Macchine di Turing, Algoritmi e Decidibilità

Tutorato 8: TMs, Algoritmi e Decidibilità

Automi e Linguaggi Formali

Gabriel Rovesti

Corso di Laurea in Informatica - Università degli Studi di Padova

Anno Accademico 2024-2025

Contents

1 Le Macchine di Turing e il Loro Ruolo

1.1 Contesto Storico

La nozione intuitiva di algoritmo esiste da migliaia di anni, ma la definizione formale è stata data solo nel XX secolo. Senza una definizione formale, sarebbe stato quasi impossibile dimostrare che un algoritmo non può esistere per risolvere certi problemi.

Nel 1936, Alan Turing pubblicò le specifiche per una "macchina astratta" in grado di definire formalmente gli algoritmi, mentre Alonzo Church pubblicò un formalismo alternativo chiamato λ-calcolo. Nel 1952, Kleene dimostrò che i due modelli sono equivalenti in termini di potere computazionale.

Concetto chiave

La Tesi di Church-Turing afferma che qualsiasi procedimento di calcolo effettivo può essere realizzato mediante una macchina di Turing. Questo collega il concetto informale di "algoritmo" con il concetto formale di "funzione calcolabile da una macchina di Turing".

Un esempio concreto dell'importanza di questa formalizzazione è il decimo problema di Hilbert, che chiedeva di trovare un algoritmo per determinare se un polinomio ha radici intere. Nel 1970, Matiyasevich dimostrò che tale algoritmo non esiste, risultato che sarebbe stato impossibile da dimostrare senza un modello formale di computazione.

1.2 Un Nuovo Modello di Calcolo

Le Macchine di Turing (TM) rappresentano un modello di calcolo più potente rispetto agli automi finiti e agli automi a pila che abbiamo studiato in precedenza.

Concetto chiave

Una macchina di Turing è un modello matematico di calcolo che descrive una macchina astratta che manipola i simboli su una striscia di nastro secondo una tabella di regole. Nonostante la semplicità del modello, è in grado di implementare qualsiasi algoritmo informatico.

Rispetto ai modelli di calcolo precedenti:

- Gli automi finiti sono limitati da una ridotta quantità di memoria
- Gli automi a pila hanno memoria illimitata ma con accesso LIFO (Last-In-First-Out)
- Le macchine di Turing hanno una memoria illimitata e senza restrizioni di accesso

2 Definizione Formale delle Macchine di Turing

2.1 Specifica Formale

Una Macchina di Turing è definita come una tupla: $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ dove:

- Q è l'insieme finito di stati

- Σ è l'alfabeto di input che non contiene il simbolo blank $\sqcup$
- Γ è l'alfabeto del nastro che contiene $\sqcup$ e Σ
- $\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{L, R\}$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $q_{\text{accept}} \in Q$ è lo stato di accettazione
- $q_{\text{reject}} \in Q$ è lo stato di rifiuto (diverso da q_{accept})

2.2 Configurazioni e Computazione

Lo stato corrente, la posizione della testina ed il contenuto del nastro formano la **configurazione** di una TM. Da questa configurazione possiamo determinare la prossima mossa della macchina.

La computazione procede secondo la funzione di transizione δ. Una TM M può comportarsi in tre modi diversi quando riceve un input:

- La macchina **accetta**: raggiunge lo stato q_{accept}
- La macchina **rifiuta**: raggiunge lo stato q_{reject}
- La macchina **va in loop**: non si ferma mai, continuando a calcolare indefinitamente

2.3 Varianti di Macchine di Turing

Esistono diverse varianti delle macchine di Turing che tutte riconoscono la stessa classe di linguaggi:

- **Macchine a nastro semi-infinito**: il nastro è infinito solo verso destra
- **Macchine multinastro**: possiedono k nastri e k testine
- **Macchine non deterministiche**: hanno più scelte possibili ad ogni passo

Teorema

Tutte queste varianti sono equivalenti in termini di potere computazionale, ovvero riconoscono esattamente la stessa classe di linguaggi.

