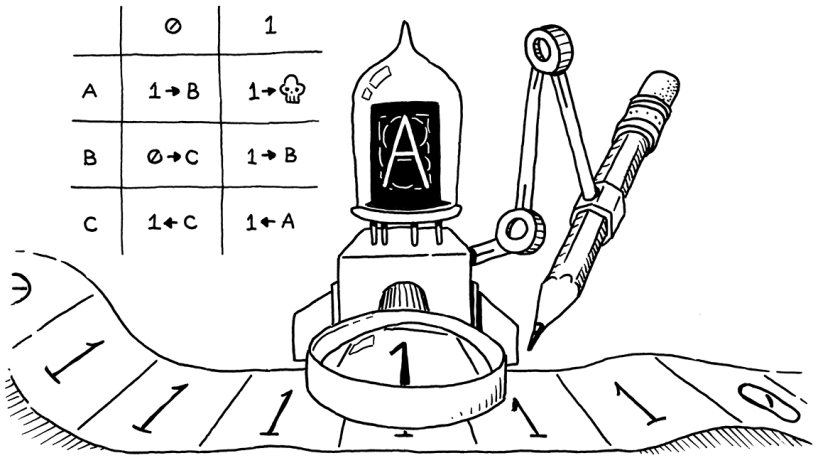
UNIVERSITÀ DEGLI STUDI DI TRIESTE

Marco Sgobino

Dispense del corso di

Complessità e Crittografia



Anno Accademico 2021-2022

Questo documento è stato prodotto in LaTeX. Il codice sorgente è reperibile al seguente indirizzo,

https://gitlab.com/sgob/compl-repo

Un grazie va a *Matthew Butterick*, ed ai suoi preziosissimi consigli sull'uso corretto (e responsabile) della tipografia.

https://practicaltypography.com/

Le immagini sono state realizzate tramite il programma *Inkscape*.

0 - Indice

1 - La macchina di Turing

1.1 - Descrizione della macchina

Una *macchina di Turing* è una macchina che è descritta da un insieme di *simboli* $\Gamma = \{\alpha, \beta, \gamma, \delta, \dots\}$ *finito* e da un insieme di *stati* $Q = \{q_1, q_2, \dots, q_n\}$ anch'esso *finito*. La macchina di Turing dispone di un nastro di memoria *potenzialmente illimitato* a destra e a sinistra, avente delle celle contenenti i simboli; essa identifica il simbolo nella posizione dove la *testina* della macchina è collocata sul nastro. Ad ogni iterazione della macchina di Turing viene letto il simbolo, e a seconda dello stato q_i, viene intrapresa un'azione fra le 3 seguenti:

- spostamento della testina a destra;
- spostamento della testina a sinistra;
- riscrittura del simbolo sotto la testina con uno qualsiasi appartenente all'alfabeto di simboli.

Nello specifico, una macchina di Turing è univocamente identificata dalla sua *matrice di transizione* $\delta : Q \times \Gamma \to Q \times (\Gamma \cup \{L, R\})$[1], dove i simboli L ed

[1] In talune circostanze, è possibile trovare una definizione differente, cioè

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\};$$

in altre parole, è una macchina che *si muove sempre sul nastro,* poiché per ogni stato è sempre definito un movimento. Per questo tipo di macchina, sono necessarie diverse regole d'ingaggio, e i programmi in essa costruiti saranno radicalmente differenti. La differenza fra i due modelli rappresenta un'evidenza della versatilità della macchina di Turing.

R sono rappresentativi dell'azione di spostarsi, rispettivamente, a sinistra e a destra del nastro di memoria. La matrice di transizione lega, dunque, ciascuno stato al simbolo collocato immediatamente sotto alla testina nel nastro di memoria, stabilendo in maniera univoca l'azione da intraprendere. Si può dire dunque che la matrice di transizione fornisce alla macchina di Turing l'elenco delle possibili azioni da intraprendere, alla lettura di un simbolo sulla cella corrente, a seconda dello stato in cui si trova. L'insieme delle azioni che la macchina di Turing compie è quindi la realizzazione del programma stesso, quello che nei termini del modello RAM si sarebbe detto essere la sequenza di istruzioni elementari.

