Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteThe Post Correspondence Problem is to determine whether a collection of dominos has a match. This problem is unsolvable by algorithms.

The proof in book shows that the Post Correspondence Problem (PCP) is undecidable by reducing the acceptance problem for Turing machines (ATM) to PCP. The key steps are:

1. From any Turing machine M and input w, construct an instance P of PCP such that P has a match if and only if M accepts w. This establishes that if PCP were decidable, ATM would also be decidable.

2. The constructed PCP instance P simulates the computation of M on w:

- Each domino in P corresponds to a step in the computation

- A match of the dominos represents an accepting computation history of M on w

- The match forces the Turing machine simulation to occur by linking positions in one configuration to the corresponding ones in the next configuration

3. First construct an instance P' of a Modified PCP (MPCP) where the match must start with the first domino. This is done by carefully designing dominos corresponding to the initial config, transitions, copying tape symbols, halting, etc.

4. Then convert the MPCP instance P' to a standard PCP instance P by adding separator symbols (\*) around the original symbols. This forces the match to start with the first domino without explicitly requiring it.

5. If the resulting PCP instance P has a match, it corresponds to an accepting computation history of M on w. Conversely, if M accepts w, the constructed P will have a match mimicking that accepting computation.

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamenteTherefore, since ATM is known to be undecidable, and we reduced it to PCP, PCP must also be undecidable. The key idea is designing the PCP dominos to force a simulation of the Turing machine's computation.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteImmagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamente

The proof describes a reduction from the Domino problem (D1) to the Eulerian Path problem (EULER) in graph theory. The key steps are:

1. Define the Domino problem (D1):

- Input: A set B of domino tiles

- Question: Is there an alignment that uses all the tiles?

2. Reduce D1 to a problem on graphs (EULER) for which a polynomial-time algorithm is known to exist.

3. Define a graph:

- An undirected graph G is a pair (V, E), where:

- V = {v1, v2, ..., vn} is a finite, non-empty set of vertices

- E ⊆ {{u, v} | u, v ∈ V} is a set of unordered pairs, each corresponding to an edge of the graph

4. Construct the domino graph:

- Vertices: The numbers on the domino tiles, V = {①, ②, ③, ④}

- Edges: The domino tiles themselves, E = {①|②, ②|③, ③|④, ②|④, ④|①}

5. Define the Eulerian Path problem (EULER):

- Input: A graph G

- Question: Does G contain an Eulerian path (a path that traverses each edge exactly once)?

6. Note that EULER is a classic graph theory problem with known polynomial-time algorithms, such as Fleury's algorithm.

Fleury's Algorithm:

- Input: An undirected graph G

- Steps:

1. Choose a vertex with odd degree (or any vertex if all degrees are even)

2. Choose an edge such that its removal does not disconnect the graph. If no such edge exists, REJECT.

3. Move to the vertex at the other end of the chosen edge

4. Remove the edge from the graph

5. Repeat from step 2 until all edges are removed

6. If all edges have been removed, ACCEPT

- Complexity: On a graph with n edges, Fleury's algorithm runs in O(n^2) time

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamenteTherefore, the reduction shows that D1 ≤m EULER, and since EULER can be solved in polynomial time (e.g., by Fleury's algorithm), D1 can also be solved in polynomial time using this reduction.

Immagine che contiene diagramma, testo, linea, schizzo

Descrizione generata automaticamenteImmagine che contiene testo, schermata, diagramma, Carattere

Descrizione generata automaticamente

Il discorso è:

* il problema non è risolvibile
* è invece facile *verificare* se la soluzione è corretta

I problemi per i quali esiste un algoritmo polinomiale vengono considerati *trattabili*, mentre quelli che richiedono un algoritmo più che polinomiale sono detti *intrattabili*

Stabilire con precisione qual è il confine tra problemi trattabili ed intrattabili è piuttosto difficile

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Immagine che contiene testo, schermata, Carattere, numero

Descrizione generata automaticamenteA verifier uses additional information to verify that a string w is a member of A. This information is called a *certificate*, or proof, of membership in A.

