Lezione di Informatica Teorica: Macchine di Turing Non Deterministiche e Classi di Computabilità

Appunti da Trascrizione Automatica

30 giugno 2025

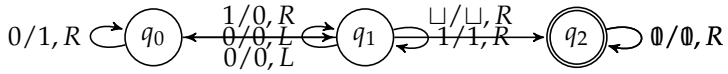
Indice

1 Introduzione alle Macchine di Turing Non Deterministiche

Nella lezione precedente abbiamo esplorato le macchine di Turing multinastro, constatando che, sebbene più pratiche da programmare, sono equivalenti in termini di potenza computazionale alle macchine di Turing mononastro. La simulazione di una macchina multinastro su una mononastro ha un costo polinomiale (quadratico), il che significa che non cambia l'ordine di complessità degli algoritmi. Questo ci permette di utilizzare le macchine multinastro per semplicità, sapendo che il risultato in termini di calcolabilità (e classi di complessità polinomiale) rimane invariato.

Oggi introduciamo un nuovo modello di macchina di Turing: la **Macchina di Turing Non Deterministica (NDTM)**. Questo modello sarà fondamentale per il resto del corso per la sua praticità.

Esempio 1 (Una Macchina di Turing Non Deterministica)**.** *Consideriamo la seguente macchina di Turing (parzialmente descritta):*



Questa macchina presenta una peculiarità nello stato q_1: se legge il simbolo 0*, ha due possibili transizioni:*

1. *Rimanere in q_1, scrivendo* 0 *e spostandosi a sinistra ($q_1 \xrightarrow{0/0,L} q_1$).*
2. *Spostarsi in q_0, scrivendo* 0 *e spostandosi a sinistra ($q_1 \xrightarrow{0/0,L} q_0$).*

In un dato stato e con un dato simbolo letto, la macchina ha più di una scelta per il prossimo passo. Questo comportamento è ciò che definisce il ***non-determinismo****.*

Le macchine viste finora (mononastro, multinastro, multitraccia) erano tutte **deterministich**, ovvero per ogni coppia (stato, simbolo letto) esiste un solo passo successivo possibile.

2 Definizione Formale di Macchina di Turing Non Deterministica

Una **Macchina di Turing Non Deterministica (NDTM)** N è una tupla $N = \langle \Sigma, \Gamma, \sqcup, Q, q_0, F, \delta \rangle$, dove:

- Σ: l'alfabeto di input (simboli che possono essere letti sull'input iniziale del nastro).
- Γ: l'alfabeto di nastro (tutti i simboli che la macchina può maneggiare sul nastro, con $\Sigma \subset \Gamma$).
- $\sqcup$: il simbolo di blank, $\sqcup \in \Gamma \setminus \Sigma$.
- Q: l'insieme finito degli stati della macchina.
- q_0: lo stato iniziale, $q_0 \in Q$.
- F: l'insieme degli stati finali (o accettanti), $F \subseteq Q$.
- δ: la **funzione di transizione** (o relazione di transizione). Per una NDTM, δ mappa una coppia (stato, simbolo letto) a un **insieme** di possibili prossimi passi:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

dove $\mathcal{P}(S)$ denota l'insieme delle parti di S (l'insieme di tutti i possibili sottoinsiemi di S). Questo significa che per una data coppia (q, a), $\delta(q, a)$ restituisce un insieme di triple (q', b, D), dove q' è il nuovo stato, b il simbolo da scrivere, e D la direzione di movimento della testina. La cardinalità di questo insieme può essere maggiore di uno.

Esempio 2. *Per l'esempio precedente, la transizione non deterministica è formalmente descritta come:*

$$\delta(q_1, 0) = \{(q_0, 0, L), (q_1, 0, R)\}$$

Non è necessario che la funzione di transizione sia non deterministica per tutte le coppie (q, a); basta che lo sia per almeno una coppia.

3 Computazione di una Macchina di Turing Non Deterministica

3.1 Configurazioni e Albero di Computazione

Le **configurazioni** (o descrizioni istantanee) di una NDTM sono definite in modo analogo a quelle delle DTM: una stringa che cattura il contenuto corrente del nastro, lo stato della macchina e la posizione della testina. Una configurazione iniziale è q_0W, dove W è la stringa di input. Una configurazione è finale se non ammette configurazioni successive. Una configurazione finale è accettante se lo stato in cui si trova la macchina è uno stato accettante ($q \in F$).

La principale differenza è che una data configurazione può avere più "successori legali". Questo porta a organizzare la sequenza delle configurazioni non più come una lista, ma come un **albero di computazione (Computation Tree)**.