Una macchina di Turing può anche essere accompagnata da un alfabeto *ausiliario*, ovverosia un alfabeto $\mathcal{V}$ comprendente simboli simili a quelli di Γ, ma che vengono utilizzati qualora la macchina di Turing avesse *già processato* la cella in questione (i simboli ausiliari sono "simili" a quelli dell'alfabeto tradizionale, ma hanno una differenza che ne permette il riconoscimento). Tipicamente, l'utilizzo dell'alfabeto ausiliario è importante nel caso specifico in cui si adoperino procedure per le quali è utile ricordare se una cella sia già stata in precedenza processata dalla macchina di Turing, oppure no - in ogni caso, l'alfabeto ausiliario è meramente un sussidio che permette una semplificazione della procedura o aiuta ad interpretare il comportamento della macchina, non è in nessun modo un qualsivoglia tipo di estensione della macchina di Turing. Come sarà mostrato in seguito, ciascuna altra possibile definizione di macchina di Turing è, dal punto di vista della potenza computazione, del tutto equivalente alla definizione già data sopra.

	q_1	q_2	$\cdots$	q_n
α	β/q_2	γ/q_2	$\cdots$	
β	L/q_1	γ/q_3	$\cdots$	
γ	γ/q_3	R/q_2	$\cdots$	
$\vdots$	$\vdots$	$\vdots$	$\ddots$	

Tabella 1.1: Possibile matrice di transizione per una macchina di Turing. Le righe corrispondono ai simboli dell'alfabeto Γ, mentre le colonne sono corrispondenti ai singoli stati dell'insieme Q. Ogni elemento della tabella indica il simbolo da scrivere/stato in cui la macchina dovrà trovarsi al passo successivo,

Diversamente dal modello RAM, la quantità di memoria destinata ad ogni cella è *limitata*, poiché vi può essere collocato soltanto un numero finito di simboli, quelli appunto dell'insieme Γ. Ciononostante, la macchina di Turing si presta meglio alla trattazione di stringhe, poiché i simboli possono rappresentare qualsivoglia tipologia di entità astratta, mentre per il modello RAM si avrebbe necessità di una codifica fra numeri reali e simboli da trattare. Non vi è più dunque la limitazione imposta dal fatto che all'interno di una cella possa risiedere esclusivamente un numero naturale, non importa quanto grande sia; nella macchina di Turing le celle possono contenere simboli di qualsiasi natura essi siano. Lo "svantaggio", tuttavia, è che all'interno di ogni cella non può essere contenuta una quantità *arbitraria* di informazione, come invece avveniva per il modello RAM, che faceva uso dei numeri naturali.

Tipicamente, all'alfabeto che definisce una macchina di Turing viene definito un sottoinsieme sigma di *simboli di input*, $\Sigma \subset \Gamma$, in concomitanza al quale viene definito un simbolo vuoto, *blank*, $b \in \Gamma - \Sigma$. Il simbolo b incarna dunque l'idea di *cella vuota* - si pensi infatti al valore che una cella di memoria primaria qualsiasi di un computer reale avrebbe, al momento immediatamente successivo all'accensione: essa risulterebbe posta allo zero logico, di fatto non conterrebbe alcun valore d'interesse, dato che essa non è ancora stata "toccata" dall'esecuzione di alcun programma. Il simbolo *blank* sta a significare proprio questo, ed è l'equivalente del valore 'zero' del modello RAM, dove all'avvio del programma ogni cella di memoria fuorché quelle contenenti i valori di ingresso contiene il numero reale 0.

Ricapitolando, ogni macchina di Turing viene univocamente definita da:

- un insieme finito di simboli Γ – essi comprendono sia i simboli di input Σ che il simbolo *blank* b;
- un insieme finito di stati Q;
- una funzione (matrice) di transizione $\delta : Q \times \Gamma \rightarrow Q \times (\Gamma \cup \{L, R\})$.

Una diversa maniera per definire una macchina di Turing è tramite la *quaterna* o *quadrupla* q_i, s_j, α, q_k, dove q_i è lo *stato in cui si trova la macchina*, s_j è il *simbolo letto dalla testina*, α è il *simbolo scritto nella matrice di transizione* e q_k è lo *stato successivo* in cui la macchina di Turing si troverà al termine

dell'esecuzione di α. In particolare, l'operazione che la macchina di Turing effettua dipende dal simbolo α scritto nella matrice di transizione:

- se $\alpha = s_i$, sostituisci il simbolo s_j con s_i;
- se $\alpha = R$, muovi la testina a destra;
- se $\alpha = L$, muovi la testina a sinistra.

