uno stato di rifiuto o di loop, A rifiuta l'input. Altrimenti, se tutti i rami raggiungono uno stato di accettazione, A accetta l'input.

Per costruzione, la macchina di Turing alternata A riconosce il linguaggio L. Se esiste almeno un percorso di calcolo accettante per un input, A lo troverà nel modo esistenziale e accetterà l'input. Se tutti i percorsi di calcolo portano all'accettazione, A lo verificherà in modalità universale e accetterà l'input. Altrimenti, se non esiste un percorso di calcolo accettabile, A rifiuterà l'input.

Poiché ogni linguaggio riconoscibile da Turing può essere riconosciuto da una macchina di Turing alternata equivalente, abbiamo dimostrato che le macchine di Turing alternate riconoscono la classe dei linguaggi riconoscibili da Turing.

Pertanto, possiamo concludere che le macchine di Turing alternate sono sufficientemente potenti per riconoscere i linguaggi riconoscibili da Turing.

- Consideriamo una variante della macchina di Turing con un nastro infinito bidimensionale, in cui la macchina può muoversi in otto direzioni possibili: Nord, Nord-Est, Est, Sud-Est, Sud, Sud-Ovest, Ovest e Nord-Ovest. Progettare una macchina di Turing con questa variante che riconosca il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$.

Per progettare una macchina di Turing con un nastro infinito bidimensionale che riconosca il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$, possiamo utilizzare il seguente approccio:

- Si inizia con la scansione della stringa di input da sinistra a destra, contrassegnando ogni simbolo man mano che lo si incontra.
- Una volta raggiunta la fine della stringa di input, si sposta la testina del nastro all'inizio, segnando ogni simbolo incontrato.
- Mentre si torna all'inizio, si confronta ogni simbolo marcato con il simbolo corrispondente sul nastro di ingresso.
- Se in qualsiasi punto si riscontra una mancata corrispondenza o se il nastro di input è più lungo di quello marcato, si rifiuta l'input.
- Se l'intero nastro viene percorso senza errori di corrispondenza e il nastro di input è esaurito, l'input viene accettato.

Ecco una descrizione più dettagliata della macchina di Turing:

Stati:

- q_0 : Stato iniziale, inizia la scansione dell'input da sinistra a destra.
- q_1 : Esegue la scansione del nastro di ingresso contrassegnando ogni simbolo.
- q_2 : Spostare la testina del nastro all'inizio mentre si segna ogni simbolo.
- q_3 : Confronto di ciascun simbolo marcato con il simbolo corrispondente sul nastro di ingresso.
- q_4 : Stato Accept, l'ingresso è accettato.
- q_{Rifiuto} : Stato di rifiuto, l'input viene rifiutato.

Transizioni:

Dallo stato q_0 :

Se il simbolo corrente è '0' o '1', contrassegnare il simbolo e spostarsi a destra. Transizione allo stato q_1 .

Se il simbolo corrente è vuoto, spostarsi a sinistra e passare allo stato q_3 .

Dallo stato q_1 :

Se il simbolo corrente è '0' o '1', spostarsi a destra. Transizione allo stato q_1 .

Se il simbolo corrente è vuoto, spostarsi a sinistra e passare allo stato q_2 .

Dallo stato q_2 :

Se il simbolo corrente è '0' o '1', contrassegnare il simbolo e spostarsi a sinistra. Transizione allo stato q_2 .

Se il simbolo corrente è vuoto, spostarsi a destra e passare allo stato q_3 .

Dallo stato q_3 :

Se il simbolo corrente sul nastro marcato è uguale al simbolo corrente sul nastro di ingresso, ci si sposta a sinistra su entrambi i nastri. Transizione allo stato q_3 .

Se il simbolo corrente sul nastro contrassegnato non è uguale al simbolo corrente sul nastro di ingresso, si passa allo stato q_{reject} .

Dallo stato qreject:

Rifiuta l'input arrestando la macchina.

Dallo stato q4:

Accetta l'input arrestando la macchina.

Questa variante della macchina di Turing con un nastro infinito bidimensionale e otto possibili direzioni può riconoscere il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$ verificando la presenza di una copia speculare della stringa di ingresso sul nastro.

Nota: L'esatta funzione di transizione e i dettagli dell'implementazione della macchina di Turing possono variare a seconda della rappresentazione specifica e delle regole del nastro bidimensionale.

- Considerare una variante della macchina di Turing con un nastro quantistico, dove la macchina può eseguire operazioni quantistiche come la sovrapposizione e l'entanglement sui simboli del nastro.

In una macchina di Turing quantistica (QTM), i simboli del nastro possono esistere in una sovrapposizione di stati, ovvero possono rappresentare contemporaneamente più valori. Questo è simile al concetto di sovrapposizione della meccanica quantistica, in cui le particelle possono esistere in più stati finché non vengono osservate.

Inoltre, il QTM può creare un entanglement tra i simboli sul nastro, il che significa che lo stato di un simbolo diventa dipendente dallo stato di un altro simbolo. Questo entanglement consente una forma di correlazione o condivisione di informazioni tra i simboli.

Progettare una macchina di Turing quantistica che riconosca il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$.

Per progettare un QTM che riconosca il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$, possiamo utilizzare il seguente approccio:

1. Iniziare con una stringa di input w sul nastro quantistico. Ogni simbolo di w è inizialmente in una sovrapposizione dei suoi possibili valori (0 o 1).
2. Applicare operazioni per agganciare le coppie di simboli corrispondenti sul nastro. L'entanglement assicura che le due copie di w siano sempre nello stesso stato.
3. Eseguire una misura sul nastro per far collassare la sovrapposizione e ottenere un valore definito per ogni simbolo.
4. Confrontare i simboli colllassati di una copia di w con i simboli corrispondenti dell'altra copia.
5. Se in qualsiasi punto si riscontra una mancata corrispondenza o se la lunghezza del nastro di ingresso non è uniforme, rifiutare l'ingresso.
6. Se l'intero nastro viene percorso senza errori di corrispondenza e il nastro di ingresso è di lunghezza pari, accettare l'input.

Ecco una descrizione più dettagliata delle fasi del QTM:

Stati:

- q0: Stato iniziale, inizio delle operazioni quantistiche.

- q1: Aggrovigliare coppie di simboli corrispondenti sul nastro.
- q2: Eseguire una misura sul nastro.
- q3: Confronto tra i simboli collassati di una copia e i simboli corrispondenti dell'altra copia.
- q4: Stato di accettazione, l'input è accettato.
- qRifiuto: Stato di rifiuto, l'input viene rifiutato.

Transizioni:

- Dallo stato q0:
 - Applica le operazioni di entanglement sulle coppie di simboli corrispondenti. Transizione allo stato q1.
- Dallo stato q1:
 - Continuare ad applicare le operazioni di entanglement finché tutte le coppie di simboli sono entanglemate. Transizione allo stato q2.
- Dallo stato q2:
 - Eseguire una misura sui simboli del nastro, facendo collassare la sovrapposizione. Transizione allo stato q3.
- Dallo stato q3:
 - Confronto di ogni simbolo collassato di una copia con il simbolo corrispondente dell'altra copia.
 - Se si riscontra una mancata corrispondenza o se la lunghezza del nastro di ingresso è dispari, si passa allo stato qreject.
 - Se tutti i simboli corrispondono e la lunghezza del nastro di ingresso è pari, si passa allo stato q4.
- Dallo stato qreject:
 - Rifiuta l'ingresso arrestando la macchina.
- Dallo stato q4:
 - Accetta l'input arrestando la macchina.

Questa spiegazione semplificata di una macchina di Turing quantistica con un nastro quantistico dimostra come la superposizione e l'entanglement possano essere utilizzati per riconoscere il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$. I dettagli esatti dell'implementazione e le operazioni quantistiche possono variare a seconda dello specifico framework o modello di calcolo quantistico utilizzato.

- Una variante della macchina di Turing con un nastro per il viaggio nel tempo permette alla macchina di viaggiare indietro e avanti nel tempo sul nastro. Progettare una macchina di Turing a viaggio nel tempo che riconosca il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$. La macchina deve essere in grado di gestire le potenziali violazioni della causalità che possono derivare dalle operazioni di viaggio nel tempo.

La progettazione di una macchina di Turing per il viaggio nel tempo che riconosca il linguaggio $L = \{ww \mid w \text{ è una stringa binaria}\}$ gestendo al contempo le potenziali violazioni della causalità richiede un'attenta considerazione. Ecco un possibile approccio per costruire una macchina di questo tipo:

Stati:

- q0: Stato iniziale, inizio delle operazioni di viaggio nel tempo.
- q1: Attraversare il nastro a destra per individuare la fine della stringa di input w.
- q2: Spostare la testina del nastro all'inizio di w.
- q3: Confronto dei simboli delle estremità sinistra e destra di w.
- q4: Stato Accept, l'input è accettato.
- qRifiuto: Stato Reject, l'input viene rifiutato.

Transizioni:

- Dallo stato q0:
 - Scansione del nastro per individuare la fine della stringa di input w. Transizione allo stato q1.
- Dallo stato q1:
 - Spostare la testina del nastro verso destra fino a raggiungere la fine di w. Transizione allo stato q2.
- Dallo stato q2:
 - Muove la testina del nastro verso sinistra, tornando all'inizio di w. Transizione allo stato q3.
- Dallo stato q3:
 - Confronto tra il simbolo sotto la testina del nastro e il simbolo corrispondente alla fine di w.
 - Se si riscontra una mancata corrispondenza, si passa allo stato qreject.
 - Se tutti i simboli corrispondono fino al punto medio di w, si passa allo stato q4.
- Dallo stato qreject:
 - Rifiuta l'input arrestando la macchina.
- Dallo stato q4:
 - Accetta l'input arrestando la macchina.

Le operazioni di viaggio nel tempo in questa macchina comportano lo spostamento della testina del nastro avanti e indietro, consentendo alla macchina di attraversare il nastro in entrambe le direzioni, violando potenzialmente i vincoli di causalità. La macchina assicura che i simboli delle estremità sinistra e destra di w siano confrontati con precisione manipolando opportunamente il movimento della testina del nastro.

È importante notare che il concetto di viaggio nel tempo introduce complesse implicazioni teoriche e filosofiche, e la fattibilità di implementare una macchina di questo tipo va oltre le attuali capacità dell'informatica classica. Questo esercizio esplora uno scenario ipotetico per sfidare la nostra comprensione delle macchine di Turing e delle loro varianti.

Soluzione Esercizi 07/06

Automi e Linguaggi Formali – Seconda Parte

Indecidibili, Decidibili & Company

- Dimostrare che $\{M \mid M \text{ è una macchina che gira in } 100n^3 + 300 \text{ passi}\}$ è indecidibile

Per dimostrare che il linguaggio $L = \{M \mid M \text{ è una macchina che funziona in } 100n^3 + 300 \text{ tempo}\}$ è indecidibile, possiamo ridurlo al problema di Halting, che è un noto problema indecidibile. Questa riduzione mostrerà che se disponessimo di un algoritmo per decidere L , potremmo usarlo per decidere il problema di Halting, il che contraddice la sua indecidibilità.

Supponiamo di avere un decisore D per il linguaggio L . Data una macchina di Turing M e una stringa di input w , vogliamo determinare se M si ferma sull'input w (cioè risolvere il Problema di Halting). Costruiremo una macchina di Turing M' che decide L , utilizzando D come subroutine.

Costruzione della macchina di Turing M' :

1. Su input x , dove x è una stringa che codifica una macchina di Turing:

2. Costruire una macchina di Turing M_x come segue:

- Per ogni input di lunghezza n , simulare M sull'input w per $100n^3 + 300$ passi.
- Se M si ferma entro questo tempo, accettare; altrimenti, rifiutare.

3. Eseguire il decider D sull'ingresso M_x .

4. Se D accetta, accettare; altrimenti, rifiutare.

Analizziamo ora il comportamento della macchina di Turing M' . Se M si ferma sull'ingresso w , allora anche M_x (costruita sulla base di M) si fermerà entro $100n^3 + 300$ passi per qualsiasi ingresso di lunghezza n . Pertanto, M' accetterà la codifica di M , poiché D accetterà anche M_x .

D'altra parte, se M non si arresta con l'ingresso w , allora M_x non si arresterà per nessun ingresso di lunghezza n , poiché simula M all'infinito. Pertanto, M' rifiuterà la codifica di M , poiché D rifiuterà anche M_x .

Abbiamo quindi costruito una macchina di Turing M' che decide L in base al comportamento di M sull'input w . Se avessimo un decider per L , potremmo usarlo per decidere il Problema di Halting costruendo M' ed eseguendo D su di essa. Tuttavia, poiché il problema di Halting è indecidibile, abbiamo raggiunto una contraddizione.

Pertanto, concludiamo che il linguaggio $L = \{M \mid M \text{ è una macchina che funziona in } 100n^3 + 300 \text{ tempo}\}$ è indecidibile.

- Il linguaggio delle macchine di Turing che non hanno punto fisso è definito come segue:

$$L_1 = \{w \in \{0,1\}^* \mid \exists x \in \{0,1\}^*: M_w(x) = x\}$$

Dimostrare che tale linguaggio è indecidibile

Per dimostrare che il linguaggio $L_1 = \{w \in \{0,1\}^* \mid \text{esiste } x \in \{0,1\}^* \text{ tale che } M_w(x) = x\}$, dove M_w è una macchina di Turing che accetta l'input x e restituisce x come output, è indecidibile, possiamo ridurlo al Problema dell'Halt di Turing, che è noto essere indecidibile.