Definizione 1 (Computation Tree)**.** *Il **computation tree** di una macchina di Turing non deterministica M su una stringa di input W è un albero i cui nodi sono tutte le possibili configurazioni in cui la macchina M può trovarsi processando W.*

- *La radice dell'albero è la configurazione iniziale q_0W.*
- *C'è un arco da una configurazione α a una configurazione β se β è uno dei successori legali di α.*

Esempio 3 (Computazione di N su input 01)**.** *Usiamo l'esempio di NDTM di cui sopra e la stringa di input $W = 01$. La configurazione iniziale è $q_0 01$. Analizziamo il processo passo-passo:*

1. *Iniziamo da $q_0 01$.*
2. *Da q_0 leggendo 0: si va a q_0, si scrive 1, si sposta a destra. Nuova configurazione: $1q_0 1$.*
3. *Da q_0 leggendo 1: si va a q_1, si scrive 0, si sposta a destra. Nuova configurazione: $10q_1\sqcup$.*
4. *Da q_1 leggendo $\sqcup$: c'è il non-determinismo (in realtà l'esempio della lezione aveva q_1 che leggeva 0, ma nel grafico è $\sqcup$, seguiamo il grafico per la computazione e l'esempio della $\delta(q_1, 0)$). Il professore ha disegnato la freccia per $(q_1, \sqcup)$ in q_2. Rivediamo l'esempio del disegno, q_1 legge 0, sposta a destra. C'è un arco da q_1 con $1/1, R$ che cicla in q_1. Poi da q_1 con $\sqcup/\sqcup, R$ va a q_2. * **Rivediamo l'esempio del professore basandoci sul disegno e l'input 01:** * Iniziale: $\mathbf{q_0}01$ * $q_0, 0 \to (q_0, 1, R)$: $1\mathbf{q_0}1$ * $q_0, 1 \to (q_1, 0, R)$: $10\mathbf{q_1}\sqcup$ * Ora siamo in $10q_1\sqcup$. La testina è sul blank. Dal diagramma, q_1 leggendo blank porta a q_2 scrivendo blank e spostandosi a destra. * $q_1, \sqcup \to (q_2, \sqcup, R)$: $10\sqcup\mathbf{q_2}\sqcup$ (questa è una configurazione accettante, in quanto $q_2 \in F$). * **Rivediamo invece l'esempio di non-determinismo che il professore ha esplicitamente discusso $\delta(q_1, 0)$:** * Configurazione $q_0 01$ * $q_0, 0 \to (q_0, 1, R)$: $1q_0 1$*

* $q_0, 1 \to (q_1, 0, R)$*: $10q_1\sqcup$ (Questo è il punto di divergenza tra il diagramma mostrato e l'input* 01 *effettivo, in quanto dopo* $10q_1\sqcup$ *la testina è sul blank, non sul* 0 *per attivare il non-determinismo).* * ***Per allineare l'esempio alla discussione del professore, dobbiamo immaginare un input diverso che porti a*** $\mathbf{q_1}0$*...: Il professore usa* `10q10` *nell'esempio sull'albero, non* `10q1_blank_`*. Questo implica che l'input doveva essere più lungo.* * **Seguiamo l'esempio disegnato dal professore, che mostra la configurazione $10q_10$ come punto di non-determinismo**: * Siamo in $10\mathbf{q_1}0$. Testina su 0, stato q_1. * $\delta(q_1, 0) = \{(q_0, 0, L), (q_1, 0, R)\}$ * ***Primo branch:*** $(q_0, 0, L)$*. Scriviamo* 0*, spostiamo a sinistra, andiamo in* q_0*. Configurazione successiva:* $1\mathbf{q_0}00$*. Da* $1\mathbf{q_0}00$*:* $q_0, 0 \to (q_0, 1, R)$*. Scriviamo* 1*, spostiamo a destra.* $11\mathbf{q_0}0$*. Da* $11\mathbf{q_0}0$*:* $q_0, 0 \to (q_0, 1, R)$*. Scriviamo* 1*, spostiamo a destra.* $111\mathbf{q_0}\sqcup$*. Da* $111\mathbf{q_0}\sqcup$*:* $q_0, \sqcup \to (q_2, \sqcup, R)$*. (Non specificato nel disegno iniziale, assumiamo). Se non c'è transizione, si blocca e rifiuta. Nell'esempio disegnato, questo ramo è bloccato e non accettante (X).* * ***Secondo branch:*** $(q_1, 0, R)$*. Scriviamo* 0*, spostiamo a destra, andiamo in* q_1*. Configurazione successiva:* $100\mathbf{q_1}\sqcup$*. Da* $100\mathbf{q_1}\sqcup$*:* $q_1, \sqcup \to (q_2, \sqcup, R)$*. Scriviamo* $\sqcup$*, spostiamo a destra. Configurazione successiva:* $100\sqcup\mathbf{q_2}\sqcup$*. (Questa è accettante,* $q_2 \in F$*).* * *L'esempio nel disegno ha un ramo ulteriore:* $10\mathbf{q_1}0 \to 10\mathbf{q_1}0$ *(ciclo su* $0/0, L$*)* $\to \ldots$*. Questo dimostra che possono esserci più rami e che un ramo può anche andare in loop.*