In parole povere, una macchina di Turing può sovrascrivere un simbolo presente sulla cella con un altro presente nel suo alfabeto, Γ, può spostare la testina di un'unità a destra, e può fare altrettanto a sinistra.

Resta da scogliere il nodo relativo alla terminazione della macchina di Turing. Nel modello RAM, la terminazione avveniva qualora le istruzioni si fossero esaurite, o più precisamente qualora l'indice dell'istruzione successiva fosse quello di un'istruzione assente nella sequenza che definisce la procedura. In assenza del concetto di "istruzione", secondo quale regole dovrebbe terminare una macchina di Turing?

Lo *stop* della computazione di una macchina di Turing avviene qualora la coppia q_i, s_j **non** sia presente nella matrice di transizione. Nel caso di una coppia simbolo—stato non presente nella matrice di transizione, la macchina di Turing avrà terminazione, e vi sarà il riconoscimento del valore finale di computazione, espresso similmente al caso del modello RAM con una convenzione che permetta di identificare il valore finale del risultato della computazione.

Se lo stop della computazione è stato chiarito, ora resta da definire il metodo con cui andremo a recuperare il valore finale della computazione. Convenzionalmente, l'esito di una macchina di Turing è una particolare configurazione di memoria dello stato finale. Si è infatti scelto che il risultato $f(x)$ sia da leggersi come il numero totale di occorrenze di 1 (meno una) sul nastro nella configurazione iniziale, se la computazione è andata a convergenza - indefinito altrimenti. La sequenza di occorrenze del simbolo 1 è delimitata dal carattere *blank*. Quindi vi saranno $f(x)$+1 simboli '1', e pertanto sarà da contare un simbolo '1' in meno (lo '0' sarà indicato con la presenza di un singolo simbolo '1'). Per quanto invece riguarda i risultati di tipo *vettoriale*, cioè del tipo $f(x_1, x_2, \ldots, x_n)$, ebbene sarà sufficiente costruire n "quadrati" in cui racchiudere gli $x + 1$ simboli 1, ciascuno delimitato dal simbolo b. In altre parole, la situazione è quella descritta dalla Figura 1.1.

Per via dell'utilizzo tramite la matrice di transizione è particolarmente difficile programmare sulla macchina di Turing - questo è principalmente dovuto al fatto che la macchina di Turing è una macchina 'a stati', dove non vi è un insieme di istruzioni da applicare direttamente, ma è necessario determinare prima di tutto la matrice di transizione relativa a ciò che bisogna calcolare, stato per stato e simbolo per simbolo.

Una macchina di Turing, non importa come sia stata definita, può essere adoperata sostanzialmente per compiere 3 operazioni:

1. per il *calcolo* di una funzione – la macchina di Turing è intesa come *calcolatore*, e lo scopo è quello di calcolare una funzione $f : \mathbb{N}^n \to \mathbb{N}$. In questo caso, la macchina di Turing risulta essere meno efficiente del modello RAM, per via dell'assenza del comodo sistema di istruzioni presente in quest ultimo;
2. per il *riconoscimento* di una stringa – la macchina di Turing è intesa come *accettore*. In questo contesto la MdT è molto più efficiente del modello RAM, poiché non è richiesta la codifica da numeri naturali a simboli;
3. per la *decisione* di un predicato – la macchina di Turing è intesa come *decisore*.

1.1.1 - Equivalenza fra macchina di Turing e $\mathcal{R}$

Un importante teorema definisce l'equivalenza della macchina di Turing (avente insieme delle funzioni computabili $\mathcal{TC}$ all'insieme delle funzioni

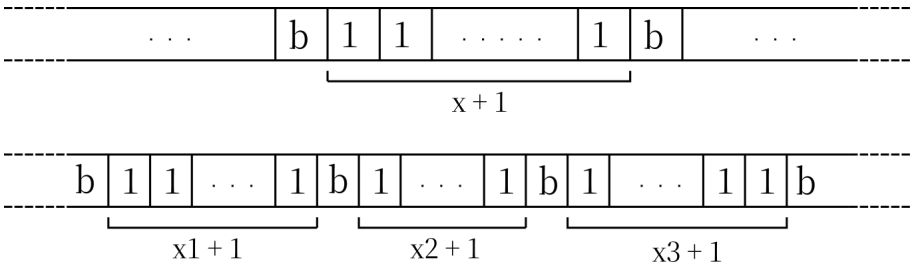


Figura 1.1: Risultato singolo (sopra) e *vettoriale* (sotto) di una macchina di Turing.

parziali ricorsive $\mathcal{R}$, e lo lega indissolubilmente all'insieme delle funzioni computabili dal modello RAM C.

