4. VARIANTI DI MACCHINE DI TURING

Esistono definizioni alternative di Macchina di Turing, chiamate *varianti*. Tra queste, vedremo delle MdT a più nastri e MdT non deterministiche. Mostreremo, inoltre, che tutte le varianti "ragionevoli" hanno la stessa capacità computazionale.
Quando proviamo che una MdT ha una certa proprietà, non ci poniamo domande sulla grandezza di tale macchina o su quanto è complesso programmarla. Per il momento, ci interessano solo alcune proprietà essenziali.
Uno **stayer** è una Macchina di Turing la cui testina può rimanere sulla stessa cella del nastro durante una transizione; formalmente, essa è la stessa settupla $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, con la sola differenza che la funzione di transizione è $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, S, R\}$, dove **S** indica che la testina resta ferma durante la transizione (S sta per "stay").
Diciamo che il potere computazionale di due modelli è lo stesso se riconoscono la stessa classe di linguaggi.
Una MdT può essere facilmente simulata da uno stayer (basta banalmente non usare la possibilità di stare nella funzione di transizione), per cui lo stayer include, per definizione, una MdT semplice. Da ciò consegue che il potere computazionale di uno stayer è almeno pari al potere computazionale di una MdT convenzionale. Resta da provare il seguente.

Corollario:

> Il potere computazionale di una MdT è almeno pari al potere computazionale di uno stayer ⇔ per ogni stayer esiste una MdT che riconosce lo stesso linguaggio.

Dimostrazione:
Assumiamo che M sia uno ***stayer***, e mostriamo un MdT M' che simula M.
M' è definita esattamente come M, tranne che ogni transizione S è sostituita da due transizioni in M':

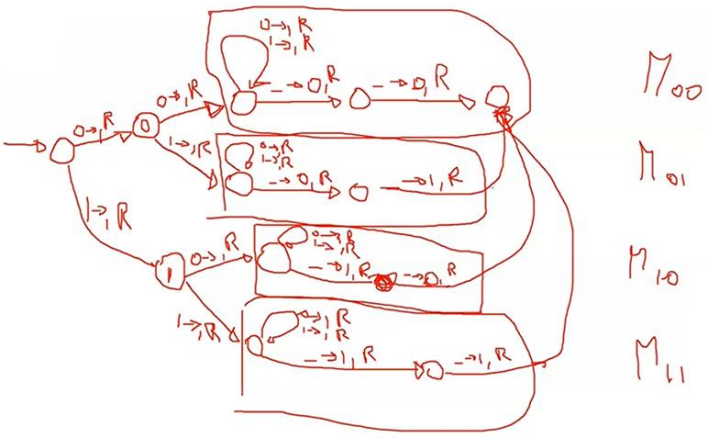
1. la prima porta M' in uno stato speciale (addizionale) e muove la testina a destra (R);
2. la seconda ritorna allo stato originale muovendo la testina a sinistra (L).

Quindi, M e M' riconoscono lo stesso linguaggio.

In generale, quando vogliamo provare che due varianti di MdT hanno lo stesso potere computazionale, mostriamo che possiamo simulare una con l'altra.
In alcune occasioni avremo bisogno di **memorizzare** l'informazione letta sul natro: *possiamo usare gli stati per memorizzare informazioni*. L'approccio è simile a quello visto nell'esempio sulle MdT convenzionali, dove per memorizzare il bit 0 andavamo in una parte della macchina, mentre per memorizzare il bit 1 andavamo in un'altra parte della macchina.

Esempio:
Supponiamo di avere una MdT M che deve leggere i primi due bit letti sul nastro e scriverli alla fine dell'input (al posto dei due _ più a sinistra). Un'idea può essere la seguente:
- costruiamo una MdT M_{00} che legge i primi due bit, cerca la fine dell'input e scrive 00 alla fine dell'input;
- replichiamo il passo precedente per tutte le possibili coppie (in questo caso M_{01}, M_{10} e M_{11});
- M: dopo aver letto i primi due bit in input, si sposta sulla replica corrispondente ai due bit letti (e quindi li scrive alla fine dell'input).

Notiamo che in base ai primi due bit in input si va in una determinata "sottomacchina".
Poiché i primi due bit, sull'alfabeto {0, 1}, possono dare 2^2 combinazioni diverse, si costruiscono 4 diverse parti di macchina che operano in dipendenza dei bit letti.