Immagine che contiene testo, Carattere, linea, diagramma

Descrizione generata automaticamente

Per fornire un verificatore per PebbleDestruction, procediamo come segue:

1. L'input è una tripla (G, p, S) dove:
   * G = (V, E) è un grafo non orientato
   * p è una funzione che assegna un numero di ciottoli a ogni vertice
   * S è una sequenza di mosse
2. Per ogni mossa (v, u) in S:
   * a. Verifica che v abbia almeno 2 ciottoli (altrimenti rifiuta)
   * b. Rimuovi 2 ciottoli da v
   * c. Aggiungi 1 ciottolo a u (se u != v)
3. Dopo aver processato tutte le mosse, verifica che ogni vertice abbia 0 o 1 ciottoli
4. Immagine che contiene testo, schermata, Carattere, linea

   Descrizione generata automaticamenteSe tutte le verifiche hanno successo, accetta. Altrimenti, rifiuta.

*Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente*Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, documento

Descrizione generata automaticamenteImmagine che contiene Line art, disegno, schizzo, illustrazione

Descrizione generata automaticamenteImmagine che contiene testo, Carattere, schermata, diagramma

Descrizione generata automaticamenteImmagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Immagine che contiene testo, diagramma, schermata, Carattere

Descrizione generata automaticamente

.

Immagine che contiene testo, schermata, Parallelo, design

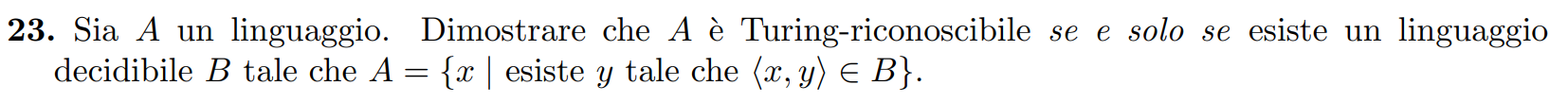
Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamente

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente



Per dimostrare questa affermazione, dobbiamo provare due direzioni:

1. Se A è Turing-riconoscibile, allora esiste un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}.

2. Se esiste un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}, allora A è Turing-riconoscibile.

Direzione 1: Supponiamo che A sia Turing-riconoscibile. Allora esiste una Macchina di Turing M che riconosce A. Costruiamo un linguaggio B nel seguente modo:

B = {<x, y> | M accetta x in al più |y| passi}

Chiaramente, A = {x | esiste y tale che <x, y> ∈ B}, perché se x ∈ A, allora M accetta x in un certo numero di passi, e possiamo prendere y come una codifica binaria di quel numero di passi.

Inoltre, B è decidibile. Ecco una Macchina di Turing M' che decide B:

1. Su input <x, y>, M' simula M su x per al più |y| passi.

2. Se M accetta x entro questi passi, M' accetta <x, y>. Altrimenti, M' rifiuta <x, y>.

Questa simulazione richiede un numero di passi finito (al più |y|), quindi M' decide B.

Direzione 2: Supponiamo che esista un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}. Poiché B è decidibile, esiste una Macchina di Turing M che decide B. Costruiamo una Macchina di Turing M' che riconosce A:

1. Su input x, M' genera sistematicamente tutte le possibili stringhe y (in ordine di lunghezza crescente).

2. Per ogni y generata, M' costruisce la coppia <x, y> e simula M su <x, y>.

3. Se M accetta <x, y>, allora M' accetta x. Se M rifiuta <x, y>, M' passa alla prossima y.

Se x ∈ A, allora esiste una stringa y tale che <x, y> ∈ B, quindi M accetterà <x, y> quando M' la genera, e M' alla fine accetterà x.