Teorema 1 *dell'equivalenza della macchina di Turing all'insieme $\mathcal{R}$ delle funzioni parziali ricorsive*

$$\mathcal{R} \equiv \mathcal{TC} \equiv C$$

DIMOSTRAZIONE — Un possibile spunto di dimostrazione di $\mathcal{TC} \subseteq \mathcal{R}$ si ha grazie al fatto che la configurazione e lo stato della MdT durante la computazione possono essere codificati da un numero naturale; le operazioni sulla macchina sono rappresentate da funzioni ricorsive su questi numeri. Il viceversa è invece mostrabile tenendo conto che si può verificare che $\mathcal{TC}$ contiene le funzioni di base ed è chiusa rispetto a sostituzione, ricorsione e minimazione illimitata.

1.1.2 - Macchina di Turing per il calcolo della somma

Supponiamo di voler fare la somma fra due numeri interi naturali, x ed y. In questo caso, la macchina di Turing dovrebbe calcolare la funzione $f(x, y) = x + y$. Per fare ciò, costruiremo gli elementi fondamentali della macchina di Turing. In particolare, avremo che l'alfabeto di simboli $\Gamma = \{0, 1, b\}$, cioè avremo bisogno esclusivamente di 2 simboli eccezion fatta per il simbolo *blank*, mentre invece faremo uso di 3 stati $Q = \{q_1, q_2, q_3\}$. Lo stato iniziale è lo stato q_1, mentre lo stato finale è q_3. Resta ora da definire la matrice di transizione. Uno fra i tanti modi di definirla è il seguente (faremo uso delle quadruple),

q_1	1	b	q_1
q_1	b	R	q_2
q_2	1	b	q_3
q_2	b	R	q_2

Tabella 1.2: Matrice di transizione per la macchina di Turing che calcola $f(x, y) = x + y$.

L'idea è quella di togliere due simboli 1, di modo che gli 1 rimanenti corrispondano al valore del risultato finale. Infatti, provando a calcolare $f(2,1) = 3$ avremo che

$$
\begin{array}{cccccccc}
q_1 & & & & & & & \\
1 & 1 & 1 & b & 1 & 1 & b & b \\
q_1 & & & & & & & \\
b & 1 & 1 & b & 1 & 1 & b & b \\
 & q_2 & & & & & & \\
b & 1 & 1 & b & 1 & 1 & b & b \\
 & & q_3 & & & & & \\
b & b & 1 & b & 1 & 1 & b & b
\end{array}
$$

ed il numero finale di simboli 1 corrisponde proprio al valore della somma, $2 + 1 = 3$.

Possiamo anche costruire un grafo della macchina di cui sopra, mostrato in Figura 1.2.

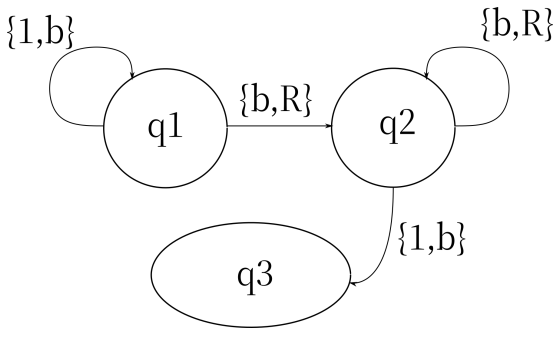


Figura 1.2: Grafo della macchina di Turing per le somme.

1.2 - Altre versioni della macchina di Turing

1.2.1 - Estensioni e menomazioni

Una macchina di Turing può essere apparentemente potenziata mediante l'estensione di essa tramite l'uso di *nastri multitraccia*. In altre parole, anziché adoperare un singolo nastro, si adoperano più nastri contemporaneamente. La macchina di Turing viene espansa tramite l'aggiunta di uno *stato della memoria suppletiva*, che ci indica il numero del nastro dove la macchina di Turing sta agendo. Dunque, ci possiamo immaginare una macchina di Turing con tanti nastri e tante testine che lavorano contemporaneamente, come illustrato in Figura 1.3.