3 Come Descrivere le Macchine di Turing

Esistono tre livelli di descrizione per le macchine di Turing:

- **Descrizione formale**: Dichiara esplicitamente tutto quanto (stati, transizioni, ecc.)
- **Descrizione implementativa**: Descrive a parole il movimento della testina e la scrittura sul nastro, senza dettagli sugli stati

- **Descrizione di alto livello**: Descrive a parole l'algoritmo, senza dettagli implementativi

La descrizione di alto livello è la più utilizzata e segue una notazione standard:

- L'input è sempre una stringa
- Se l'input è un oggetto, deve essere rappresentato come una stringa
- Un oggetto O codificato come stringa è indicato con $\langle O \rangle$
- L'algoritmo viene descritto con un testo, indentato e con struttura a blocchi

4 Linguaggi e Decidibilità

4.1 Classi di Linguaggi

Definizione 1. Un linguaggio è **Turing-riconoscibile** (o ricorsivamente enumerabile) se esiste una macchina di Turing che lo riconosce, ovvero si ferma e accetta tutte e sole le stringhe del linguaggio.

Definizione 2. Un linguaggio è **Turing-decidibile** (o ricorsivo) se esiste una macchina di Turing che lo decide, cioè che termina sempre (accettando o rifiutando) per ogni input.

Un **decisore** è una TM che termina sempre la computazione, senza andare in loop. Accetta o rifiuta ogni stringa in input.

Suggerimento

Tutti i linguaggi Turing-decidibili sono anche Turing-riconoscibili, ma il contrario non è vero. Esistono linguaggi Turing-riconoscibili che non sono decidibili.

5 Problemi Decidibili sui Linguaggi Regolari

Per i linguaggi regolari, esistono diversi problemi che sono decidibili:

5.1 Problema dell'Accettazione

Il problema dell'accettazione consiste nel determinare se un automa a stati finiti accetta una data stringa:

$A_{DFA} = \{\langle B, w \rangle \mid B \text{ è un DFA che accetta la stringa } w\}$

Procedimento di risoluzione

Una TM che decide questo problema può essere costruita come segue:

1. Simula il DFA B sull'input w
2. Se la simulazione termina in uno stato finale, accetta; altrimenti, rifiuta

5.2 Test del Vuoto

Il test del vuoto verifica se un automa a stati finiti accetta almeno una stringa:
$E_{DFA} = \{\langle A \rangle \mid A \text{ è un DFA e } L(A) = \emptyset\}$

Procedimento di risoluzione

Una TM che decide questo problema può essere costruita come segue:

1. Marca lo stato iniziale di A
2. Ripeti: marca ogni stato di A che ha una transizione proveniente da uno stato già marcato
3. Se nessuno degli stati finali è marcato, accetta; altrimenti, rifiuta

5.3 Test di Equivalenza

Il test di equivalenza verifica se due automi a stati finiti riconoscono lo stesso linguaggio:
$EQ_{DFA} = \{\langle A, B \rangle \mid A \text{ e } B \text{ sono DFA e } L(A) = L(B)\}$

Procedimento di risoluzione

Una TM che decide questo problema può essere costruita come segue:

1. Costruisce un DFA C per la differenza simmetrica di A e B: $L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$
2. Esegue il test del vuoto su C
3. Se C accetta il linguaggio vuoto, allora $L(A) = L(B)$

6 Problemi Decidibili sui Linguaggi Context-Free

6.1 Problema dell'Accettazione

Il problema dell'accettazione per grammatiche context-free consiste nel determinare se una grammatica genera una data stringa:
$A_{CFG} = \{\langle G, w \rangle \mid G \text{ è una CFG che genera la stringa } w\}$

Procedimento di risoluzione

Una TM che decide questo problema può essere costruita come segue:

1. Converti G in forma normale di Chomsky
2. Se w è la stringa vuota, accetta se e solo se la stringa vuota è in $L(G)$
3. Altrimenti, enumera tutte le derivazioni di lunghezza $2|w| - 1$ (in forma normale di Chomsky, ogni derivazione di una stringa w ha lunghezza esattamente $2|w| - 1$)
4. Se una di queste derivazioni genera w, accetta; altrimenti, rifiuta