Supponiamo di avere un algoritmo A che può decidere il linguaggio L1. Dato un input w, vogliamo determinare se esiste un punto fisso x per la macchina di Turing Mw. Costruiremo una macchina di Turing M' che utilizza l'algoritmo A per decidere l'Halt di Turing, il che contraddice la sua indecidibilità.

Costruzione della macchina di Turing M':

1. Sulla stringa di input x:
2. Costruisci una nuova macchina di Turing M_x come segue:
 - Simula la macchina di Turing Mw sull'input x.
 - Se Mw si arresta e restituisce x come output, accetta.
 - Altrimenti, se Mw non si arresta o restituisce un output diverso da x, entra in un loop infinito.
3. Esegui l'algoritmo A sulla codifica di M_x.
4. Se A accetta, accetta. Altrimenti, se A rifiuta, rifiuta.

Ora, analizziamo il comportamento della macchina di Turing M'. Se esiste un punto fisso x per Mw, allora M_x (costruito in base a Mw) si arresterà e restituirà x come output. Pertanto, M' accetterà l'input x, poiché A accetterà anche M_x.

Al contrario, se non esiste un punto fisso x per Mw, allora M_x continuerà a eseguire indefinitamente senza mai restituire x come output. Pertanto, M' entrerà in un loop infinito senza mai terminare, poiché A rifiuterà anche M_x.

Così abbiamo costruito una macchina di Turing M' che decide il linguaggio L1 basandosi sul comportamento di Mw. Se avessimo un algoritmo A che decidesse L1, potremmo utilizzarlo per decidere l'Halt di Turing costruendo M' e eseguendo A su di esso. Tuttavia, poiché l'Halt di Turing è indecidibile, abbiamo raggiunto una contraddizione.

Di conseguenza, concludiamo che il linguaggio $L1 = \{w \in \{0,1\}^* \mid \text{esiste } x \in \{0,1\}^* \text{ tale che } Mw(x) = x\}$ è indecidibile.

- Sia:

$$S = \{\langle M \rangle \mid \text{TM } M \text{ su input 3 in qualche punto scrive il simbolo "3" sulla terza cella del suo nastro}\}$$

Dimostrare che questo problema è indecidibile

Per dimostrare che il linguaggio $S = \{\langle M \rangle \mid \text{Macchina di Turing } M, \text{ a un certo punto su input 3, scrive il simbolo "3" sulla terza cella del suo nastro}\}$ è indecidibile, possiamo ridurlo al Problema di Halting, che è un noto problema indecidibile.

Supponiamo di avere un decider D per il linguaggio S. Data una macchina di Turing M, vogliamo determinare se M si ferma su un input w (cioè risolvere il Problema di Halting). Costruiremo una macchina di Turing M' che decida S, utilizzando D come subroutine.

Costruzione della macchina di Turing M':

1. Su input x, dove x è una stringa che codifica una macchina di Turing:
2. Costruire una macchina di Turing M_x come segue:
 - Se l'input non è 3, rifiuta.

- Simulare l'esecuzione di M sull'ingresso w finché non si arresta o finché la testina della macchina di Turing non raggiunge la terza cella del nastro.

- Se la testina raggiunge la terza cella e scrive il simbolo "3", accettare; altrimenti, rifiutare.

3. Eseguire il decider D sull'ingresso M_x.

4. Se D accetta, accetta; altrimenti, rifiuta.

Analizziamo ora il comportamento della macchina di Turing M'. Se M si ferma sull'input w, allora M_x (costruita sulla base di M) si fermerà prima o al punto in cui la testina raggiunge la terza cella del nastro. Se M scrive il simbolo "3" sulla terza cella, M' accetterà la codifica di M, così come D accetterà anche M_x.

D'altra parte, se M non si ferma all'ingresso w, o se M si ferma senza raggiungere la terza cella del nastro, M_x rifiuterà o entrerà in un ciclo infinito senza mai scrivere il simbolo "3" sulla terza cella. Pertanto, M' rifiuterà la codifica di M, così come D rifiuterà anche M_x.

Abbiamo quindi costruito una macchina di Turing M' che decide S in base al comportamento di M sull'input w. Se avessimo un decider per S, potremmo usarlo per decidere il Problema di Halting costruendo M' ed eseguendo D su di essa. Tuttavia, poiché il problema di Halting è indecidibile, abbiamo raggiunto una contraddizione.

Pertanto, concludiamo che il linguaggio $S = \{\langle M \rangle \mid \text{Macchina di Turing } M, \text{ a un certo punto su input 3, scrive il simbolo "3" sulla terza cella del suo nastro}\}$ è indecidibile.

- Utilizzare una riduzione da:

$$\text{Complement}(A_{TM}) = \{\langle M, w \rangle \mid M \text{ è una TM, } w \notin L(M)\}$$

a questa Macchina di Turing, nota come PARSE_{TM} :

$$\text{PARSE}_{TM} = \{\langle M \rangle \mid M \text{ è TM a 1 nastro, } |L(M)| \leq 1000\}$$

Dimostrare che il problema è indecidibile

Per dimostrare che il linguaggio $\text{PARSE TM} = \{\langle M \rangle \mid M \text{ è una macchina di Turing a 1 nastro e il linguaggio } L(M) \text{ riconosciuto da } M \text{ contiene al massimo 1000 stringhe}\}$ è indecidibile, possiamo ridurlo dal complemento della Macchina di Turing di Accettazione (ATM), indicato con Complemento(ATM).

Il complemento di ATM è definito come $\text{Complemento(ATM)} = \{\langle M, w \rangle \mid M \text{ è una macchina di Turing e la stringa di input } w \text{ non è nel linguaggio } L(M)\}$.

Mostriremo una riduzione da Complemento(ATM) a SPARSE TM, dimostrando che se avessimo un decider per SPARSE TM, potremmo usarlo per decidere Complemento(ATM), il che contraddice l'indecidibilità di Complemento(ATM).

Supponiamo di avere un decider D per SPARSE TM. Data una macchina di Turing M e una stringa di input w, vogliamo determinare se w non è nel linguaggio riconosciuto da M. Costruiremo una macchina di Turing M' a 1 nastro che decide SPARSE TM, usando D come subroutine.

Costruzione della macchina di Turing M':

1. Su input x, dove x è una stringa che codifica una macchina di Turing M e una stringa di input w:

2. Costruire una macchina di Turing a 1 nastro M_x come segue:

- Simulare l'esecuzione di M sull'ingresso w.

- Se M accetta w , rifiuta.

- Se M rifiuta w , accettare.

3. Eseguire il decider D sull'input M_x .

4. Se D accetta, accettare; altrimenti, rifiutare.

Analizziamo ora il comportamento della macchina di Turing M' . Se w è nel linguaggio riconosciuto da M , allora M_x (costruita sulla base di M e w) rifiuterà w , poiché M accetta w . Pertanto, M' rifiuterà la codifica di M e w , poiché D rifiuterà anche M_x .

D'altra parte, se w non fa parte del linguaggio riconosciuto da M , allora M_x accetterà w , poiché M rifiuta w . Pertanto, M' accetterà la codifica di M e w , poiché D accetterà anche M_x .

Abbiamo quindi costruito una macchina di Turing a 1 nastro M' che decide SPARSE TM in base al comportamento di M e w . Se disponessimo di un decider per SPARSE TM, potremmo usarlo per decidere Complement(ATM) costruendo M' ed eseguendo D su di essa. Tuttavia, poiché Complement(ATM) è indecidibile, abbiamo raggiunto una contraddizione.

Pertanto, concludiamo che il linguaggio SPARSE TM = $\{\langle M \rangle \mid M \text{ è una macchina di Turing a 1 nastro e il linguaggio } L(M) \text{ riconosciuto da } M \text{ contiene al massimo 1000 stringhe}\}$ è indecidibile.

- Dati i linguaggi:

$$L_1 = \{w\#x \mid w, x \in \{0, 1\}^* \text{ where } M_w \text{ with input } x \text{ visits each of its states at least once}\}$$

$$L_2 = \{w\#x \mid w, x \in \{0, 1\}^* \text{ where } M_w \text{ with input } x \text{ does not visit at least one of its states}\}$$

Possono essere considerati decidibili oppure indecidibili?

Si può dimostrare che il linguaggio $L_1 = \{w\#x \mid w, x \in \{0, 1\}^*\}$ in cui la macchina di Turing M_w , dato l'input x , visita almeno una volta ciascuno dei suoi stati} e il linguaggio $L_2 = \{w\#x \mid w, x \in \{0, 1\}^*\}$ in cui la macchina di Turing M_w , dato l'input x , non visita almeno uno dei suoi stati} è indecidibile.

Per dimostrarlo, possiamo ridurre il linguaggio L_1 a L_2 (cioè dimostrare che L_1 è riducibile a L_2) e poi dimostrare l'indecidibilità di L_2 .

Riduzione da L_1 a L_2 :

Dato un input $w\#x$ per L_1 , possiamo costruire un input equivalente $w'\#x$ per L_2 tale che se M_w visita ogni suo stato almeno una volta, allora $M_{w'}$ visita tutti i suoi stati tranne uno. Viceversa, se M_w non visita almeno uno dei suoi stati, allora $M_{w'}$ non visita nessuno dei suoi stati.

Applicando questa riduzione, possiamo vedere che se avessimo un decider per L_2 , potremmo risolvere L_1 trasformando l'input e usando il decider per L_2 . Tuttavia, poiché L_1 è indecidibile, questa riduzione dimostra che anche L_2 deve essere indecidibile.

Pertanto, sia L_1 che L_2 sono linguaggi indecidibili.

- Sia data la macchina $T = \{\langle M \rangle \mid M \text{ non si ferma su nessun input}\}$. Si dimostri che è indecidibile.

Per dimostrare che il linguaggio $L = \{\langle M \rangle \mid M \text{ è una macchina di Turing che non si ferma su nessun input}\}$ è indecidibile, possiamo utilizzare una riduzione dal Problema di Arresto (Halting Problem), che è noto per essere indecidibile.

Supponiamo di avere un decidere D per il linguaggio L. Dato un input $\langle M, w \rangle$, dove M è una macchina di Turing e w è una stringa di input, vogliamo determinare se M si arresta su w (risolvere il Problema di Arresto). Costruiremo una nuova macchina di Turing M' che decide il linguaggio L, utilizzando D come subroutine.

Costruzione della macchina di Turing M':

Sull'input x, dove x è una stringa che codifica una macchina di Turing M:

Costruiamo una nuova macchina di Turing M_x come segue:

Simuliamo l'esecuzione di M su tutti gli input possibili.

Se M si ferma su almeno uno degli input, allora M_x accetta.

Se M non si ferma su nessun input, allora M_x entra in un loop infinito.

Eseguiamo il decidere D sull'input M_x.

Se D accetta, allora M' accetta; altrimenti, M' rifiuta.

Analizziamo ora il comportamento della macchina di Turing M'. Se M si arresta su almeno un input, allora M_x (costruita in base a M) si arresterà su almeno uno degli input simulati, quindi M' rifiuterà l'encoding di M, poiché D rifiuterà M_x.

D'altra parte, se M non si arresta su nessun input, allora M_x entrerà in un loop infinito durante la simulazione di tutti gli input possibili. Di conseguenza, M' si bloccherà su M_x e non terminerà mai.

Pertanto, M' accetterà l'encoding di M, poiché D accetterà M_x.

Abbiamo quindi costruito una macchina di Turing M' che decide il linguaggio L in base al comportamento di M su tutti gli input possibili. Se avessimo un decidere per L, potremmo utilizzarlo per risolvere il Problema di Arresto, costruendo M' ed eseguendo D su di essa. Tuttavia, poiché il Problema di Arresto è indecidibile, abbiamo raggiunto una contraddizione.

Pertanto, possiamo concludere che il linguaggio $L = \{\langle M \rangle \mid M \text{ è una macchina di Turing che non si ferma su nessun input}\}$ è indecidibile.

- Dimostrare che $L_1 = \{\langle M \rangle \mid M \text{ è una macchina di Turing e visita } q_0 \text{ almeno due volte su } \epsilon\} \notin R$ non è decidibile

Per dimostrare che il linguaggio $L_1 = \{\langle M \rangle \mid M \text{ è una macchina di Turing che visita lo stato } q_0 \text{ almeno due volte sull'input vuoto } \epsilon\}$ non è decidibile, possiamo ridurlo al problema di Halting, che è un noto problema indecidibile.

Supponiamo di avere un decider D per L1. Data una macchina di Turing M, vogliamo determinare se M si ferma sull'ingresso vuoto ϵ (cioè risolvere il problema di Halting). Costruiremo una macchina di Turing M' che decida L1, utilizzando D come subroutine.

Costruzione della macchina di Turing M':

1. Su input x, dove x è una stringa che codifica una macchina di Turing M':

2. Costruire una macchina di Turing M_x come segue:

- Simulare l'esecuzione di M sull'input vuoto ϵ .

- Contare il numero di volte che M visita lo stato q_0 durante la simulazione.

- Se il conteggio è almeno 2, accettare; altrimenti, rifiutare.

3. Eseguire il decider D sull'ingresso M_x .

4. Se D accetta, accettare; altrimenti, rifiutare.

Analizziamo ora il comportamento della macchina di Turing M' . Se M si ferma sull'ingresso vuoto ϵ e visita lo stato q_0 almeno due volte durante l'esecuzione, allora M_x (costruita sulla base di M) accetterà, poiché il numero di visite a q_0 sarà almeno 2. Pertanto, M' accetterà la codifica di M , poiché D accetterà anche M_x .