3.2 Condizione di Accettazione per NDTM

Una macchina di Turing non deterministica M **accetta** un input W se e solo se all'interno del computation tree di M su W **esiste almeno una configurazione accettante**.

Ciò significa che la macchina non deve trovare *tutti* i cammini accettanti, ne basta uno. Se la macchina ha un modo per accettare, allora accetta. Perché una NDTM **rifiuti** un input, deve succedere che *tutte* le computazioni all'interno del computation tree o terminano in una configurazione non accettante, o non terminano mai (loop).

4 Simulazione di NDTM tramite DTM

Una domanda fondamentale è: le Macchine di Turing Non Deterministiche sono più potenti delle Macchine di Turing Deterministiche (DTM) in termini di capacità di calcolo? Ovvero, possono calcolare funzioni o accettare linguaggi che le DTM non possono? La risposta è **No**.

Teorema 1. *Per ogni linguaggio L accettato da una macchina di Turing non deterministica N, esiste una macchina di Turing deterministica M che accetta L.*

Questo teorema è cruciale, perché significa che, anche se le NDTM non sono fisicamente realizzabili (non "sanno" quale ramo scegliere o non si "sdoppiano"), possiamo usarle come modello di calcolo astratto perché tutto ciò che possono fare può essere fatto anche da una DTM.

4.1 Strategia di Simulazione (BFS)

Per dimostrare il teorema, è necessario mostrare come una DTM possa simulare una NDTM. La strategia comune è una **ricerca in ampiezza (Breadth-First Search - BFS)** sull'albero di computazione della NDTM.

Supponiamo di avere una NDTM N (che possiamo assumere mononastro, poiché sappiamo convertire multinastro in mononastro). Vogliamo costruire una DTM M (multinastro per facilità, poi riconvertibile a mononastro) che simuli N.

La macchina M utilizzerà più nastri. Una configurazione tipica per la simulazione è l'uso di tre nastri:

1. **Nastro 1 (Input/Originale)**: Contiene l'input originale W e non viene modificato.
2. **Nastro 2 (Simulazione)**: Contiene la configurazione corrente di N che M sta simulando.
3. **Nastro 3 (Coda delle configurazioni)**: Contiene una coda di configurazioni di N che devono ancora essere esplorate, separate da un simbolo speciale (es. $*$).

Algoritmo di Simulazione (BFS):

1. M inizializza il Nastro 3 scrivendo la configurazione iniziale di N su W (q_0W).
2. M entra in un ciclo di esplorazione:
   (a) Prende la prima configurazione ID_k dal Nastro 3. Se il Nastro 3 è vuoto, M rifiuta (non ha trovato alcun percorso accettante) e si ferma.
   (b) Copia ID_k sul Nastro 2.
   (c) M simula un singolo passo della NDTM N a partire da ID_k (Nastro 2).
   - Se ID_k è una configurazione finale accettante (lo stato è in F), M accetta W e si ferma.
   - Se ID_k è una configurazione finale non accettante (si blocca o lo stato non è in F), M scarta questo ramo e continua con il passo (a).
   - Se ID_k ammette k' successori legali (data la δ di N, M sa quanti e quali sono), M genera questi k' nuovi ID e li aggiunge in coda al Nastro 3, separati da asterischi o altri delimitatori.

Questa strategia garantisce che M accetti W se e solo se N accetta W. La BFS è cruciale perché impedisce a M di bloccarsi in un ramo infinito non accettante, garantendo che se un cammino accettante esiste, M lo troverà prima o poi (a meno che l'albero intero non sia infinito e senza cammini accettanti).

4.2 Costo della Simulazione

Il costo di questa simulazione è **esponenziale**. Se la NDTM N compie K passi per accettare (ovvero il cammino accettante più breve ha lunghezza K), e il fattore di branching massimo della NDTM è C (cioè, ogni configurazione può avere al massimo C successori), allora:

- Al livello 0: 1 configurazione.
- Al livello 1: C configurazioni.
- Al livello 2: C^2 configurazioni.
- Al livello K: C^K configurazioni.