Se x ∉ A, allora non esiste alcuna y tale che <x, y> ∈ B, quindi M rifiuterà tutte le coppie <x, y> generate da M', e M' non accetterà mai x.

Quindi M' riconosce A, il che significa che A è Turing-riconoscibile.

Queste due direzioni insieme dimostrano l'affermazione: A è Turing-riconoscibile se e solo se esiste un linguaggio decidibile B tale che A = {x | esiste y tale che <x, y> ∈ B}.

Immagine che contiene testo, schermata, Carattere, informazione

Descrizione generata automaticamenteIn conclusione, X è indecidibile.

(a) La funzione di transizione δ di una ETM è definita come:

δ: Q × (Γ × {F,B}) → Q × (Γ × {F,B}) × {L,R,F}

Dove:

- Q è l'insieme finito degli stati

- Γ è l'alfabeto del nastro che include il simbolo blank (▢)

- {F,B} indica il lato del nastro (F per fronte, B per retro)

- {L,R,F} indica il movimento della testina (L per sinistra, R per destra, F per girare sull'altro lato)

Quindi, δ(q,a,s) = (p,b,t,m) significa che se la ETM è nello stato q, legge il simbolo a sul lato s del nastro, allora passa nello stato p, scrive il simbolo b sul lato t del nastro, e muove la testina secondo m.

(b) Per mostrare che le ETM riconoscono esattamente i linguaggi Turing-riconoscibili, dobbiamo provare due direzioni:

1. Ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

2. Ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM.

Direzione 1: Data una ETM M, possiamo costruire una TM ordinaria M' che simula M. M' usa tre nastri:

- Il primo nastro simula il lato frontale del nastro di M.

- Il secondo nastro simula il lato posteriore del nastro di M.

- Il terzo nastro tiene traccia della posizione e del lato della testina di M.

Ad ogni passo, M' legge lo stato corrente di M dal suo stato, il simbolo dal nastro appropriato (primo o secondo) in base alla posizione e al lato della testina, applica la funzione di transizione di M, scrive il nuovo simbolo sul nastro appropriato, aggiorna lo stato e muove la sua testina sui tre nastri in base al movimento della testina di M.

M' accetta se e solo se M accetta. Quindi, ogni linguaggio riconosciuto da una ETM è Turing-riconoscibile.

Direzione 2: Data una TM ordinaria M, possiamo costruire una ETM M' che simula M. M' usa solo il lato frontale del suo nastro per simulare il nastro di M, e non usa mai il movimento F.

Ad ogni passo, M' legge il suo stato corrente e il simbolo sul lato frontale del nastro, applica la funzione di transizione di M, scrive il nuovo simbolo sul lato frontale e muove la sua testina a sinistra o a destra in base al movimento della testina di M.

M' accetta se e solo se M accetta. Quindi, ogni linguaggio Turing-riconoscibile è riconosciuto da una ETM.

Immagine che contiene testo, Carattere, schermata, algebra

Descrizione generata automaticamenteQueste due direzioni insieme dimostrano che le ETM riconoscono esattamente la classe dei linguaggi Turing-riconoscibili.

(a) Per formulare il problema come un linguaggio, definiamo ENTHUSIASTICTM come:

ENTHUSIASTICTM = {<M> | M è una TM che su input vuoto non scrive mai "332" su tre celle adiacenti del suo nastro}

Dove <M> è la codifica della Macchina di Turing M.

(b) Per dimostrare che ENTHUSIASTICTM è indecidibile, possiamo ridurre da HALTTM, che sappiamo essere indecidibile.

Costruiamo una riduzione f da HALTTM a ENTHUSIASTICTM:

Per una data istanza <M, w> di HALTTM, definiamo f(<M, w>) = <M'>, dove M' è una TM che funziona come segue:

1. M' scrive w sul suo nastro.

2. M' simula M su w.

3. Se M si ferma, M' scrive "332" su tre celle adiacenti del nastro e poi si ferma.

Se <M, w> ∈ HALTTM, allora M si ferma su w, quindi M' scriverà "332" e <M'> ∉ ENTHUSIASTICTM.