D'altra parte, se M non si ferma sull'ingresso vuoto ϵ , o se M visita lo stato q_0 meno di due volte, M_x rifiuterà o entrerà in un ciclo infinito. Pertanto, M' rifiuterà la codifica di M , così come D rifiuterà anche M_x .

Abbiamo quindi costruito una macchina di Turing M' che decide L_1 in base al comportamento di M sull'input vuoto ϵ . Se avessimo un decider per L_1 , potremmo usarlo per decidere il Problema di Halting costruendo M' ed eseguendo D su di essa. Tuttavia, poiché il problema di Halting è indecidibile, abbiamo raggiunto una contraddizione.

Pertanto, concludiamo che il linguaggio $L_1 = \{\langle M \rangle \mid M \text{ è una macchina di Turing che visita lo stato } q_0 \text{ almeno due volte sull'input vuoto } \epsilon\}$ non è decidibile.

- Si consideri il problema di determinare se una macchina di Turing a due nastri scriva mai scrive mai un simbolo non vuoto sul suo secondo nastro durante il corso della sua computazione su una qualsiasi stringa di input. Formulate questo problema come un linguaggio e dimostrate che è indecidibile.

Per formulare il problema come linguaggio, possiamo definirlo come segue:

$L = \{\langle M \rangle \mid M \text{ è una macchina di Turing a due nastri che scrive un simbolo non vuoto sul suo secondo nastro durante il corso della sua computazione su qualsiasi stringa di input}\}$.

In altre parole, L è il linguaggio che consiste in codifiche di macchine di Turing a due nastri che scriveranno sempre un simbolo non vuoto sul secondo nastro, indipendentemente dalla stringa di input fornita.

Per dimostrare che questo problema è indecidibile, possiamo ridurlo al problema dell'halting. Il problema dell'halting è un noto problema indecidibile che chiede se una macchina di Turing si ferma su un dato input.

Supponiamo di avere una macchina di Turing H in grado di risolvere il problema dell'halting. Costruiremo una nuova macchina di Turing D che decida il nostro problema, L , utilizzando H come subroutine. L'idea è quella di simulare il calcolo della macchina di Turing a due nastri su tutte le possibili stringhe di input e verificare se scrive mai un simbolo non vuoto sul secondo nastro.

Ecco come funziona la macchina di Turing D :

1. Dato un input $\langle M \rangle$, dove M è una macchina di Turing a due nastri.
2. Costruire una nuova macchina di Turing M' che simuli M su tutte le possibili stringhe di input.
3. Per ogni stringa di input w :
 - a. Eseguire M su w utilizzando H come subroutine per verificare se M si arresta su w .
 - b. Se M si ferma su w , simulare M' su w finché non scrive un simbolo sul secondo nastro.
 - c. Se M' scrive un simbolo non vuoto sul secondo nastro, accettare.
4. Se il ciclo precedente si completa per tutte le possibili stringhe di input, rifiutare.

Se la macchina di Turing costruita M' scrive un simbolo non vuoto sul secondo nastro durante una qualsiasi delle simulazioni, D accetta. Altrimenti, rifiuta.

Ora, supponiamo che esista una macchina di Turing D' che decide L . Possiamo usare D' per risolvere il problema di halting come segue:

1. Dato un input $\langle M, w \rangle$, dove M è una macchina di Turing e w è una stringa di input.
2. Costruire una nuova macchina di Turing M' che ignori il suo input e si comporti come segue:
 - a. Simulare M su w .
 - b. Se M si ferma su w , scrivere un simbolo non vuoto sul secondo nastro.
3. Eseguire D' sull'ingresso $\langle M' \rangle$.
4. Se D' accetta, significa che M' scrive un simbolo non vuoto sul secondo nastro, il che implica che M si ferma su w . Accettare.
5. Se D' rifiuta, significa che M' non scrive un simbolo non vuoto sul secondo nastro, il che implica che M non si ferma su w . Rifiutare.

Costruendo D' che decide L , abbiamo risolto il problema dell'halting, che è noto per essere indecidibile. Di conseguenza, la nostra ipotesi iniziale che D' esista deve essere falsa.

Pertanto, possiamo concludere che il problema di determinare se una macchina di Turing a due nastri scrive mai un simbolo non vuoto sul suo secondo nastro durante il corso della sua computazione su qualsiasi stringa di input (L) è indecidibile.

- Consideriamo il problema di determinare se una macchina di Turing M su un input w tenta mai di spostare la testa a sinistra quando la testa si trova sulla cella del nastro più a sinistra.
Formulare questo problema come un linguaggio e dimostrate che è indecidibile.

Per formulare il problema come un linguaggio, possiamo definirlo nel seguente modo:

$L = \{\langle M, w \rangle \mid M \text{ è una macchina di Turing che su input } w \text{ tenta di spostare la testa a sinistra quando si trova sulla cella del nastro più a sinistra}\}$

In altre parole, L è il linguaggio che contiene le codifiche delle coppie $\langle M, w \rangle$ in cui la macchina di Turing M tenta di spostare la testa a sinistra quando si trova sulla cella del nastro più a sinistra, dato l'input w .

Per dimostrare che questo problema è indecidibile, possiamo utilizzare una riduzione dal problema dell'arresto (halting problem). Il problema dell'arresto è noto per essere indecidibile e consiste nel determinare se una macchina di Turing si ferma su un determinato input.

Supponiamo di avere una macchina di Turing H che può risolvere il problema dell'arresto. Costruiremo una nuova macchina di Turing D che decide il nostro problema L utilizzando H come sottoprogramma. L'idea è simulare la computazione della macchina di Turing a due nastri fornita su tutte le possibili stringhe di input e verificare se tenta di spostare la testa a sinistra sulla cella più a sinistra.

Ecco come funziona la macchina di Turing D :

1. Dato l'input $\langle M, w \rangle$, dove M è una macchina di Turing a due nastri.
2. Costruisci una nuova macchina di Turing M' che simula M su tutte le possibili stringhe di input.
3. Per ogni stringa di input w :

- a. Esegui M su w utilizzando H come sottoprogramma per verificare se M si arresta su w .
 - b. Se M si arresta su w , simula M' su w fino a quando tenta di spostare la testa a sinistra sulla cella più a sinistra.
 - c. Se M' tenta di spostare la testa a sinistra sulla cella più a sinistra, accetta.
4. Se il ciclo precedente viene completato per tutte le possibili stringhe di input, rifiuta.

Se la macchina di Turing M' costruita tenta di spostare la testa a sinistra sulla cella più a sinistra durante una delle simulazioni, D accetta. Altrimenti, la rifiuta.

Supponiamo ora che esista una macchina di Turing D' che decide L . Possiamo utilizzare D' per risolvere il problema dell'arresto nel seguente modo:

1. Dato l'input $\langle M, w \rangle$, dove M è una macchina di Turing e w è una stringa di input.
2. Costruisci una nuova macchina di Turing M' che ignora il suo input e si comporta come segue:
 - a. Simula M su w .
 - b. Se M si arresta su w , tenta di spostare la testa a sinistra sulla cella più a sinistra.
3. Esegui D' sull'input $\langle M' \rangle$.
4. Se D' accetta, significa che M' tenta di spostare la testa a sinistra sulla cella più a sinistra, implicando che M si arresta su w . Accetta.
5. Se D' rifiuta, significa che M' non tenta

Macchine di Turing: costruzione formale, varianti, etc.

- Una macchina di Turing-GPT è una macchina in grado di utilizzare il noto modello di calcolo artificiale GPT-4, con il quale risolvere problemi e rispondere a qualsiasi domanda posta sul proprio nastro di input. Qualora la domanda non sia conosciuta, la macchina è in grado lo stesso, variando tra più nastri e testine, di rispondere alla domanda. Descrivere questa macchina con una TM a nastro singolo, dimostrando la sua appartenenza ai linguaggi Turing-riconoscibili

Per descrivere la macchina di Turing-GPT utilizzando una macchina di Turing a nastro singolo, possiamo utilizzare una TM non deterministica che emula le capacità della macchina di calcolo GPT-4. La TM non deterministica può effettuare un doppio passaggio tra il nastro della TM e il "nastro virtuale" che rappresenta il modello di calcolo GPT-4.

Descriviamo le transizioni della TM non deterministica per implementare questa emulazione:

- Inizialmente, la TM non deterministica legge l'input sul proprio nastro di input.
- La TM non deterministica duplica l'input, creando una copia sul nastro virtuale per il passaggio al modello GPT-4.
- La TM non deterministica esegue un passaggio verso il nastro virtuale per utilizzare il modello GPT-4 per risolvere il problema o rispondere alla domanda. Questo passaggio non deterministico consente alla TM di esplorare diverse sequenze di azioni che potrebbero essere prese dal modello GPT-4.
- Durante l'esecuzione sul nastro virtuale, la TM non deterministica può utilizzare un insieme di regole o transizioni per simulare le azioni del modello GPT-4. Queste regole specificano come il modello GPT-4 risponde alle query o alle domande poste.
- Dopo aver eseguito il passaggio sul nastro virtuale, la TM non deterministica effettua un passaggio di ritorno al proprio nastro di input.
- La TM non deterministica legge il risultato ottenuto dal modello GPT-4 sul proprio nastro di input e può elaborarlo per produrre una risposta finale.
- La TM non deterministica può quindi continuare l'esecuzione sul proprio nastro di input, adottando diverse transizioni e regole specifiche per rispondere a domande o risolvere problemi non noti inizialmente.

La TM non deterministica descritta sopra emula l'interazione tra la macchina di calcolo GPT-4 e la macchina di Turing tradizionale. Poiché una TM non deterministica può simulare tutte le possibili sequenze di transizioni, la macchina di Turing descritta è in grado di riconoscere un linguaggio Turing-riconoscibile.

Tuttavia, va notato che l'effettiva implementazione di una TM non deterministica richiede risorse illimitate in termini di tempo e spazio, il che la rende puramente teorica. In pratica, la macchina di calcolo GPT-4 ha capacità che superano le limitazioni di una macchina di Turing tradizionale, come il tempo di esecuzione e la memoria disponibile.

- Una Universal Lego Machine è una macchina in grado di simulare il posizionamento di vari blocchi LEGO; da qualsiasi blocco è possibile, dato un insieme di forme definite, creare una serie probabilmente infinita di oggetti con essa. Descrivere questa macchina con una TM a nastro singolo, dimostrando la sua appartenenza ai linguaggi Turing-riconoscibili

Per rappresentare la Universal Lego Machine utilizzando una TM a nastro singolo, si possono seguire i seguenti passi:

Definire una serie di simboli per l'alfabeto della TM, che rappresentano i diversi blocchi LEGO e i loro orientamenti. Ad esempio, possiamo usare caratteri diversi per rappresentare ogni tipo di blocco LEGO e il suo orientamento.

Inizializzare il nastro con una rappresentazione dello stato iniziale dei blocchi LEGO, utilizzando l'alfabeto definito.

Definire un insieme finito di stati per la TM che rappresenti le diverse fasi di funzionamento della macchina, come ad esempio "posizionare un blocco" o "controllare le collisioni".

Definire un insieme di regole di transizione che descrivano come il TM passa da uno stato all'altro in base al simbolo corrente letto sul nastro. Queste regole devono anche descrivere come il TM aggiorna il nastro per rappresentare il nuovo stato dei blocchi LEGO dopo ogni operazione.

Eseguire il TM, simulando il posizionamento dei blocchi LEGO in base all'insieme di forme predefinite.

Ad esempio, consideriamo un caso semplice in cui la Universal Lego Machine può posizionare blocchi LEGO 1x1 in una linea orizzontale. Possiamo definire la TM come segue:

Alfabeto: {0, 1}, dove 0 rappresenta uno spazio vuoto e 1 rappresenta un blocco LEGO 1x1.

Nastro iniziale: 0000000000 (10 spazi vuoti)

Stati: {START, PLACE_BLOCK, HALT}

Regole di transizione:

(START, 0) -> (PLACE_BLOCK, 1, R) // piazza un blocco e si sposta a destra

(PLACE_BLOCK, 0) -> (PLACE_BLOCK, 1, R) // Piazza un blocco e si sposta a destra

(PLACE_BLOCK, 1) -> (HALT) // Si ferma quando incontra un blocco esistente

Questa TM posiziona blocchi LEGO 1x1 in linea orizzontale finché non incontra un blocco esistente. Il nastro verrà aggiornato per rappresentare il nuovo stato dei blocchi LEGO dopo ogni operazione.

In conclusione, descrivendo la Universal Lego Machine utilizzando una Macchina di Turing a nastro singolo, abbiamo dimostrato la sua appartenenza ai linguaggi riconoscibili da Turing. Questo dimostra che la macchina può essere rappresentata come un modello computazionalmente equivalente a una TM e può essere utilizzata per simulare la logica di qualsiasi algoritmo informatico.

- Un automa a ingresso/uscita temporizzato (Timed Input/Output Automaton - TIOA) è una variante di una macchina di Turing che incorpora il tempo come fattore nelle sue operazioni di ingresso e uscita. Opera su un nastro infinito diviso in celle, dove ogni cella può contenere un simbolo e un timestamp. La TIOA è dotata di un orologio che tiene traccia dello scorrere del tempo. Descrivere questa macchina con una TM a nastro singolo, dimostrando la sua appartenenza ai linguaggi Turing-riconoscibili

Per rispondere in modo preciso all'esercizio, descriveremo formalmente un Timed Input/Output Automaton (TIOA) utilizzando una macchina di Turing a nastro singolo. Dimostreremo sia la conversione da TIOA a macchina di Turing a nastro singolo che da macchina di Turing a nastro singolo a TIOA.