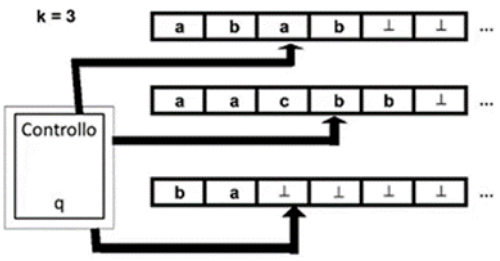
Nota: Questa tecnica è utilizzabile *in generale*.
Se vogliamo che M memorizzi una sequenza di k simboli, possiamo:
- avere una replica per ogni possibile sequenza di k simboli;
- M si sposta sulla replica che corrisponde alla sequenza mentre la legge.

Questo implica che sono necessari circa $|\Sigma|^k$ stati aggiuntivi. Infatti, se $|\Sigma|^k = c$, allora avremo c (stati dopo aver letto il primo simbolo) + c^2 (per ogni stato precedente, ci sono altri c stati) + … + $c^k = c + c^2 + … + c^k = O(c^k)$.

4.1 MACCHINA DI TURING MULTI-NASTRO

Una MdT **multinastro** (a k nastri) è una normale Macchina di Turing avente k nastri. Inizialmente, l'input compare sul primo nastro e gli altri sono vuoti. La sua funzione di transizione è $\boldsymbol{\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, S, R\}^k}$; dunque, essa muove (in modo indipendente) le testine dei vari nastri.
Questa variante è simile ad una MdT convenzionale: usa k nastri, con $k \geq 1$. Ciascun nastro ha una propria testina, e con una mossa si specificano (oltre al nuovo stato): i k simboli letti, i k simboli da scrivere e i k movimenti delle k testine. Come configurazione iniziale, essa avrà l'input sul primo nastro ed i rimanenti nastri saranno vuoti.

Nella figura a destra, ad esempio, abbiamo una MdT a 3 nastri, dove in quel momento lo stato occupato dalla macchina è q, la prima testina punta alla cella 2 (a), la seconda testina punta alla cella 3 (b), la terza testina punta alla cella 2 (_). In questo caso, quindi, la funzione di transizione δ(q, (a, b, _)) ci restituirà un nuovo stato, tre simboli che saranno scritti in quelle celle, e i movimenti delle tre testine.
In maniera compatta, usiamo l'espressione $\delta(q_i, a_1, …, a_k) = (q_j, b_1, …, b_k, D_1, …, D_k)$ per indicare che se la MdT a k nastri M si trova nello stato q_i, e la testina i legge a_i (per i = 1, …, k), allora M va nello stato q_j, la testina scrive b_i e si muove nella direzione $D_i \in \{L, S, R\}$ (per i = 1, …, k).



Esempio:
Premessa: spesso, si userà il simbolo ⍰ per identificare il blank (_).
Questa macchina MdT a 2 nastri per 0^n1^n, la cui funzione di transizione è descritta dalla tabella a lato:
1. Scorre il primo nastro verso destra fino al primo 1: per ogni 0, scrive un 1 sul secondo nastro;
2. Scorre il primo nastro verso destra e il secondo nastro verso sinistra: se i simboli letti sono diversi, allora termina in uno stato non finale;
3. Se legge _ su entrambi i nastri, allora termina in uno stato finale.

stato	simboli	stato	simboli	movimenti
q0	$(0, \perp)$	q0	$(\perp, 1)$	(R, R)
q0	$(1, \perp)$	q1	$(1, \perp)$	(S, L)
q1	$(1, 1)$	q1	$(\perp, \perp)$	(R, L)
q1	$(\perp, \perp)$	q2	$(\perp, \perp)$	(S, S)

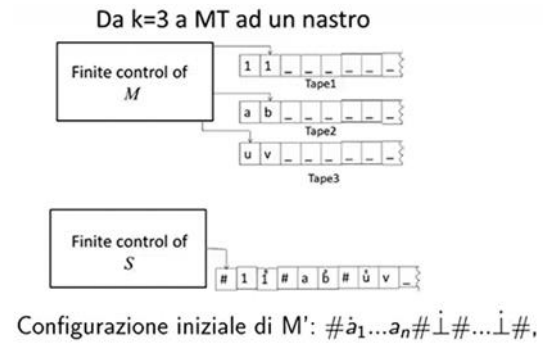
Nota: Le MdT multinastro sembrerebbero più potenti delle MdT ordinarie. In realtà, possiamo dimostrare che le due varianti sono equivalenti.