La DTM M deve esplorare tutti i nodi fino al livello K per trovare il cammino accettante (nella BFS), o tutti i nodi fino a un certo punto per esaurire le opzioni. Il numero di configurazioni da esplorare cresce esponenzialmente con la lunghezza del cammino più breve (K). Quindi, il tempo di esecuzione di M è $O(C^K)$. Questo significa che c'è un **gap esponenziale** tra la velocità di una NDTM e quella di una DTM che la simula. Se un problema può essere risolto da una NDTM in tempo polinomiale, la DTM che lo simula potrebbe richiedere tempo esponenziale. Questo è il cuore del famoso problema **P vs NP**.

Non si sa se esista una simulazione più efficiente (es. polinomiale) delle NDTM su DTM. Nessuno è riuscito a trovarla, né a dimostrare che non esista.

5 La Tesi di Church-Turing

Abbiamo esaminato vari modelli di calcolo: DTM mononastro, DTM multitraccia, DTM multinastro, NDTM. Tutti questi modelli, pur variando in efficienza, hanno la stessa potenza computazionale: possono calcolare lo stesso insieme di funzioni e accettare lo stesso insieme di linguaggi.

Negli anni '30, quando Alan Turing e altri svilupparono i loro modelli di calcolo (es. λ-calcolo di Alonzo Church, sistemi di Post), si scoprì che tutti i modelli conosciuti erano computazionalmente equivalenti alle Macchine di Turing. Questo portò alla formulazione della **Tesi di Church-Turing**:

Teorema 2 (Tesi di Church-Turing)**.** Tutto ciò che è calcolabile è calcolabile da una macchina di Turing.

È chiamata "tesi" e non "teorema" perché non è una dimostrazione formale, ma un'affermazione riguardante la natura della computazione. Non si definisce la calcolabilità a priori e poi si dimostra che la MT la raggiunge, ma si propone che la MT catturi il concetto intuitivo di calcolabilità. Ad oggi, nessun modello di calcolo più potente è stato scoperto o inventato, il che rafforza la fiducia in questa tesi.

6 Classi di Computabilità

Sulla base della Tesi di Church-Turing, possiamo definire le principali classi di problemi (o linguaggi) in base alla loro calcolabilità da parte di una Macchina di Turing.

6.1 Classe R (Linguaggi Ricorsivi / Problemi Decidibili)

Definizione 2 (Classe R)**.** *La classe R (linguaggi **ricorsivi**) contiene tutti i linguaggi L per i quali esiste una Macchina di Turing M che **decide** L.*

Una Macchina di Turing M **decide** un linguaggio L se per ogni input $W \in \Sigma^*$:

- Se $W \in L$, M si arresta in uno stato accettante.
- Se $W \notin L$, M si arresta in uno stato non accettante.

In altre parole, una macchina che decide un linguaggio **termina sempre** per ogni input, dando una risposta definitiva (sì o no). I problemi che corrispondono ai linguaggi in R sono chiamati **problemi decidibili**.

6.2 Classe RE (Linguaggi Ricorsivamente Enumerabili / Problemi Accettabili)

Definizione 3 (Classe RE)**.** *La classe RE (linguaggi **ricorsivamente enumerabili**) contiene tutti i linguaggi L per i quali esiste una Macchina di Turing M che **accetta** L.*

Una Macchina di Turing M **accetta** un linguaggio L se per ogni input $W \in \Sigma^*$:

- Se $W \in L$, M si arresta in uno stato accettante.
- Se $W \notin L$, M *potrebbe non arrestarsi mai* o arrestarsi in uno stato non accettante.

I problemi che corrispondono ai linguaggi in RE sono chiamati **problemi accettabili**.

6.3 Relazione tra R e RE

È chiaro dalla definizione che ogni linguaggio che può essere deciso può anche essere accettato (una macchina che termina sempre e dà una risposta è anche una macchina che accetta). Quindi, $R \subseteq RE$. In realtà, $R \subset RE$, ovvero esistono linguaggi che sono accettabili ma non decidibili. Questi problemi sono chiamati **problemi indecidibili**. A volte, i linguaggi in $RE \setminus R$ (cioè, i linguaggi accettabili ma non decidibili) sono chiamati **semidecidibili**. Per questi problemi, se la risposta è "sì", l'algoritmo si ferma e lo comunica. Ma se la risposta è "no", l'algoritmo potrebbe non fermarsi mai, lasciandoci nell'incertezza sulla risposta. Questo li rende problematici per l'uso pratico.

Con queste definizioni, abbiamo stabilito un quadro per classificare i problemi in base alla loro calcolabilità da parte delle Macchine di Turing.