Descrizione del TIOA come macchina di Turing a nastro singolo:

Per convertire il TIOA in una macchina di Turing a nastro singolo, dobbiamo definire come rappresentare le informazioni temporizzate all'interno della macchina di Turing.

1. Definizione dell'alfabeto:

- L'alfabeto dell'input della macchina di Turing includerà sia i simboli di input della TIOA che i simboli di timestamp associati. Ad esempio, se l'alfabeto di input della TIOA è $\Sigma = \{a, b, c\}$, l'alfabeto dell'input della

macchina di Turing sarà $\Sigma' = \{a0, a1, b0, b1, c0, c1\}$, dove il suffisso "0" indica un timestamp di inizio e "1" indica un timestamp di fine.

- L'alfabeto del nastro della macchina di Turing includerà solo i simboli di input senza i timestamp associati. Quindi, l'alfabeto del nastro sarà $\Gamma = \{a, b, c\}$.

2. Definizione delle regole di transizione:

- Le regole di transizione della macchina di Turing dovranno tener conto dei timestamp associati ai simboli. Ad esempio, se una regola di transizione nella TIOA implica la scrittura del simbolo "a" con timestamp di inizio, la corrispondente regola di transizione nella macchina di Turing sarà responsabile di scrivere il simbolo "a" senza il timestamp.

- Sarà necessario introdurre una notazione aggiuntiva per tenere traccia del tempo all'interno della macchina di Turing. Ad esempio, potremmo utilizzare un simbolo speciale come "T" per rappresentare l'istante corrente all'interno della macchina di Turing.

3. Conversione da TIOA a macchina di Turing a nastro singolo:

- Ogni stato della TIOA sarà rappresentato come uno stato corrispondente nella macchina di Turing.

- Le transizioni nella TIOA saranno tradotte in regole di transizione nella macchina di Turing, prendendo in considerazione le modifiche necessarie per gestire il tempo.

Dimostrazione della conversione da macchina di Turing a nastro singolo a TIOA:

La conversione da macchina di Turing a nastro singolo a TIOA richiede l'inclusione di informazioni temporizzate e dell'orologio all'interno della TIOA.

1. Definizione dell'alfabeto:

- L'alfabeto dell'input della TIOA includerà sia i simboli dell'alfabeto del nastro della macchina di Turing che i simboli di timestamp. Ad esempio, se l'alfabeto del nastro della macchina di Turing è $\Gamma = \{a, b, c\}$, l'alfabeto dell'input della TIOA sarà $\Sigma =$

$\{a0, a1, b0, b1, c0, c1\}$, dove il suffisso "0" indica un timestamp di inizio e "1" indica un timestamp di fine.

- L'alfabeto del nastro della TIOA includerà solo i simboli dell'alfabeto del nastro della macchina di Turing senza i timestamp associati.

2. Definizione delle regole di transizione:

- Le regole di transizione della TIOA dovranno tener conto dei timestamp associati ai simboli.

- Sarà necessario introdurre un'istruzione speciale per leggere l'orologio e valutare il tempo corrente nella TIOA.

Questi passaggi dimostrano come convertire un TIOA in una macchina di Turing a nastro singolo e viceversa. La dimostrazione formale richiederà l'utilizzo di notazioni e simboli matematici per descrivere le regole di transizione, gli alfabeti e i comportamenti delle macchine coinvolte.

- Una macchina di Turing con azzeramento a sinistra è simile a una macchina di Turing ordinaria, ma la funzione di transizione ha la forma di funzione di transizione ha la forma

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, \text{RESET}\}.$$

Se $\delta(q, a) = (r, b, \text{RESET})$, quando la macchina si trova nello stato q e legge un a , la testina della macchina salta all'estremità sinistra del nastro dopo aver scritto b sul nastro ed entra nello stato r . Si noti che queste macchine non hanno la consueta capacità di spostare la testina di un simbolo a sinistra.

Dimostrare che le macchine di Turing con reset a sinistra riconoscono la classe dei linguaggi riconoscibili da Turing.

Per dimostrare che le macchine di Turing con reset a sinistra riconoscono la classe dei linguaggi riconoscibili da Turing, dobbiamo mostrare che possono simulare una macchina di Turing ordinaria.

Per fare ciò, dobbiamo fornire una costruzione formale di una simulazione di una macchina di Turing ordinaria utilizzando una macchina di Turing con reset a sinistra. La simulazione deve mantenere la stessa computazione e produrre lo stesso risultato.

Sia M una macchina di Turing ordinaria con un insieme di stati Q , un alfabeto di nastro Γ , una funzione di transizione δ , uno stato iniziale q_0 e un insieme di stati finali F . Dobbiamo costruire una macchina di Turing con reset a sinistra M' che simula M .

La macchina di Turing con reset a sinistra M' avrà un insieme di stati Q' , un alfabeto di nastro Γ , una funzione di transizione δ' , uno stato iniziale q_0' e un insieme di stati finali F' . La funzione di transizione δ' avrà la forma $\delta' : Q' \times \Gamma \rightarrow Q' \times \Gamma \times \{R, \text{RESET}\}$.

La costruzione di M' è la seguente:

1. Gli stati di M saranno gli stessi stati di M' . Quindi $Q' = Q$.

2. L'alfabeto di nastro di M' sarà lo stesso di M . Quindi $\Gamma' = \Gamma$.

3. La funzione di transizione di M' sarà definita come segue:

- Per ogni transizione $\delta(q, a) = (r, b, L)$ in M , avremo $\delta'(q, a) = (r, b, \text{RESET})$ in M' .

- Per ogni altra transizione $\delta(q, a) = (r, b, D)$ in M , avremo $\delta'(q, a) = (r, b, D)$ in M' . Questa transizione è la stessa in M' e M .

4. Lo stato iniziale di M' sarà lo stesso stato iniziale di M . Quindi $q_0' = q_0$.

5. Gli stati finali di M' saranno gli stessi stati finali di M . Quindi $F' = F$.

Questa costruzione mostra che ogni transizione di M può essere simulata da una transizione corrispondente in M' . In particolare, quando una transizione di M richiede di spostare la testina a sinistra, viene utilizzato il RESET per simulare questo comportamento nella macchina di Turing con reset a sinistra.

Inoltre, poiché la simulazione mantiene gli stessi stati iniziali e finali, e produce gli stessi risultati per ogni configurazione di input, possiamo affermare che le macchine di Turing con reset a sinistra riconoscono la stessa classe di linguaggi riconoscibili da Turing.

Quindi, le macchine di Turing con reset a sinistra sono in grado di riconoscere la classe dei linguaggi riconoscibili da Turing.

- Dimostrare che i TM a nastro singolo che non possono scrivere sulla porzione di nastro contenente la stringa di input la stringa di input riconoscono solo i linguaggi regolari.

Per dimostrare che le macchine di Turing a nastro singolo (STM) che non possono scrivere sulla porzione di nastro contenente la stringa di input riconoscono solo linguaggi regolari, dobbiamo dimostrare che qualsiasi linguaggio riconosciuto da una STM di questo tipo è un linguaggio regolare. Per farlo, è necessario costruire un'espressione regolare che generi lo stesso linguaggio della STM.

Sia L un linguaggio riconosciuto da un STM che non può scrivere sulla porzione di nastro contenente la stringa di input. Possiamo assumere che l'STM si ferma sempre su qualsiasi input, perché altrimenti riconoscerebbe un linguaggio infinito.

Sia w una stringa in L e sia n la lunghezza di w . Possiamo rappresentare il calcolo dell'STM su w come una sequenza di configurazioni, dove ogni configurazione è una stringa di lunghezza $n+1$ che rappresenta il contenuto del nastro e lo stato corrente dell'STM. Il primo carattere della configurazione rappresenta lo stato attuale dell'STM, mentre i restanti n caratteri rappresentano il contenuto del nastro, con la stringa di input al centro.

Poiché l'STM non può scrivere sulla porzione di nastro contenente la stringa di input, il contenuto di questa porzione di nastro rimane invariato per tutta la durata della computazione. Pertanto, possiamo rappresentare il calcolo dell'STM su w come una sequenza di stati che l'STM attraversa durante la lettura della stringa di input.

Sia q_0, q_1, \dots, q_k la sequenza di stati che l'STM attraversa durante la lettura della stringa di input w . Poiché l'STM si ferma sempre su w , esiste uno stato q_f tale che $(q_k, _) \rightarrow (q_f, _)$ è una transizione dell'STM. Possiamo assumere senza perdita di generalità che q_f sia uno stato accettante dell'STM.

Possiamo costruire un'espressione regolare R che genera lo stesso linguaggio dell'STM come segue:

- Sia X l'espressione regolare che genera la stringa di input w .
- Per ogni coppia di stati consecutivi q_i, q_{i+1} nella sequenza q_0, q_1, \dots, q_k , sia Y_i l'espressione regolare che genera l'insieme di stringhe che l'STM può leggere durante la transizione da q_i a q_{i+1} . Questo insieme di stringhe include la stringa di ingresso w . Questo insieme di stringhe include la stringa di input w e tutti i simboli aggiuntivi che l'STM può leggere durante la transizione da q_i a q_{i+1} .
- Sia Z l'espressione regolare che genera l'insieme di stringhe che l'STM può leggere durante la transizione da q_k a q_f . Questo insieme di stringhe include la stringa di input w e tutti i simboli aggiuntivi che l'STM può leggere durante la transizione da q_k a q_f .
- L'espressione regolare R si ottiene concatenando $X, Y_0, Y_1, \dots, Y_{k-1}$ e Z .

Poiché ogni Y_i è un'espressione regolare e la concatenazione di espressioni regolari è essa stessa un'espressione regolare, abbiamo dimostrato che R è un'espressione regolare che genera lo stesso linguaggio dell'STM. Pertanto, il linguaggio L riconosciuto dall'STM è un linguaggio regolare.

Questo risultato è interessante perché dimostra che una restrizione sulle capacità di una macchina di Turing può avere un impatto significativo sulla classe di linguaggi che può riconoscere.

- Esaminate la definizione formale di macchina di Turing per rispondere alle seguenti domande e spiegate il vostro ragionamento.
 - Una macchina di Turing può mai scrivere il simbolo vuoto sul suo nastro?
 - L'alfabeto del nastro Γ può essere uguale all'alfabeto di input Σ ?
 - La testa di una macchina di Turing può mai trovarsi nella stessa posizione in due passi successivi?

- Una macchina di Turing può contenere un solo stato?
- a. Sì, una macchina di Turing può scrivere il simbolo vuoto sul suo nastro. Il simbolo bianco fa parte dell'alfabeto del nastro Γ , che comprende tutti i simboli che possono apparire sul nastro. La macchina di Turing può scrivere il simbolo vuoto sul nastro specificandolo come simbolo di uscita nella sua funzione di transizione.
- b. Sì, l'alfabeto del nastro Γ può essere lo stesso dell'alfabeto di ingresso Σ . L'alfabeto del nastro Γ è un superset dell'alfabeto di input Σ , e include simboli aggiuntivi come il simbolo vuoto e qualsiasi altro simbolo di cui la macchina di Turing ha bisogno per eseguire la sua computazione. Pertanto, è possibile che Γ sia uguale a Σ .
- c. No, la testa di una macchina di Turing non può trovarsi nella stessa posizione in due passi successivi, a meno che la macchina di Turing non sia in uno stato di arresto. La macchina di Turing sposta la testa a sinistra o a destra sul nastro mentre legge e scrive simboli, e la sua funzione di transizione specifica il nuovo stato e il simbolo di uscita in base allo stato corrente e al simbolo di ingresso. Pertanto, la testa della macchina di Turing deve spostarsi in una nuova posizione sul nastro in ogni fase della sua computazione.
- d. No, una macchina di Turing non può contenere un solo stato. Come minimo, una macchina di Turing deve avere due stati: uno stato iniziale e uno stato di arresto. Lo stato iniziale è lo stato iniziale della macchina di Turing, mentre lo stato di arresto è lo stato finale in cui la macchina di Turing entra quando ha terminato la sua computazione. Senza questi due stati, la macchina di Turing non sarebbe in grado di avviare e arrestare la propria computazione e non sarebbe in grado di riconoscere alcun linguaggio.

- Spiegare perché la seguente non è una descrizione di una macchina di Turing legittima.

M_{bad} = "Su input $< p >$, un polinomio su variabili x_1, \dots, x_k :

1. Prova tutte le possibili impostazioni di x_1, \dots, x_k : a valori interi.
2. Valutare p su tutte queste impostazioni.
3. Se una di queste impostazioni ha valore 0, accettare; altrimenti, rifiutare".

La descrizione fornita di M_{bad} non rappresenta una macchina di Turing legittima perché viola alcuni dei requisiti fondamentali che definiscono una macchina di Turing.