Teorema:

MdT e MdT multinastro sono modelli equivalenti.

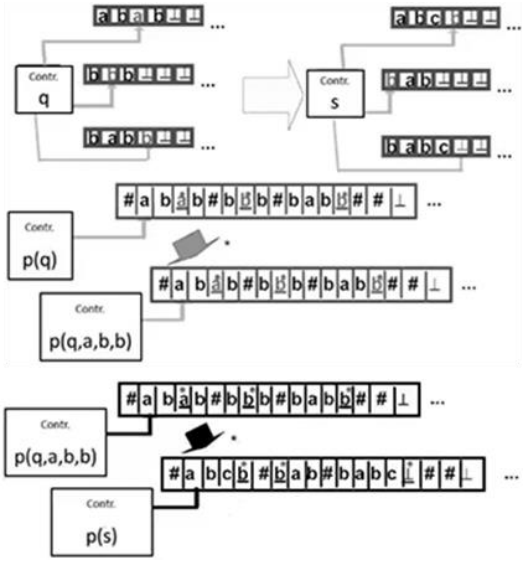
Dimostrazione:
L'implicazione diretta è ovvia, in quanto una MdT è una MdT multinastro, con k = 1. Vogliamo dimostrare che per ogni MdT multinastro è possibile costruire una MdT convenzionale che riconosce lo stesso linguaggio. A tal scopo, effettuiamo una simulazione. Utilizziamo come esempio la MdT multinastro, con k = 3, definita in precedenza.
Una soluzione è immaginare i k nastri affiancati, ognuno con indicata la posizione della testina. Dobbiamo codificare questa informazione su un solo nastro. Per fare ciò, quindi, possiamo concatenare il contenuto dei k nastri, su k blocchi consecutivi separati da un carattere particolare (in questo esempio, #):
- ogni blocco avrà lunghezza variabile che dipende dal contenuto del nastro corrispondente;
- un elemento marcato (con ˙) nel blocco i-esimo indica la posizione della testina i-esima (ad esempio, se la testina S punta ad un elemento del primo blocco e legge γ˙, allora la testina del primo nastro è in questa posizione e legge γ);
- usiamo un alfabeto esteso Γ_2 tale che γ˙ $\in \Gamma_2$ per ogni $\gamma \in \Gamma$.

Nella figura a destra, notiamo che il primo nastro (input 11) punta alla cella 1 (1), il secondo nastro (input ab) punta alla cella 1 (b), il terzo nastro (input uv) punta alla cella 0 (u). Sotto è posta la macchina M' convenzionale.



Per ogni istruzione della MdT multinastro M, la MdT M':
1. scorre i k nastri e "raccoglie" informazioni sui k simboli letti;
2. applica la transizione, scorrendo i k nastri e applicando su ciascuno scrittura e spostamento.
3. infine la MdT entra nello stato che ricorda il nuovo stato di S e riposiziona la testina all'inizio del nastro.
Questo comporta molti più stati e mosse, ma otteniamo comunque che S simula M.

A lato è posto un esempio, dove si applica la transizione δ(q, a, b, b) = (s, c, a, c, R, L, R).
Nella macchina ad un nastro, prima legge il contenuto dei nastri (quindi, sa che sul primo nastro viene letta a, sul secondo viene letta b, sul terzo viene letta b); ora, sa che la la mossa è δ(q, a, b, b) = (s, c, a, c, R, L, R), per cui si dovrà scrivere:
nastro 1) c al posto di a˙, per poi spostarsi a destra (quindi, si dovrà sostituire la b in cella 4 con una b˙);
nastro 2) a al posto di b˙, per poi spostarsi a sinistra (quindi, si dovrà sostituire la b in cella 6 con una b˙);
nastro 3) c al posto di b˙, per poi spostarsi a destra (quindi, si dovrà sostituire il # in qualche modo).
Nella parte finale della figura notiamo il cambiamento: nel nastro 3, abbiamo shiftato a destra tutti i caratteri dopo la c, ed inserito un ⍰˙ per indicare che quella è la fine del nastro. Questo è possibile, dato che la lunghezza del nastro di una MdT è variabile nel tempo.
Per ogni mossa del tipo $\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, D_1, \ldots, D_k)$:
- S scorre il nastro verso destra, fino al primo ⍰, memorizzando nello stato i simboli marcati sui singoli nastri. Si ricordi che la memorizzazione viene implementata con uno stato aggiuntivo per ogni possibile sequenza di ≤ k simboli di Γ;
- la testina si riposiziona all'inizio del nastro;
- poi il nastro viene scorso di nuovo, eseguendo su ogni sezione (cioè un nastro di M) le azioni che simulano quelle delle testine di M, ossia scrittura e spostamento.
Durante il secondo passaggio, S scrive su tutti i simboli "marcati", e sposta il marcatore alla nuova posizione della testina corrispondente di M.