La domanda ora è se l'introduzione della multitraccia consenta di generare una nuova macchina, con capacità di calcolo superiori a quelle della macchina di Turing. La risposta è negativa: dal punto di vista della capacità computazione, una macchina di Turing multinastro non aumenta né diminuisce le capacità. Una macchina di Turing multinastro può essere implementata con un *sistema multitraccia* - in altre parole, vengono adoperate *tante testine quante sono i nastri*. Ci si può facilmente ricondurre alla macchina di Turing convenzionale semplicemente eliminando il sistema multitraccia e facendo agire la macchina su un nastro alla volta, o per meglio dire, una macchina di Turing multitraccia può essere *simulata* da una macchina di Turing convenzionale: essa dunque, non produce alcun tipo di miglioramento dal punto di vista della computazione, cioè le due macchine hanno **la stessa** potenza computazionale (Figura 1.4).

Tale rappresentazione, tuttavia, può avere il vantaggio di presentare una maggiore somiglianza con il tipo di computazione svolto all'interno di un computer moderno. Si pensi infatti alla memoria RAM, alla memoria cache, al disco rigido e così via; una macchina di Turing può dunque "simulare" qualsiasi computer moderno[2] semplicemente introducendo tanti nastri e tan-

[2]Un computer può a sua volta simulare da una macchina di Turing, dal momento che possono essere applicati potenzialmente infiniti banchi di memoria al computer - nella pratica,

te tracce quante sono quelle dei dispositivi fisici adoperati dal calcolatore moderno. Nella fattispecie, si avrà un nastro ed una traccia per la memoria RAM, un altro nastro ed un'altra traccia per la memoria a disco rigido, e così via. In realtà, si tratta esclusivamente di un artificio che ci consente di tracciare un collegamento fra il calcolatore moderno e la macchina di Turing, poiché una macchina multinastro, sia essa multitraccia, può essere *emulata* da una macchina di Turing a nastro singolo.

Il medesimo discorso si applica anche al tentativo di *menomare* la macchina di Turing, nel senso che potremmo pensare di rendere il nastro semi-infinito, cioè illimitato solo a destra o solo a sinistra. In tal caso, benché questa apparente limitazione venga messa in atto, la macchina di Turing menomata avrà di fatto la medesima capacità computazionale di quella "standard", perché possiamo sempre avere a disposizione una quantità illimitata di memoria da un lato (un po' come per il modello RAM), o far uso di trucchi come quello dell'alfabeto ausiliario $\mathcal{V}$ che comunque faciliterebbero le computazioni in una situazione simile. Comunque la si veda, una macchina di Turing non si può né potenziare né depotenziare, a meno di non effettuare operazioni che sconvolgano il suo funzionamento, oppure rimuovendo l'ipotesi della memoria illimitata da entrambi i lati.

tuttavia, sappiamo che ciò non è possibile, e i banchi di memoria non saranno mai del tutto *illmitati*

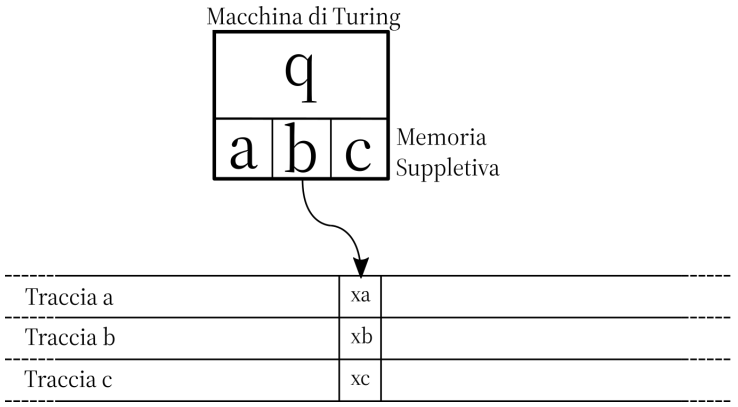


Figura 1.3: Macchina di Turing avente memoria con nastro multitraccia. La testina può collocarsi, una alla volta, su ciascuna delle nastre. La memoria suppletiva rende possibile tenere traccia di quale nastro si sta adoperando per l'esecuzione delle operazioni.