1. La prima violazione riguarda il passaggio iniziale della macchina di Turing, che afferma di provare tutte le possibili impostazioni delle variabili x_1, \dots, x_k a valori interi. Una macchina di Turing ha un nastro di input di lunghezza finita e non può eseguire iterazioni infinite su input infiniti come richiesto in questa descrizione. Una macchina di Turing deve lavorare con un input specifico, che può essere di dimensione arbitrariamente grande ma non infinita.
2. La seconda violazione riguarda il passaggio in cui afferma di valutare il polinomio p su tutte le possibili impostazioni delle variabili. Valutare un polinomio su tutte le possibili impostazioni richiede un tempo infinito, il che non è consentito nelle macchine di Turing. Una macchina di Turing deve avere una sequenza finita di passi e deve terminare in un numero finito di passi.
3. La terza violazione riguarda la condizione di accettazione o rifiuto. La macchina di Turing accetta solo se raggiunge uno stato finale specificato, ma in questa descrizione si afferma che la macchina accetta se una delle impostazioni ha un valore 0. Questo non rispetta la definizione di accettazione di una macchina di Turing.

In sintesi, la descrizione fornita di M_bad non è una descrizione corretta di una macchina di Turing legittima poiché viola i principi fondamentali delle macchine di Turing, come la limitazione delle dimensioni dell'input, la necessità di eseguire un numero finito di passi e le condizioni di accettazione specificate.

Soluzione Esercizi 13/06

Automi e Linguaggi Formali – Seconda Parte

Macchine di Turing: costruzione formale, varianti, etc.

- *Macchine di Turing con funzione di transizione ristretta*

In questa variante, la funzione di transizione della macchina di Turing è limitata alla forma:

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{S\}$$

La macchina può rimanere solo nella stessa posizione e la testa non può muoversi a destra o a sinistra. Mostrare che le macchine di Turing con questa funzione di transizione ristretta riconoscono un sottoinsieme adeguato dei linguaggi riconoscibili da Turing.

Le macchine di Turing con una funzione di transizione ristretta, in cui la testa della macchina non può muoversi a destra o a sinistra, riconoscono un sottoinsieme adeguato dei linguaggi riconoscibili da una macchina di Turing standard. In particolare, possono riconoscere solo linguaggi che possono essere riconosciuti da macchine di Turing standard in cui la testa rimane sempre nella stessa posizione.

Per dimostrarlo, consideriamo un linguaggio L che può essere riconosciuto da una macchina di Turing standard. Questo significa che esiste una macchina di Turing M , con una funzione di transizione generale δ , che accetta L .

Possiamo costruire una macchina di Turing ristretta M' che riconosce lo stesso linguaggio L , ma con una funzione di transizione limitata alla forma $\delta': Q \times \Gamma \rightarrow Q \times \Gamma \times \{S\}$, in cui la testa non può muoversi.

Per fare ciò, modifichiamo la funzione di transizione di M in modo che, ogni volta che M avrebbe dovuto muovere la testa a destra o a sinistra, invece scriva lo stesso simbolo sulla stessa posizione e rimanga nello stesso stato. Formalmente, per ogni transizione di M della forma $\delta(q, a) = (p, b, D)$, dove D è " L " o " R ", creiamo una corrispondente transizione per M' della forma $\delta'(q, a) = (p, b, S)$.

Ora dimostriamo che M' riconosce lo stesso linguaggio L di M . Se M accetta una stringa w , allora esiste una sequenza di configurazioni C_1, C_2, \dots, C_n tale che C_1 è la configurazione iniziale, C_n è una configurazione accettante e per ogni $i = 1, 2, \dots, n-1$, la configurazione C_{i+1} è ottenuta dalla configurazione C_i applicando una transizione di M .

Poiché ogni transizione di M può essere simulata da una transizione corrispondente di M' , possiamo ottenere una sequenza di configurazioni C'_1, C'_2, \dots, C'_n per M' , dove C'_1 è la configurazione iniziale, C'_n è una configurazione accettante e per ogni $i = 1, 2, \dots, n-1$, la configurazione C'_{i+1} è ottenuta dalla configurazione C'_i applicando la stessa transizione di M' corrispondente alla transizione di M .

Pertanto, M' accetta la stessa stringa w di M , dimostrando che il linguaggio L è riconosciuto da M' .

In conclusione, le macchine di Turing con una funzione di transizione ristretta riconoscono un sottoinsieme adeguato dei linguaggi riconoscibili da una macchina di Turing standard, limitando la possibilità della testa di muoversi a destra o a sinistra.

- *Macchine di Turing con input di lunghezza variabile*

In questa variante, la macchina di Turing accetta stringhe di input di lunghezza variabile. Progettate una funzione di transizione che possa gestire stringhe di input di lunghezza arbitraria e discutete la potenza computazionale e i limiti delle macchine di Turing con input di lunghezza variabile.

Le macchine di Turing standard sono progettate per accettare input di lunghezza fissa, dove la lunghezza dell'input è specificata in anticipo e la testa della macchina può muoversi solo entro i limiti di questa lunghezza prefissata. Tuttavia, esiste una variante delle macchine di Turing che può gestire input di

lunghezza variabile, chiamate macchine di Turing con input di lunghezza variabile (VLTM, Variable-Length Turing Machine).

Le VLTM sono estensioni delle macchine di Turing standard che consentono la manipolazione di stringhe di lunghezza arbitraria. Per poter gestire input di lunghezza variabile, le VLTM richiedono un'adeguata codifica dell'input che indichi il confine tra diverse parti dell'input.

Una possibile implementazione delle VLTM prevede l'aggiunta di un simbolo speciale, chiamato "delimitatore", che separa le diverse parti dell'input. Ad esempio, se l'input consiste in una sequenza di numeri separati da virgole, il delimitatore potrebbe essere una virgola. La macchina di Turing potrebbe quindi utilizzare la presenza del delimitatore per identificare e manipolare le diverse parti dell'input.

La funzione di transizione di una VLTM deve essere in grado di gestire l'input di lunghezza variabile. Ciò significa che la funzione di transizione deve essere progettata per considerare il delimitatore e le possibili combinazioni di simboli nell'input. Ad esempio, potrebbe essere necessario definire transizioni specifiche per leggere i simboli fino al delimitatore e poi passare a una nuova parte dell'input.

La potenza computazionale delle VLTM con input di lunghezza variabile è equivalente a quella delle macchine di Turing standard. Ciò significa che le VLTM possono risolvere gli stessi problemi calcolabili e riconoscere gli stessi linguaggi formali delle macchine di Turing standard.

Tuttavia, le VLTM possono richiedere un tempo di esecuzione più lungo rispetto alle macchine di Turing standard per gestire input di lunghezza variabile. Poiché la lunghezza dell'input non è limitata, una VLTM potrebbe dover esaminare una quantità arbitraria di simboli prima di raggiungere il delimitatore e avanzare verso la parte successiva dell'input. Di conseguenza, la complessità computazionale delle VLTM può aumentare in presenza di input di lunghezza variabile.

Inoltre, le VLTM introducono una complessità aggiuntiva nella progettazione degli algoritmi, poiché è necessario gestire la logica di separazione e manipolazione delle diverse parti dell'input tramite il delimitatore.

In conclusione, le macchine di Turing con input di lunghezza variabile consentono di gestire input di lunghezza arbitraria, ma richiedono un'adeguata codifica dell'input e una progettazione specifica della funzione di transizione per gestire la variabilità della lunghezza dell'input. Nonostante ciò, le VLTM mantengono la stessa potenza computazionale delle macchine di Turing standard, ma possono richiedere più tempo di esecuzione e introducono complessità aggiuntiva nella progettazione degli algoritmi.

- *Macchine di Turing con alfabeti a nastro adattivi*

Considerare macchine di Turing con alfabeti a nastro adattivi, in cui la macchina può modificare o espandere dinamicamente il proprio alfabeto a nastro durante la computazione. Progettate una funzione di transizione che incorpori gli alfabeti a nastro adattivi e discutete la potenza computazionale e i limiti di queste macchine di Turing.

Le macchine di Turing con alfabeti a nastro adattivi sono un'estensione delle macchine di Turing standard in cui l'alfabeto del nastro può essere modificato o espanso durante la computazione. Ciò significa che la macchina di Turing può aggiungere simboli all'alfabeto del nastro durante l'esecuzione del programma, e queste modifiche possono essere utilizzate per rappresentare dati aggiuntivi o per supportare funzionalità di calcolo più complesse.

Per progettare una funzione di transizione per una macchina di Turing con alfabeti a nastro adattivi, dobbiamo definire come la macchina può modificare l'alfabeto del nastro. Ad esempio, potremmo definire una nuova transizione che consente alla macchina di aggiungere un nuovo simbolo all'alfabeto del nastro

ogni volta che incontra un simbolo che non è presente nell'alfabeto corrente. In questo modo, la macchina può espandere dinamicamente l'alfabeto del nastro per adattarsi ai dati di input.

La potenza computazionale delle macchine di Turing con alfabeti a nastro adattivi è equivalente a quella delle macchine di Turing standard, poiché entrambe le macchine possono simulare l'una l'altra. Tuttavia, l'uso di alfabeti a nastro adattivi può rendere alcune computazioni più efficienti o semplici. Ad esempio, se una macchina di Turing deve rappresentare dati strutturati come alberi o grafi, potrebbe utilizzare simboli speciali per rappresentare i nodi e le relazioni tra di essi, anziché utilizzare una codifica binaria o ASCII. In questo modo, la macchina potrebbe eseguire operazioni su questi dati strutturati in modo più naturale e intuitivo.

Tuttavia, ci sono anche limiti alle macchine di Turing con alfabeti a nastro adattivi. Una limitazione è che la dimensione dell'alfabeto del nastro può crescere senza limiti, il che potrebbe portare a problemi di memoria e di efficienza. Inoltre, la definizione di una funzione di transizione per una macchina di Turing con alfabeti a nastro adattivi potrebbe essere più complessa rispetto a una funzione di transizione per una macchina di Turing standard, poiché dobbiamo gestire la logica per la modifica e l'espansione dell'alfabeto del nastro.

- Progettare una macchina di Turing che prenda in input i coefficienti di un polinomio $c_1x^n + c_2x^{n-1} + \dots + c_nx + c_{n+1}$ e determina se il polinomio ha una radice $x = x_0$ tale che $|x_0| < (n + 1) \frac{c_{\max}}{|c_1|}$, dove c_{\max} è il più grande valore assoluto tra i coefficienti.
 - o Come input accetta un elenco codificato in binario di numeri interi che rappresentano i coefficienti del polinomio in ordine decrescente del loro esponente, verificando se esiste una radice che soddisfa la diseguaglianza.
 - o Risolvere il problema come specificato.

Per risolvere il problema specificato, possiamo progettare una macchina di Turing che prende in input una codifica binaria degli interi che rappresentano i coefficienti del polinomio. Di seguito viene fornita una possibile descrizione della macchina di Turing che risolve il problema:

1. La macchina di Turing inizia nella configurazione iniziale, leggendo il primo simbolo dell'input.
2. La macchina di Turing esegue i seguenti passi in un ciclo:
 - a. Legge il coefficiente corrente e il relativo esponente.
 - b. Calcola il valore assoluto del coefficiente corrente e lo confronta con il valore assoluto del coefficiente massimo (inizialmente impostato a 0).
 - c. Se il valore assoluto del coefficiente corrente è maggiore del valore assoluto del coefficiente massimo, aggiorna il valore del coefficiente massimo.
 - d. Se l'esponente corrente è pari a 0 (indicando il termine costante), passa al passo successivo. Altrimenti, torna al passo a per leggere il prossimo coefficiente.
 - e. Alla fine del ciclo, la macchina di Turing ha determinato il valore del coefficiente massimo.
3. La macchina di Turing esegue i seguenti passi per calcolare il valore di $(n + 1) * c_{\max} / |c_1|$:
 - a. Legge il valore di n (l'esponente massimo).
 - b. Legge il valore di c_{\max} (il coefficiente massimo).
 - c. Legge il valore di c_1 (il coefficiente del termine di grado più alto).
 - d. Calcola il valore di $(n + 1) * c_{\max} / |c_1|$.

4. La macchina di Turing esegue un altro ciclo per verificare se esiste una radice $x = x_0$ tale che $|x_0| < (n + 1) * c_{\max} / |c_1|$:

- a. Legge il valore del coefficiente corrente e il relativo esponente.
- b. Calcola il valore assoluto del coefficiente corrente e lo divide per il valore di $(n + 1) * c_{\max} / |c_1|$.
- c. Se il valore calcolato è maggiore di 1, passa al passo successivo. Altrimenti, accetta l'input e termina.
- d. Se l'esponente corrente è pari a 0 (indicando il termine costante), passa al passo successivo. Altrimenti, torna al passo a per leggere il prossimo coefficiente.
- e. Se il ciclo termina senza trovare una radice che soddisfi la diseguaglianza, rifiuta l'input e termina.

Questa è solo una possibile implementazione della macchina di Turing per risolvere il problema specificato. È importante notare che l'implementazione esatta potrebbe variare in base alle specifiche dettagliate del codice di input e all'encoding utilizzato per rappresentare i coefficienti e gli esponenti.

- Dimostrare il Teorema della gerarchia temporale, che afferma che per due funzioni $f(n)$ e $g(n)$ costruibili nel tempo, tali che $f(n)$ cresce più lentamente di $g(n)$, esiste un linguaggio che può essere deciso da una macchina di Turing in tempo $O(g(n))$ ma non in tempo $O(f(n))$ attraverso una macchina di Turing, dimostrando che è riconoscibile da Turing.

Il Teorema della gerarchia temporale afferma che, per due funzioni costruibili nel tempo $f(n)$ e $g(n)$ tali che $f(n)$ cresce più lentamente di $g(n)$, esiste un linguaggio che può essere deciso da una macchina di Turing in tempo $O(g(n))$ ma non in tempo $O(f(n))$ attraverso una macchina di Turing, dimostrando che è riconoscibile da Turing.