Se <M, w> ∉ HALTTM, allora M non si ferma su w, quindi M' non scriverà mai "332" e <M'> ∈ ENTHUSIASTICTM.

Quindi HALTTM ≤m ENTHUSIASTICTM. Poiché HALTTM è indecidibile, anche ENTHUSIASTICTM deve essere indecidibile.

(c) Poiché ENTHUSIASTICTM è il complemento di {<M> | M è una TM che su input vuoto scrive "332" su tre celle adiacenti del suo nastro}, e abbiamo dimostrato che ENTHUSIASTICTM è indecidibile, segue dal Teorema di Rice (per ogni proprietà non banale delle funzioni parziali calcolabili, l'insieme delle Macchine di Turing che calcolano una funzione con quella proprietà è indecidibile) che ENTHUSIASTICTM non è Turing-riconoscibile.

(d) ENTHUSIASTICTM non è Turing-riconoscibile, come dimostrato in (c).

Immagine che contiene testo, schermata, Carattere, documento

Descrizione generata automaticamente

Possiamo formulare il problema AGREEDFA come il linguaggio:

AGREEDFA = {<A, B, w> | A e B sono DFA che entrambi accettano la stringa w}

ii. Per mostrare che AGREEDFA è decidibile, possiamo usare il decider per EQDFA (l'equivalenza di DFA) dal Capitolo 4.

Data un'istanza <A, B, w> di AGREEDFA, costruiamo due nuovi DFA A' e B' come segue:

A' simula A su w. Se A accetta w, A' accetta tutte le stringhe. Altrimenti, A' rifiuta tutte le stringhe.

B' simula B su w. Se B accetta w, B' accetta tutte le stringhe. Altrimenti, B' rifiuta tutte le stringhe.

Ora, <A, B, w> ∈ AGREEDFA se e solo se A' e B' sono equivalenti (cioè, accettano esattamente le stesse stringhe).

Quindi, possiamo decidere AGREEDFA eseguendo il decisore per EQDFA su <A', B'>.

iii. Per mostrare che AGREEDFA è decidibile testando direttamente i DFA su stringhe fino a una certa lunghezza:

Sia n la somma degli stati in A e B. Qualsiasi stringa di lunghezza n o superiore che è accettata da A o B deve fare entrare A o B in un loop (per il pumping lemma).

Quindi, se esiste una stringa accettata sia da A che da B, ce ne deve essere una di lunghezza inferiore a n.

Possiamo quindi decidere AGREEDFA generando tutte le stringhe fino alla lunghezza n-1 e testandole su entrambi i DFA. Se troviamo una stringa accettata da entrambi, accettiamo. Altrimenti, rifiutiamo.

(b)

i. Possiamo formulare REVDFA come:

REVDFA = {<A, B> | A e B sono DFA e L(A) = L(B)^R}

dove L(A) è il linguaggio accettato da A e L(B)^R è il linguaggio formato invertendo ogni stringa in L(B).

Per mostrare che REVDFA è decidibile, data un'istanza <A,B>, costruiamo un DFA B' che inverte ogni stringa accettata da B (scambiando stati finali e iniziali e invertendo le transizioni). Quindi eseguiamo il decider per EQDFA su <A, B'>.

ii. Se provassimo a decidere REVPDA analogamente, usando un decider per l'equivalenza di PDA invece di EQDFA, falliremmo perché non esiste un tale decider. L'equivalenza di PDA è indecidibile.

Intuitivamente, invertire un PDA è molto più complicato che invertire un DFA a causa dello stack. Non possiamo semplicemente scambiare stati finali/iniziali e invertire le transizioni; dovremmo anche invertire in qualche modo le operazioni di stack, che è impossibile in generale.