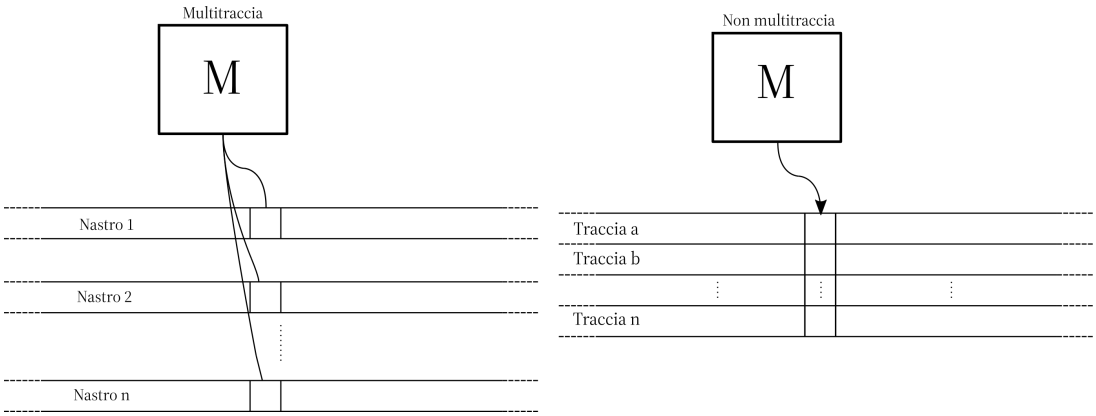


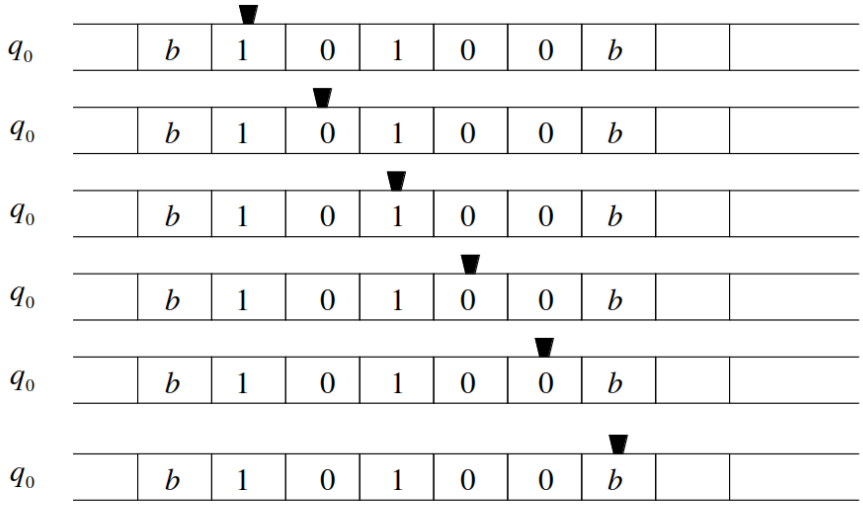
Figura 1.4: Equivalenza fra una macchina di Turing a multitraccia e una macchina di Turing a singola testina. Non importa il numero di testine: la macchina a singola testina potrà sempre percorrere un nastro dopo l'altro, simulando la macchina multitraccia.

1.2.2 - Macchina RAM per *accettare* stringhe

Uno dei possibili utilizzi per una macchina di Turing (o più in generale, per una macchina Turing-equivalente) è quello di *accettore* di stringhe: la macchina riceve in ingresso un simbolo, una *stringa*; essa si dirà *accettata* qualora la computazione risultante terminasse nello stato di ACCETTAZIONE, altrimenti si dirà *rifiutata* qualora la computazione terminasse invece in uno stato di RIFIUTO. Si osservi che, in ogni caso, una macchina di Turing potrebbe ciclare all'infinito; in quel caso saremmo di fronte ad una divergenza.