Per dimostrare questo teorema, possiamo utilizzare una tecnica di diagonalizzazione simile a quella utilizzata nella dimostrazione del Teorema di diagonalizzazione di Cantor. Supponiamo di avere due funzioni costruibili nel tempo $f(n)$ e $g(n)$, dove $f(n)$ cresce più lentamente di $g(n)$. Sia L un linguaggio che può essere deciso in tempo $O(g(n))$ da una macchina di Turing, ma non in tempo $O(f(n))$ attraverso una macchina di Turing.

Costruiamo una tabella infinita in cui le righe sono etichettate da tutte le stringhe binarie di lunghezza n , e le colonne sono etichettate da tutte le macchine di Turing che possono essere costruite in tempo $O(f(n))$. Ogni cella della tabella contiene il risultato della macchina di Turing corrispondente quando viene eseguita sull'input corrispondente.

Per costruzione, sappiamo che esiste almeno una riga della tabella che non viene riconosciuta dalla macchina di Turing in tempo $O(f(n))$, ma che può essere riconosciuta dalla macchina di Turing in tempo $O(g(n))$. Sia w questa riga.

Ora costruiamo una nuova macchina di Turing che utilizza la tabella come input, eseguendo la seguente procedura:

- Inizializza un contatore i a 1
- Esegui la macchina di Turing corrispondente alla riga i della tabella sull'input w
- Se la macchina di Turing accetta w , accetta il linguaggio, altrimenti incrementa i di 1 e ripeti dal passo 2.

Ora dimostriamo che questa nuova macchina di Turing riconosce il linguaggio L in tempo $O(g(n))$. Infatti, in quanto la macchina di Turing esegue la tabella in ordine, alla riga i -esima, la macchina di Turing che riconosce L ha avuto tempo $O(g(n))$ per decidere se accettare o meno l'input w . Pertanto, la complessità temporale complessiva della macchina di Turing è $O(g(n))$.

D'altra parte, dimostriamo che questa nuova macchina di Turing non può essere implementata in tempo $O(f(n))$. Infatti, se ci fosse una macchina di Turing che potesse fare ciò, allora potremmo usare questa macchina di Turing per eseguire la tabella in tempo $O(f(n))$, e quindi la riga non riconosciuta dalla macchina di Turing in tempo $O(f(n))$ sarebbe stata riconosciuta in questo tempo, il che è una contraddizione. Quindi, abbiamo dimostrato che esiste un linguaggio che può essere deciso in tempo $O(g(n))$, ma non in tempo $O(f(n))$, dimostrando così il Teorema della gerarchia temporale.

- Consideriamo la formica di Langton, un automa cellulare che opera su una griglia. La formica si muove sulla griglia, cambiando il colore di ogni cella che visita e modificando la sua direzione in base al colore della cella corrente. Segue una spiegazione più dettagliata:
 - o A ogni passo, la formica legge il colore della cella su cui si trova.
 - Se la cella è vuota, la formica ruota di 90 gradi verso destra, cambia il colore della cella in nero e avanza di un passo.
 - Se la cella è nera, la formica ruota di 90 gradi verso sinistra, cambia il colore della cella in bianco e avanza di un passo.
 - Dopo aver eseguito l'azione, la formica ripete il processo dal punto 1, considerando il nuovo colore della cella su cui si trova.
 - o Progettare una macchina di Turing che simuli la formica di Langton su un nastro infinito bidimensionale

Per progettare una macchina di Turing che simuli la formica di Langton su un nastro infinito bidimensionale, possiamo rappresentare la griglia come una matrice di celle sul nastro della macchina di Turing. Inizialmente, la matrice è vuota, e la formica si trova in una posizione arbitraria.

La macchina di Turing può essere implementata con uno stato che rappresenta la posizione della formica e la direzione in cui sta guardando. Inoltre, dobbiamo mantenere una rappresentazione della matrice di celle sul nastro, in modo che la macchina possa leggere e scrivere i colori delle celle.

La funzione di transizione della macchina di Turing può essere progettata per implementare le tre regole della formica di Langton:

1. Se la cella su cui si trova la formica è vuota, ruota di 90 gradi verso destra, cambia il colore della cella in nero e avanza di un passo.
2. Se la cella su cui si trova la formica è nera, ruota di 90 gradi verso sinistra, cambia il colore della cella in bianco e avanza di un passo.
3. Dopo aver eseguito l'azione, la formica ripete il processo dal punto 1, considerando il nuovo colore della cella su cui si trova.

La funzione di transizione può essere implementata come segue:

1. Leggi il colore della cella sulla quale si trova la formica.
2. Se il colore è bianco, scrivi il colore nero sulla cella, ruota di 90 gradi verso destra e avanza di un passo.
3. Se il colore è nero, scrivi il colore bianco sulla cella, ruota di 90 gradi verso sinistra e avanza di un passo.
4. Ripeti dal punto 1, considerando il nuovo colore della cella su cui si trova la formica.

Questa funzione di transizione può essere eseguita ripetutamente dalla macchina di Turing per simulare il comportamento della formica di Langton. La posizione e la direzione della formica possono essere rappresentate come parte dello stato della macchina di Turing, e la matrice di celle può essere rappresentata come parte del nastro.

In questo modo, la macchina di Turing può simulare l'evoluzione della griglia come descritto dalla formica di Langton. La simulazione continuerà all'infinito, poiché la griglia è infinita e la formica può continuare a muoversi in qualsiasi direzione.

Indecidibili, Decidibili & Company

- Definiamo $\Sigma = \{a, b, c, \dots, z\}$ è l'insieme delle lettere dell'alfabeto inglese. Dimostrare che il seguente linguaggio è indecidibile: $Gatti - VS - Cani = \{(M) \mid "gatti" \in L(M) \text{ o } "cani" \in L(M), \text{ ma non entrambi}\}$.

Per dimostrare che il linguaggio Gatti-VS-Cani è indecidibile, possiamo utilizzare il metodo di diagonalizzazione.

Supponiamo per assurdo che esista una macchina di Turing TM che decida il linguaggio Gatti-VS-Cani. Questa macchina prende in input una codifica di un'altra macchina di Turing M e determina se "gatti" appartiene al linguaggio di M, se "cani" appartiene al linguaggio di M, o entrambi.

Costruiamo ora una nuova macchina di Turing M' che opera come segue:

M':

- Per ogni input x:
 - Simula l'azione di M su x.
 - Se M accetta x, allora M' rifiuta x.
 - Se M rifiuta x, allora M' accetta x.

Ora, prendiamo in considerazione l'input "gatti". Se M accetta "gatti", allora M' rifiuterà "gatti" e viceversa. In altre parole, se "gatti" appartiene al linguaggio di M, allora "gatti" non appartiene al linguaggio di M'. Se invece "gatti" non appartiene al linguaggio di M, allora "gatti" appartiene al linguaggio di M'.

Ora applichiamo M' a sé stessa. Se M' accetta la sua stessa codifica, allora, secondo la definizione di M', M' rifiuta la sua stessa codifica. Se M' rifiuta la sua stessa codifica, allora, secondo la definizione di M', M' accetta la sua stessa codifica.

Ciò porta a una contraddizione: se M' accetta la sua stessa codifica, allora M' rifiuta la sua stessa codifica, ma se M' rifiuta la sua stessa codifica, allora M' accetta la sua stessa codifica. Quindi, non può esistere una macchina di Turing TM che decida il linguaggio Gatti-VS-Cani.

Di conseguenza, il linguaggio Gatti-VS-Cani è indecidibile. Non esiste una macchina di Turing che possa determinare se una data macchina di Turing accetta "gatti" o "cani", ma non entrambi.

- Data una TM M, esiste un limite K_M tale che iniziando la computazione da una qualsiasi configurazione c la macchina si fermi dopo al massimo k_M passi?

Non esiste un limite K_M generale valido per tutte le macchine di Turing M. Ci sono infatti alcune macchine di Turing che non si fermano mai per alcuna configurazione c, e quindi non esiste un limite al numero di passi che la macchina deve eseguire prima di fermarsi.

Inoltre, esistono altre macchine di Turing che si comportano in modo imprevedibile, per cui non è possibile stabilire un limite al numero di passi che la macchina deve eseguire prima di fermarsi.

Tuttavia, esistono alcune classi di macchine di Turing per le quali è possibile stabilire un limite K_M al numero di passi che la macchina deve eseguire prima di fermarsi, a seconda dei dettagli specifici dell'implementazione della macchina.

Ad esempio, se M è una macchina di Turing a nastro singolo che utilizza un alfabeto finito e ha un numero finito di stati, allora esiste un limite K_M al numero di passi che la macchina deve eseguire prima di fermarsi per qualsiasi configurazione c . Questo limite può essere calcolato a priori, e dipende dal numero di stati, dal numero di simboli dell'alfabeto e dalla lunghezza massima del nastro.

In generale, tuttavia, non esiste un limite generale al numero di passi che una macchina di Turing deve eseguire prima di fermarsi, e questo dipende dalle specifiche della macchina stessa.

- $NXOR$ è il linguaggio di $\langle M, x, y \rangle$ | la TM M accetta entrambi gli input x e y o non accetta nessuno dei due. Dimostrare che, dato un certo M trovare 2 input x e y s.t. $\langle M, x, y \rangle \in NXOR$ può essere fatto in tempo polinomiale.

Per dimostrare che è possibile trovare due input x e y tali che $\langle M, x, y \rangle \in NXOR$ in tempo polinomiale per una data macchina di Turing M , possiamo utilizzare un approccio di simulazione.

Consideriamo la seguente procedura per trovare gli input x e y :

1. Simula la macchina di Turing M su tutti i possibili input di lunghezza massima n , dove n è una funzione polinomiale rispetto alla dimensione di M . Ogni simulazione viene eseguita per un numero massimo di passi polinomiale rispetto a n .
2. Osserva il comportamento di M su ogni possibile coppia di input. Se M accetta entrambi gli input, allora prendiamo questi due input (x e y) e terminiamo la procedura.
3. Se M rifiuta almeno uno degli input, continuiamo con la simulazione su altri input fino a quando non troviamo una coppia di input che soddisfa la condizione richiesta.

Poiché la simulazione di M su tutti i possibili input di lunghezza massima n richiede un numero finito di passi polinomiali rispetto a n e il numero totale di coppie di input è anch'esso polinomiale rispetto a n , la procedura descritta sopra terminerà in tempo polinomiale.

Inoltre, la procedura garantisce di trovare una coppia di input x e y tali che $\langle M, x, y \rangle \in NXOR$, poiché stiamo esplorando tutte le possibili combinazioni di input e verifichiamo se M accetta entrambi gli input o non accetta nessuno dei due.

Quindi, è possibile trovare due input x e y tali che $\langle M, x, y \rangle \in NXOR$ per una data macchina di Turing M in tempo polinomiale.

- Siano M e N due macchine di Turing. Perché non è possibile dimostrare che M e N calcolano la stessa funzione? Dimostrare che questo problema è indecidibile.

Non è possibile dimostrare in generale che due macchine di Turing M e N calcolano la stessa funzione perché il problema di equivalenza tra due macchine di Turing è indecidibile, ovvero non esiste alcun algoritmo che possa determinare se due macchine di Turing sono equivalenti per ogni possibile input.

Per dimostrare che il problema di equivalenza tra due macchine di Turing è indecidibile, possiamo utilizzare una riduzione dal problema dell'arresto di una macchina di Turing, che è noto essere indecidibile.

Supponiamo di avere una macchina di Turing H che, dato in input una coppia di macchine di Turing (M, N) , determina se M e N calcolano la stessa funzione. Costruiamo una nuova macchina di Turing K che utilizza H per decidere se una macchina di Turing P si arresta su un input w .

La macchina di Turing K funziona in questo modo:

1. Costruisci una nuova macchina di Turing M' che, dato in input x , esegue la seguente procedura:

- a. Esegui la macchina di Turing P sull'input w.
 - b. Se P si arresta su w, accetta x.
 - c. Altrimenti, entra in un loop infinito.
2. Costruisci una nuova macchina di Turing N' che entra in un loop infinito su qualsiasi input.
3. Usa H per determinare se M' e N' calcolano la stessa funzione.
4. Se H accetta la coppia (M', N'), ovvero se M' e N' calcolano la stessa funzione, allora P si arresta su w, altrimenti P non si arresta su w.

In altre parole, la macchina di Turing K costruisce una nuova macchina di Turing M' che esegue P su w e accetta solo se P si arresta su w, e una nuova macchina di Turing N' che entra in un loop infinito su qualsiasi input. Quindi, K utilizza H per determinare se M' e N' calcolano la stessa funzione, ovvero se M' accetta solo se P si arresta su w. Se H accetta, allora M' e N' calcolano la stessa funzione, e quindi P si arresta su w, altrimenti M' e N' non calcolano la stessa funzione, e quindi P non si arresta su w.

Tuttavia, questo implica che se esistesse un algoritmo per determinare se M e N calcolano la stessa funzione, allora potremmo utilizzarlo per risolvere il problema dell'arresto di una macchina di Turing, il che è impossibile. Pertanto, il problema di equivalenza tra due macchine di Turing è indecidibile.

- Siano A, B due linguaggi definiti come
 $A = \{\langle M \rangle | M(\langle M \rangle) = \text{rifiuta}\}$ e $B = \{\langle M \rangle | M(\langle M \rangle) = \text{accetta}\}$. In altre parole, A è l'insieme delle descrizioni dei TM che rifiutano le proprie descrizioni e B è definito in modo simile. I linguaggi A e B sono disgiunti e sono entrambi riconoscibili da Turing. Si rovi che non esiste un linguaggio decidibile $C \subseteq \Sigma^*$ tale che $A \subseteq C$ e $B \subseteq \bar{C}$.