6.2 Test del Vuoto

Il test del vuoto per grammatiche context-free consiste nel determinare se una grammatica genera almeno una stringa:

$E_{CFG} = \{\langle G \rangle \mid G \text{ è una CFG e } L(G) = \emptyset\}$

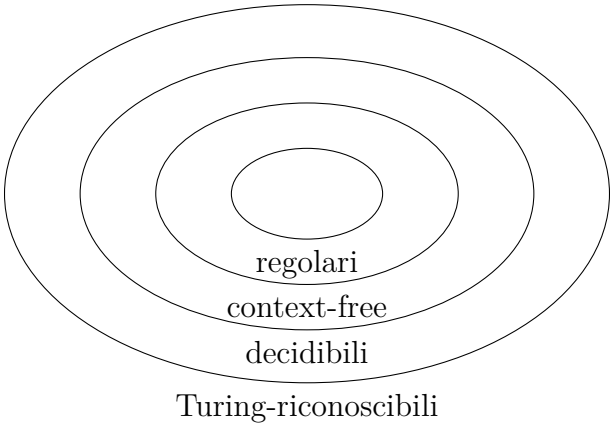
Procedimento di risoluzione

Una TM che decide questo problema può essere costruita come segue:

1. Marca tutti i simboli terminali di G
2. Ripeti: marca ogni variabile A tale che esiste una regola $A \to u_1...u_k$ dove ogni simbolo $u_1...u_k$ è già stato marcato
3. Se la variabile iniziale non è marcata, accetta (il linguaggio è vuoto); altrimenti, rifiuta

7 Relazioni tra Classi di Linguaggi

Le classi di linguaggi formano una gerarchia inclusiva:



Concetto chiave

Queste non sono solo classi di linguaggi, ma anche classi di capacità computazionale:

- I linguaggi regolari sono riconosciuti dagli automi a stati finiti
- I linguaggi context-free sono riconosciuti dagli automi a pila
- I linguaggi decidibili sono decisi dalle macchine di Turing che terminano sempre
- I linguaggi Turing-riconoscibili sono riconosciuti dalle macchine di Turing generali

Teorema

Ogni linguaggio regolare è context-free, ogni linguaggio context-free è decidibile, e ogni linguaggio decidibile è Turing-riconoscibile. Tuttavia, esistono linguaggi Turing-riconoscibili che non sono decidibili, linguaggi decidibili che non sono context-free, e linguaggi context-free che non sono regolari.

8 Algoritmi per Macchine di Turing

Gli algoritmi per macchine di Turing seguono una notazione standard:

- La prima riga dell'algoritmo descrive l'input macchina
- Il corpo dell'algoritmo è descritto con un testo strutturato a blocchi
- L'algoritmo deve terminare per ogni input se vogliamo un decisore

8.1 Esempio: Test di Connessione per Grafi

Un problema classico decidibile è il test di connessione per grafi:
$A = \{\langle G \rangle \mid G \text{ è un grafo connesso}\}$

Procedimento di risoluzione

Una TM che decide questo problema può essere costruita come segue:

1. Marca il primo nodo di G
2. Ripeti: marca ogni nodo di G che è connesso con un arco ad un nodo già marcato
3. Se tutti i nodi sono marcati, accetta; altrimenti, rifiuta

9 Limiti della Decidibilità

Non tutti i problemi sono decidibili. Un esempio notevole è il problema dell'equivalenza per grammatiche context-free:

$EQ_{CFG} = \{\langle G, H\rangle \mid G \text{ e } H \text{ sono CFG e } L(G) = L(H)\}$

Questo problema non è decidibile, a differenza del suo analogo per i linguaggi regolari. La difficoltà nasce dal fatto che i linguaggi context-free non sono chiusi per complementazione e intersezione, operazioni necessarie per costruire la differenza simmetrica.

Concetto chiave

L'esistenza di problemi non decidibili evidenzia i limiti fondamentali della computazione algoritmica e dimostra l'importanza della teoria della calcolabilità per comprendere cosa può e non può essere calcolato in modo effettivo.