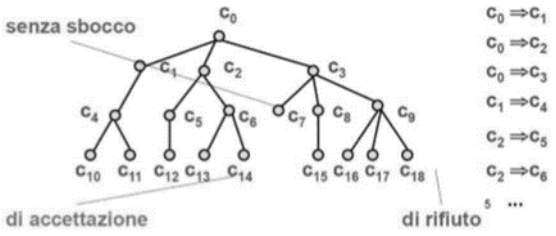


Nota: Se una testina di M muove su ⍰ più a sinistra del suo nastro, allora la testina virtuale (il puntino su un simbolo) sul segmento corrispondente del nastro di S si sposta sul delimitatore #. In questo caso, S deve spostare di una posizione tutto il suffisso del nastro a partire da #˙ fino all'ultimo # a destra. Dopo, S scrive ⍰˙ nella prima posizione soggetta a shift (cioè, dove c'era #˙).

4.2 MACCHINA DI TURING NON DETERMINISTICA

Una MdT **non deterministica** (NMdT) è una macchina di Turing convenzionale avente funzione di transizione $\delta: Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$, dove P è l'*insieme potenza* di tale prodotto cartesiano, invece di $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$; il non determinismo, anche in questo caso, permette di avere associato ad uno stato e ad un simbolo letto sul nastro più transizioni: per ogni transizione, la computazione proseguirà autonomamente in parallelo.
La computazione di una NMdT è ben descritta da un albero contenente ogni configurazione raggiungibile dalla iniziale (c_0) su un dato input: se un cammino (da c_0) raggiunge lo stato accetta, allora la NMdT accetta l'input, anche se altri cammini raggiungono uno stato reject.
Siccome l'insieme potenza contiene anche il vuoto, è possibile che una computazione si fermi a seguito di input incorretto: nella figura a lato, ad

esempio, c_7 non produce alcuna nuova configurazione.
In questo caso, la macchina si ferma dopo tre passi: avendo raggiunto una configurazione di accettazione (c_{14}), la stringa in input verrà accettata, indipendentemente da ciò che accade sugli altri cammini.



Le NMdT sembrerebbero più potenti delle MdT ordinarie. In realtà, possiamo dimostrare che le due varianti sono equivalenti.

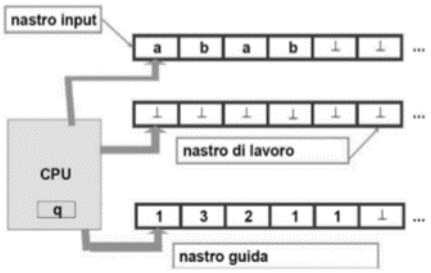
Teorema:

Per ogni ***NMdT*** esiste una ***MdT deterministica equivalente***.

Dimostrazione:
In un verso è ovvio, in quanto una MdT è anche una NMdT, per la definizione stessa.
Dimostriamo, ora, che se abbiamo un linguaggio riconosciuto da una NMdT allora esso è riconosciuto anche da una MdT. A tal scopo, guardiamo alla computazione di una NMdT N come a delle configurazioni raggiungibili da quella iniziale c_0 su input w: esaminiamo l'albero della figura precedente, e simuliamo questo comportamento seguendo l'albero delle configurazioni, eseguendo tutte le computazioni.
Visitiamo l'albero con strategia *BFS* – Breadth-First Search (cioè, *ricerca in ampiezza*) – ed eseguiamo un ramo per volta fin quando non lo esaminiamo per intero. Partendo dalla configurazione iniziale c_0 andiamo in c_1, c_2 e c_3; al secondo step, torniamo al primo nodo e visitiamo tutti i nodi che esso produce (c_4), torniamo al secondo nodo e visitiamo tutti i nodi che esso produce (c_5 e c_6), torniamo al terzo nodo e visitiamo tutti i nodi che esso produce (c_7, c_8, c_9); proseguiamo in questo modo fin quando non raggiungiamo una configurazione di accettazione (o di rifiuto).
Formalmente, per ogni input w:
- M deve eseguire tutte le possibili computazioni di N su w;
- M deve accettare se e solo se almeno una raggiunge lo stato accetta.