Per dimostrare che non esiste un linguaggio decidibile $C \subseteq \Sigma^*$ tale che $A \subseteq C$ e $B \subseteq \neg C$, possiamo utilizzare un argomento di diagonalizzazione.

Supponiamo per assurdo che esista un linguaggio decidibile C che soddisfa $A \subseteq C$ e $B \subseteq \neg C$. Consideriamo la seguente macchina di Turing M_C:

M_C:

- Prende in input una descrizione di una macchina di Turing M.
- Simula l'azione di M sulla sua descrizione $\langle M \rangle$.
- Se M rifiuta $\langle M \rangle$, allora M_C accetta.
- Se M accetta $\langle M \rangle$, allora M_C rifiuta.

Ora, prendiamo in considerazione l'input $\langle M_C \rangle$ per M_C stesso. Se M_C accetta $\langle M_C \rangle$, allora secondo la definizione di M_C, M_C rifiuta $\langle M_C \rangle$. D'altra parte, se M_C rifiuta $\langle M_C \rangle$, allora secondo la definizione di M_C, M_C accetta $\langle M_C \rangle$.

Questo ci porta a una contraddizione: se M_C accetta $\langle M_C \rangle$, allora M_C rifiuta $\langle M_C \rangle$, ma se M_C rifiuta $\langle M_C \rangle$, allora M_C accetta $\langle M_C \rangle$. Quindi, non può esistere una macchina di Turing M_C che soddisfi $A \subseteq C$ e $B \subseteq \neg C$.

Di conseguenza, non può esistere un linguaggio decidibile $C \subseteq \Sigma^*$ tale che $A \subseteq C$ e $B \subseteq \neg C$. I linguaggi A e B sono disgiunti e non possono essere inclusi e complementari rispetto a un unico linguaggio decidibile.

- Definiamo $L = \{< M, w, k > \mid M(w) \text{ raggiunge la configurazione } \alpha q \beta \text{ con } |\alpha \beta| \geq k\}$, con $\alpha, \beta \in \Gamma^*$. Tale linguaggio può essere considerato decidibile?

Il linguaggio L non può essere considerato decidibile, in quanto il problema di determinare se una macchina di Turing raggiunge una determinata configurazione in un numero di passi limitato non è decidibile in generale.

Per dimostrare che L non è decidibile, possiamo utilizzare una riduzione dal problema dell'arresto di una macchina di Turing, che è noto essere indecidibile.

Supponiamo di avere una macchina di Turing H che, dato in input una tripla $< M, w, k >$, determina se $M(w)$ raggiunge la configurazione $\alpha q \beta$ con $|\alpha \beta| \geq k$. Costruiamo una nuova macchina di Turing K che utilizza H per decidere se una macchina di Turing P si arresta su un input w.

La macchina di Turing K funziona in questo modo:

1. Costruisci una nuova macchina di Turing M' che, dato in input x, esegue la seguente procedura:
 - a. Esegui la macchina di Turing P sull'input w.
 - b. Se P si arresta su w, accetta x.
 - c. Altrimenti, entra in un loop infinito.
2. Costruisci una nuova stringa w' che consiste in w seguito da un simbolo qualsiasi.
3. Usa H per determinare se M' raggiunge la configurazione $\alpha q \beta$ con $|\alpha \beta| \geq k$ su input w' .
4. Se H accetta, allora P si arresta su w, altrimenti P non si arresta su w.

In altre parole, la macchina di Turing K costruisce una nuova macchina di Turing M' che esegue P su w e accetta solo se P si arresta su w, e utilizza H per determinare se M' raggiunge la configurazione $\alpha q \beta$ con $|\alpha \beta| \geq k$ su input w' . Se H accetta, allora M' raggiunge la configurazione $\alpha q \beta$ con $|\alpha \beta| \geq k$ su input w' , il che significa che P si arresta su w, altrimenti M' non raggiunge la configurazione $\alpha q \beta$ con $|\alpha \beta| \geq k$ su input w' , il che significa che P non si arresta su w.

Tuttavia, questo implica che se esistesse un algoritmo per determinare se una macchina di Turing raggiunge una determinata configurazione in un numero di passi limitato, allora potremmo utilizzarlo per risolvere il problema dell'arresto di una macchina di Turing, il che è impossibile. Pertanto, il linguaggio L non è decidibile.

- Il problema di rilevare un virus, definito come *InfectFile* è noto essere un problema indecidibile; non si può infatti sapere con certezza se il programma che viene eseguito può essere considerato pericoloso o meno; occorre dimostrare che in almeno un'occorrenza possa essere pericoloso e sia dimostrato dalla base di dati dell'antivirus che si sta usando. Dimostra che *InfectFile* è effettivamente indecidibile usando una riduzione.

Per dimostrare che il problema InfectFile è indecidibile, possiamo utilizzare una riduzione dal problema dell'arresto, che è noto essere indecidibile. Supponiamo per assurdo che esista un algoritmo o una macchina di Turing che decida il problema InfectFile.

Il problema dell'arresto chiede se, dato un programma P e un input I, il programma P termina la sua esecuzione su input I o entra in un ciclo infinito. Formalmente, il problema dell'arresto può essere definito come segue:

$\text{Halt} = \{< P, I > \mid \text{il programma P termina l'esecuzione su input I}\}$.

Costruiamo ora una riduzione dal problema dell'arresto al problema InfectFile.

Supponiamo di avere un algoritmo A che risolve il problema InfectFile. Possiamo utilizzare A per risolvere il problema dell'arresto nel seguente modo:

Data un'istanza del problema dell'arresto $\langle P, I \rangle$, costruiamo un file F che rappresenta un programma che esegue P su input I. Inoltre, includiamo una parte di codice nel file F che infetta altri file nel sistema. La presenza di questa parte di codice può essere rilevata da un antivirus.

Ora, se l'algoritmo A segnala che il file F è infetto, possiamo concludere che il programma P entra in un ciclo infinito sull'input I nel problema dell'arresto. Altrimenti, se A segnala che il file F non è infetto, possiamo concludere che il programma P termina la sua esecuzione sull'input I nel problema dell'arresto.

Quindi, se esistesse un algoritmo o una macchina di Turing che risolve il problema InfectFile, potremmo utilizzarlo per risolvere il problema dell'arresto, che è noto essere indecidibile. Tuttavia, ciò contraddice l'ipotesi che il problema InfectFile sia decidibile.

Pertanto, possiamo concludere che il problema InfectFile è effettivamente indecidibile, utilizzando una riduzione dal problema dell'arresto. Non esiste un algoritmo generale che possa determinare con certezza se un programma è pericoloso o meno in base ai dati dell'antivirus, proprio perché il problema dell'arresto è indecidibile.

- Non tutti i file possono essere compressi; supponi di avere un file che, in alcune circostanze (già compresso, criptato oppure eccessiva perdita di qualità) non sia possibile comprimere o farlo ulteriormente. Quanto descritto rappresenta un caso in cui non si può decidere la natura della soluzione, ma in alcuni casi può esserlo. Formula questo problema come indecidibile e usa una riduzione per dimostrarlo.

Il problema di decidere se un dato file può essere compresso o meno, o se può essere compresso ulteriormente, è in generale indecidibile.

Per dimostrarlo, possiamo utilizzare una riduzione dal problema dell'arresto di una macchina di Turing, che è noto essere indecidibile.

Supponiamo di avere una macchina di Turing H che, dato in input un file F, determina se F può essere compresso o meno. Costruiamo una nuova macchina di Turing K che utilizza H per decidere se una macchina di Turing P si arresta su un input w.

La macchina di Turing K funziona in questo modo:

1. Costruisci una nuova macchina di Turing M' che, dato in input x, esegue la seguente procedura:
 - a. Esegui la macchina di Turing P sull'input w.
 - b. Se P si arresta su w, accetta x.
 - c. Altrimenti, entra in un loop infinito.
2. Costruisci un file F che consiste nella rappresentazione della macchina di Turing M'.
3. Usa H per determinare se F può essere compresso o meno.
4. Se H accetta, allora P si arresta su w, altrimenti P non si arresta su w.

In altre parole, la macchina di Turing K costruisce una nuova macchina di Turing M' che esegue P su w e accetta solo se P si arresta su w, e costruisce un file F che rappresenta la macchina di Turing M'. Quindi, K utilizza H per determinare se F può essere compresso o meno. Se H accetta, allora F può essere compresso,

il che significa che P si arresta su w , altrimenti F non può essere compresso, il che significa che P non si arresta su w .

Tuttavia, questo implica che se esistesse un algoritmo per determinare se un file può essere compresso o meno, allora potremmo utilizzarlo per risolvere il problema dell'arresto di una macchina di Turing, il che è impossibile. Pertanto, il problema di decidere se un dato file può essere compresso o meno, o se può essere compresso ulteriormente, è indecidibile.

- Considera il linguaggio $Different_{PDA_{DFA}} = \{< M_1, M_2 > \mid M_1 \text{ è un PDA ed } M_2 \text{ è un DFA con } L(M_1) \neq L(M_2)\}$. Si mostri che tale linguaggio è indecidibile.

Per dimostrare che il linguaggio $Different_{(PDA_DFA)}$ è indecidibile, possiamo utilizzare una riduzione dal problema dell'arresto, che è noto essere indecidibile. Supponiamo per assurdo che esista un algoritmo o una macchina di Turing che decida il linguaggio $Different_{(PDA_DFA)}$.

Il problema dell'arresto per una macchina di Turing M , notato come $Halt(M)$, richiede di determinare se la macchina M termina la sua esecuzione su un input vuoto o entra in un ciclo infinito. Formalmente, il problema dell'arresto può essere definito come segue:

$Halt = \{< M > \mid \text{la macchina di Turing } M \text{ termina l'esecuzione su input vuoto}\}$.

Costruiamo ora una riduzione dal problema dell'arresto al linguaggio $Different_{(PDA_DFA)}$.

Supponiamo di avere un algoritmo A che risolve il linguaggio $Different_{(PDA_DFA)}$. Possiamo utilizzare A per risolvere il problema dell'arresto nel seguente modo:

Data un'istanza del problema dell'arresto $< M >$, costruiamo una macchina di Turing M_1 che simula M su input vuoto e una macchina di Turing M_2 che accetta sempre l'input vuoto, ovvero il linguaggio vuoto $\{\}$. In altre parole, M_2 è un DFA che accetta il linguaggio vuoto.

Ora, se l'algoritmo A segnala che $< M_1, M_2 >$ appartiene a $Different_{(PDA_DFA)}$, possiamo concludere che la macchina di Turing M entra in un ciclo infinito sull'input vuoto nel problema dell'arresto. Altrimenti, se A segnala che $< M_1, M_2 >$ non appartiene a $Different_{(PDA_DFA)}$, possiamo concludere che la macchina di Turing M termina la sua esecuzione sull'input vuoto nel problema dell'arresto.

Quindi, se esistesse un algoritmo o una macchina di Turing che risolve il linguaggio $Different_{(PDA_DFA)}$, potremmo utilizzarlo per risolvere il problema dell'arresto, che è noto essere indecidibile. Tuttavia, ciò contraddice l'ipotesi che il linguaggio $Different_{(PDA_DFA)}$ sia decidibile.

Pertanto, possiamo concludere che il linguaggio $Different_{(PDA_DFA)}$ è effettivamente indecidibile, utilizzando una riduzione dal problema dell'arresto. Non esiste un algoritmo generale che possa determinare con certezza se un PDA e un DFA riconoscono linguaggi diversi.

Esercizi Bonus

Automi e Linguaggi Formali – Seconda Parte

- Sia $\Gamma = \{0, 1, _, \}\}$ l'alfabeto del nastro per tutte le TM di questo problema. Definire la funzione *busy beaver BB*: $N \rightarrow N$ come segue. Per ogni valore di k , si considerano tutti le TM a k stati che si arrestano quando vengono avviati con un nastro vuoto. Sia $BB(k)$ il numero massimo di 1 che rimangono sul nastro tra tutte queste macchine. Dimostrare che BB non è una funzione computabile.

Per dimostrare che la funzione *busy beaver BB*: $N \rightarrow N$ non è calcolabile, utilizzeremo una prova per contraddizione.

Supponiamo, per amore di contraddizione, che la funzione *busy beaver BB* sia computabile. Ciò significa che esiste una macchina di Turing M_{BB} in grado di calcolare $BB(k)$ per qualsiasi input k .

Costruiamo ora un'altra macchina di Turing, chiamata D , che utilizza M_{BB} come subroutine per risolvere il problema dell'*halting*. Il problema di halting è noto per essere indecidibile, quindi se possiamo risolverlo usando M_{BB} , abbiamo una contraddizione.

Ecco come funziona la macchina di Turing D :

1. Dato un input $\langle M, w \rangle$, dove M è una macchina di Turing e w è una stringa di input.
 - a. Simulare M su w .
 - b. Se M si ferma su w , scrivere un simbolo non vuoto sul nastro.
 - c. Altrimenti, lasciare il nastro vuoto.
2. Costruire una nuova macchina di Turing M' che si comporti come segue:
 - a. Se M si ferma su w , accettare.
 - b. Altrimenti, rifiutare.
3. Eseguire M_{BB} sul numero di stati in M' (k) utilizzando M_{BB} come subroutine.
4. Se M_{BB} restituisce un valore n , dove n è il numero massimo di 1 che rimangono sul nastro tra tutte le macchine di Turing a k stati che si arrestano, procedere come segue:
 - a. Se $n > 0$, significa che M si arresta su w . Accettare.
 - b. Altrimenti, significa che M non si ferma su w . Rifiutare.

Costruendo D che utilizza M_{BB} come subroutine, abbiamo effettivamente risolto il problema dell'*arresto*, che è notoriamente indecidibile. Questo contraddice la nostra ipotesi che BB sia computabile.

Pertanto, possiamo concludere che la funzione *busy beaver BB* non è computabile.

- Dimostrare che A è decidibile se $A \leq_m 0^*1^*$

Per dimostrare che il linguaggio A è decidibile se e solo se $A \leq_m 0^*1^*$, dobbiamo dimostrare entrambe le direzioni dell'affermazione.

In primo luogo, assumiamo che A sia decidibile. Ciò significa che esiste una macchina di Turing M_A che può decidere se una data stringa di input è in A o no.

Per dimostrare che $A \leq_m 0^*1^*$, dobbiamo fornire una riduzione computabile da A a 0^*1^* . Possiamo costruire una macchina di Turing M_R che esegue questa riduzione come segue:

Su input w :

1. Simulare M_A sull'ingresso w .
2. Se M_A accetta w , produce la stringa "0*1*".
3. In caso contrario, viene emessa la stringa vuota.

La riduzione M_R mappa effettivamente gli ingressi in A a "0*1*" e gli ingressi non in A alla stringa vuota.

Consideriamo ora l'altra direzione. Supponiamo che $A \leq_m 0^*1^*$, cioè che esista una riduzione computabile da A a 0^*1^* . Possiamo costruire una macchina di Turing $M_{A'}$ che decide A utilizzando questa riduzione come segue:

Su input w :

1. Utilizzare la riduzione da A a 0^*1^* per ottenere l'output di riduzione per w .
2. Se l'output della riduzione è "0*1*", accettare w .
3. Altrimenti, rifiutare w .

Poiché la riduzione da A a 0^*1^* è computabile, la macchina di Turing $M_{A'}$ può decidere A in base all'output della riduzione.

Pertanto, abbiamo dimostrato entrambe le direzioni dell'affermazione:

- Se A è decidibile, allora $A \leq_m 0^*1^*$.
- Se $A \leq_m 0^*1^*$, allora A è decidibile.

Quindi, possiamo concludere che A è decidibile se e solo se $A \leq_m 0^*1^*$.

- Un automa finito bidimensionale (*2DIM – DFA*) è definito come segue. L'input è un rettangolo $m \times n$, per qualsiasi $m, n \geq 2$. I quadrati lungo il confine del rettangolo contengono il simbolo # e i quadrati interni contengono i simboli dell'alfabeto di ingresso Σ .
 - La funzione di transizione $\delta : Q \times (\Sigma \cup \{\#\}) \rightarrow Q \times \{L, R, U, D\}$ indica lo stato successivo e la nuova posizione della testa (sinistra, destra, su, giù).
 - La macchina accetta quando entra in uno degli stati di accettazione designati. Rifiuta se cerca di spostarsi dal rettangolo di input o se non si ferma mai. Due macchine di questo tipo sono equivalenti se accettano gli stessi rettangoli.
 - Consideriamo il problema di determinare se due di queste macchine sono equivalenti. Formulate questo problema come un linguaggio e dimostrate che è indecidibile.

Per formulare il problema di determinare se due automi finiti bidimensionali (*2DIM-DFA*) sono equivalenti come un linguaggio, possiamo definirlo nel seguente modo:

$L = \{<M_1, M_2> \mid M_1$ e M_2 sono due automi finiti bidimensionali che accettano gli stessi rettangoli di input}

In altre parole, L è il linguaggio che contiene le codifiche delle coppie $<M_1, M_2>$ di due automi finiti bidimensionali che accettano esattamente gli stessi rettangoli di input.

Per dimostrare che questo problema è indecidibile, possiamo ridurlo al problema dell'arresto (halting problem), che è noto per essere indecidibile. Supponiamo di avere una macchina di Turing H che può risolvere il problema dell'arresto.

Costruiamo una nuova macchina di Turing D che utilizza H come sottoprogramma per decidere il nostro problema L . L'idea è di simulare simultaneamente l'esecuzione di M_1 e M_2 su tutti i possibili rettangoli di input, verificando se le due macchine accettano gli stessi rettangoli.

Ecco come funziona la macchina di Turing D:

1. Dato l'input $\langle M_1, M_2 \rangle$, dove M_1 e M_2 sono due automi finiti bidimensionali.
2. Per ogni rettangolo di input R di dimensioni $m \times n$, per m e n maggiori o uguali a 2:
 - a. Simula contemporaneamente M_1 e M_2 su R per un numero finito di passi.
 - b. Se M_1 accetta R e M_2 rifiuta R , o viceversa, rifiuta.
 - c. Se entrambe le simulazioni di M_1 e M_2 si arrestano senza accettare o rifiutare R , passa al prossimo rettangolo di input.
3. Se tutte le simulazioni completano senza rifiutare, accetta.

La macchina di Turing D simula in modo esaustivo l'esecuzione di M_1 e M_2 su tutti i possibili rettangoli di input. Se trova anche un singolo rettangolo su cui M_1 e M_2 accettano o rifiutano in modo diverso, rifiuta. In caso contrario, accetta.

Supponiamo ora che esista una macchina di Turing D' che decide L . Possiamo utilizzare D' per risolvere il problema dell'arresto nel seguente modo:

1. Dato l'input $\langle M, w \rangle$, dove M è una macchina di Turing e w è una stringa di input.
2. Costruisci due automi finiti bidimensionali M_1 e M_2 come segue:
 - M_1 accetta tutti i rettangoli di input.
 - M_2 si comporta come M su w e rifiuta tutti gli altri rettangoli di input.
3. Esegui D' sull'input $\langle M_1, M_2 \rangle$.
4. Se D' accetta, significa che M_1 e M_2 accettano gli stessi rettangoli di input, cioè M si arresta su w . Accetta.
5. Se D' rifiuta, significa che M_1 e M_2 accettano o rifiutano in modo diverso almeno un rettangolo di input, cioè M non si arresta su w . Rifiuta.

Poiché il problema dell'arresto è indecidibile, abbiamo ottenuto una contraddizione, dimostrando che il problema di determinare l'equivalenza di due automi finiti bidimensionali è indecidibile.

Pertanto, possiamo concludere che il problema di determinare se due automi finiti bidimensionali sono equivalenti è indecidibile.

- *Teorema di Rice.* Sia P una qualsiasi proprietà non banale del linguaggio di una macchina di Turing. Dimostrare che il problema di determinare se il linguaggio di una data macchina di Turing ha la proprietà P è indecidibile.
 - In termini più formali, sia P un linguaggio costituito da descrizioni di macchine di Turing in cui P soddisfa due condizioni. In primo luogo, P non è banale: contiene alcune, ma non tutte, le descrizioni delle macchine di Turing. In secondo luogo, P è una proprietà del linguaggio della macchina di Turing, in qualsiasi caso ci sia $L(M_1) = L(M_2)$, si ha $\langle M_1 \rangle \in P$ se e solo se $\langle M_2 \rangle \in P$. Qui, M_1 e M_2 sono TM qualsiasi. TM. Dimostrare che P è un linguaggio indecidibile.

Per dimostrare il teorema di Rice, assumiamo che P sia una proprietà non banale del linguaggio di una macchina di Turing. Mostreremo che il problema di determinare se il linguaggio di una data macchina di Turing ha la proprietà P è indecidibile.

Per prima cosa, definiamo un linguaggio L_P come segue:

$L_P = \{<M> \mid M \text{ è una macchina di Turing e } L(M) \in P\}$

In altre parole, L_P è costituito dalle codifiche delle macchine di Turing il cui linguaggio appartiene alla proprietà P .

Supponiamo ora che esista una macchina di Turing M_P in grado di decidere se il linguaggio di una data macchina di Turing appartiene a P o meno. Useremo M_P per costruire una macchina di Turing M_H che risolva il problema dell'halting, che è noto essere indecidibile. Questo porterà a una contraddizione, dimostrando che P è indecidibile.

Ecco la costruzione di M_H :

Su input $<M, w>$:

1. Costruire una macchina di Turing M' come segue:

- a. Eseguire M su w .
- b. Se M si ferma su w , accettare.
- c. Se M non si arresta su w , eseguire un loop indefinito.

2. Eseguire M_P sulla codifica di M' .

3. Se M_P accetta, accettare.

4. Se M_P rifiuta, rifiutare.

Consideriamo ora i due casi:

Caso 1: $L(M')$ appartiene a P .

- Se M si ferma su w , allora $L(M') = \{w\}$, che appartiene a P .
- Se M non si ferma su w , allora $L(M') = \Sigma^*$, che non appartiene a P .

Pertanto, in questo caso, M_P deve accettare la codifica di M' e M_H deve accettare $<M, w>$.

Caso 2: $L(M')$ non appartiene a P .

- Se M si arresta su w , allora $L(M') = \{w\}$, che non appartiene a P .
- Se M non si ferma su w , allora $L(M') = \Sigma^*$, che non appartiene a P .

Pertanto, in questo caso, M_P dovrebbe rifiutare la codifica di M' e M_H dovrebbe rifiutare $<M, w>$.

Analizziamo ora il comportamento di M_H :

- Se M si ferma su w , M_H accetta $<M, w>$ se $L(M')$ appartiene a P , e rifiuta se $L(M')$ non appartiene a P .
- Se M non si ferma su w , M_H esegue un loop indefinito, indipendentemente dal fatto che $L(M')$ appartenga a P o meno.

Poiché il problema dell'arresto è indecidibile, non esiste una macchina di Turing in grado di risolverlo. Pertanto, la nostra ipotesi che M_P esista deve essere falsa.

Di conseguenza, abbiamo dimostrato che il problema di determinare se il linguaggio di una data macchina di Turing ha la proprietà P è indecidibile, il che dimostra il teorema di Rice.

Quindi, qualsiasi proprietà non banale P del linguaggio di una macchina di Turing è indecidibile.

- REJECT_{TM} è definito come $\{\langle M, w \rangle | M \text{ è una macchina di Turing e } M \text{ rifiuta } w\}$. Dimostrare che REJECT_{TM} è riconoscibile da Turing.

Per dimostrare che REJECT_{TM} è riconoscibile da una macchina di Turing, dobbiamo costruire una macchina di Turing che accetta tutte le coppie $\langle M, w \rangle$ in cui M rifiuta w .

La macchina di Turing che riconosce REJECT_{TM} può essere definita come segue:

Input: $\langle M, w \rangle$

1. Simula la macchina di Turing M sull'input w .

2. Se M rifiuta w , accetta.

La macchina di Turing sopra riconosce correttamente tutti gli input $\langle M, w \rangle$ in cui M rifiuta w . Se la macchina di Turing M effettivamente rifiuta l'input w , la simulazione di M su w terminerà e la macchina di Turing che riconosce REJECT_{TM} accetterà l'input. Se M accetta o entra in un loop infinito su w , la simulazione di M su w non terminerà e la macchina di Turing che riconosce REJECT_{TM} non accetterà l'input.

Pertanto, abbiamo dimostrato che REJECT_{TM} è riconoscibile da una macchina di Turing.

- Una sequenza infinita di numeri naturali $a(1)a(2)a(3)\dots$ si dice strettamente crescente se $a(1) < a(2) < a(3) < \dots$ Sia B l'insieme di tutte le sequenze strettamente crescenti di numeri naturali. Usate la diagonalizzazione per dimostrare che B non è numerabile.

Per dimostrare che l'insieme B di tutte le sequenze strettamente crescenti di numeri naturali non è numerabile, possiamo utilizzare un argomento di diagonalizzazione.

Supponiamo, per amor di contraddizione, che B sia numerabile, il che significa che i suoi elementi possono essere enumerati come b_1, b_2, b_3 e così via.

Ogni elemento b_i in B è una sequenza di numeri naturali, quindi possiamo scriverlo come $b_i = a(i,1)a(i,2)a(i,3)\dots$, dove $a(i,j)$ è l'elemento j della sequenza i .

Ora costruiamo una nuova sequenza $c = c_1c_2c_3\dots$, dove c_i è definito come segue:

$$c_i = a(i,i) + 1$$

In altre parole, ogni elemento c_i di c si ottiene prendendo l'iesimo elemento dell'iesima sequenza di B e aggiungendovi 1.

Ora analizziamo c . Poiché ogni c_i è ottenuto aggiungendo 1 all'elemento corrispondente della sequenza corrispondente in B , c è garantito essere una sequenza di numeri naturali.

Inoltre, possiamo vedere che c differisce da ogni sequenza in B . Per costruzione, per qualsiasi indice i , l'iesimo elemento di c (c_i) non è uguale all'iesimo elemento dell'iesima sequenza in B ($a(i,i)$) perché ci si ottiene aggiungendo 1 ad $a(i,i)$. Pertanto, c non può essere uguale a nessuna sequenza in B .

Questa contraddizione nasce dal fatto che abbiamo assunto che B è numerabile e abbiamo cercato di costruire una sequenza c che non è in B . Pertanto, la nostra ipotesi è falsa.

Pertanto, l'insieme B di tutte le sequenze strettamente crescenti di numeri naturali non è numerabile.