Utilizziamo una BFS dell'albero delle computazioni (eseguendo tutte le simulazioni in contemporanea), in quanto alcune delle computazioni di N possono essere infinite, e ciò implica che:
- alcuni cammini sono infiniti;
- se M esegue la simulazione e segue un cammino infinito, va in loop (anche se su un altro cammino raggiungerebbe lo stato accetta).

L'algoritmo specificato in precedenza è formalizzato come segue:
1. Esegui il primo passo di ogni computazione. Se almeno una accetta, allora accetta;
2. Esegui il secondo passo di ogni computazione. Se almeno una accetta, allora accetta;
…
i. Esegui il passo i-esimo di ogni computazione. Se almeno una accetta, allora accetta.



Questa simulazione la facciamo mediante una MdT D a 3 nastri. Inizialmente, il nastro 1 contiene l'input di N e gli altri due sono vuoti; la simulazione di N procede come segue:
1. Copia l'input sul nastro 2 (usato per eseguire la computazione);
2. Esegui un prefisso di questa computazione, usando il contenuto del nastro 3. Se si raggiunge una configurazione accetta, allora accetta l'input;
3. Aggiorna il contenuto del nastro 3 per ottenere la successiva computazione (si incrementa il contenuto di 1, fin quando non si arriva al numero b, dato che la cifra è b-aria);
4. Vai al passo 1.

Per gestire il contenuto del nastro 3, formalmente, sappiamo che N ha più possibili transizioni da una stessa configurazione:
- per ogni configurazione di N, la MdT D codifica tutte le possibili transizioni e le enumera;
- sia $COMP_i$ il prefisso lungo *i* di una computazione di N (dove *i* rappresenta il numero di passi da eseguire). Codifichiamo $COMP_i$ con stringa $b_1 \dots b_i$ di *i* simboli, dove b è il massimo numero di transizioni possibili da qualsiasi configurazione. Allora:
  - D parte con la configurazione iniziale;
  - b_1 è il numero della transizione al primo passo di $COMP_i$;
  - per $2 \le j \le i$, b_j è il numero della transizione attuale al passo j-esimo di $COMP_i$.

Ad esempio, la stringa 421 codifica un prefisso di lunghezza 3 (quindi, b = 3) in cui:
- al primo passo, N esegue la quarta transizione nella lista delle transizioni possibili (dalla configurazione iniziale);
- al secondo passo, N esegue la seconda transizione nella lista delle transizioni possibili (dalla configurazione raggiunta al passo 1);
- al terzo passo, N esegue la prima transizione nella lista delle transizioni possibili (dalla configurazione raggiunta al passo 2).

Nota. Alcune configurazioni possono ammettere meno di b scelte, quindi non tutte le sequenze b-arie rappresentano un prefisso di computazione.
Riassumendo, la simulazione di N con D risulta:
1. Scrivi 1 sul nastro 3;
2. Copia input sul nastro 2;
3. Se il prefisso del numero b-ario è la codifica di un prefisso di una computazione di N, allora eseguilo. Se la computazione accetta, allora accetta, altrimenti vai al passo 4;
4. Incrementa di 1 il numero b-ario sul nastro 3 (con i numeri che vanno da 1 a b);
5. Vai al passo 2.

Non abbiamo detto *come* funziona, in quanto la macchina deterministica diventa molto complicata da costruire. Non diamo maggiori dettagli, ma ci limitiamo a dire come dovrebbe funzionare: questo perché a noi basta sapere che una tale macchina esiste.
Si è dimostrato che questi tre modelli di calcolo sono tra loro equivalenti. A noi basta conoscere l'equivalenza tra MdT e calcolatori (programmi o algoritmi).

Tesi di Church-Turing

Se esiste un algoritmo per eseguire un calcolo, allora questo calcolo può essere eseguito da una MdT (o equivalenti).

Da questa Tesi ricaviamo la sua negazione: infatti, ciò è equivalente a dire che se non esiste una MdT allora non esiste un algoritmo. In seguito, vedremo, basandoci su questo fatto, che esistono problemi per cui non è possibile trovare un algoritmo che risolve quel problema: cioè, esistono problemi che *non sono computabili*.