Nel modello RAM, per accettare una stringa è necessario operare una codifica. In particolare, la stringa viene codificata in un numero naturale e la macchina risponde con "1" o "0" a seconda che la stringa venga o meno accettata. Con la macchina di Turing, invece, si può far intervenire direttamente uno *stato di accettazione* ACCETTAZIONE q_Y o uno *stato di rifiuto* RIFIUTO q_N. La computazione terminerà qualora uno fra questi due stati venisse raggiunto dalla macchina - con conseguente accettazione o rifiuto della stringa a seconda dello stato finale. Un'altra possibilità è quella di accontentarci di *semi-decidere* riguardo l'accettazione di una stringa, cioè di dotarsi di una macchina in grado di riconoscere sì la stringa in questione, ma di *non poterla rifiutare*, poiché in tal caso vi sarebbe un'infinita computazione, una divergenza.

Lo stato iniziale viene di solito denotato con q_0, e potrebbero esserci degli stati ulteriori "intermedi" fra quello iniziale e quelli terminanti. Un esempio di accettazione è dato dalla macchina illustrata in Figura 1.5. La macchina riconosce il linguaggio dato dalle stringhe con due "zeri" nelle ultime due posizioni a destra, in particolare essa riconosce tutte le stringhe che terminano con "00".



	0	1	b
q_0	$(q_0,0,R)$	$(q_0,1,R)$	$(q_1,0,L)$
q_1	$(q_2,0,L)$	$(q_3,0,L)$	$(q_N,0,L)$
q_2	$(q_Y,0,L)$	$(q_N,0,L)$	$(q_N,0,L)$
q_3	$(q_N,0,L)$	$(q_N,0,L)$	$(q_N,0,L)$

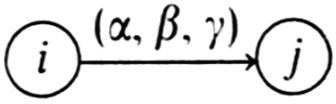
Figura 1.5: Macchina che riconosce il linguaggio dato dalle stringhe con due 0 nelle ultime due posizioni a destra, e suo funzionamento.

1.2.3 - Macchina di Turing definita a grafo

Una macchina di Turing può anche essere "definita a grafo". In questo modello di definizione, la macchina di Turing,

- ha un *nastro semi-illimitato* a destra, diviso in celle;
- ha un alfabeto *ausiliario* $\mathcal{V}$;
- ha un simbolo di spaziatura Δ, equivalente al simbolo *blank* b;
- un puntatore, del tutto equivalente alla testina;
- un *programma*, definito come **grafo finito orientato**, con i vertici definiti come *stato*. Vi è uno stato di inizio, indicato con INIZIO, e un sottoinsieme eventualmente vuoto di stati di arresto, indicati con ACCETTAZIONE. I nodi del grafo sono collegati da *archi*.

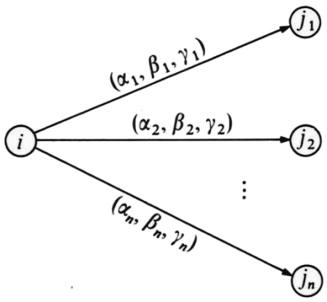
Ciascun arco è della forma



dove

$$\alpha \in \Sigma \cup \mathcal{V} \cup \{\Delta\}, \beta \in \Sigma \cup \mathcal{V} \cup \{\Delta\} \text{ e } \gamma \in \{L, R\}.$$

Dunque, siamo nello stato i; la macchina legge α, scrive β al posto di α, e infine va a destra oppure a sinistra a seconda che il simbolo γ sia pari ad R o ad L. Una proprietà importante è che tutti gli archi che partono da un medesimo vertice devono avere α diversi.



Se così non fosse, si avrebbero più cammini possibili per uno stesso simbolo -- un'ipotesi che come vedremo in seguito sarà violata assumendo che una macchina di Turing possa non essere di tipo *deterministico*.

Si possono disegnare le macchine di Turing direttamente con i grafi. Il problema espresso in Figura 1.6 è il problema del riconoscimento di una stringa avente forma $a^n b^n | n \geq 0$, ed è molto noto nella teoria della computabilità, poiché è un tipico esempio di problema che è risolubile da una macchina di Turing, ma **non risolubile** mediante una *macchina a stati finiti*. Tali macchine, infatti, non presentano alcun tipo di *memoria*, e dunque per questa particolare mancanza non sono in grado di risolvere il problema del riconoscimento. La macchina di Turing, invece, è in grado di risolverlo in virtù della sua superiore potenza di computazione.

Un ulteriore esempio dell'applicazione della macchina di Turing a grafo è mostrato in Figura 1.7, dove la macchina in questione è in grado di concatenare due stringhe a due lettere a e b.

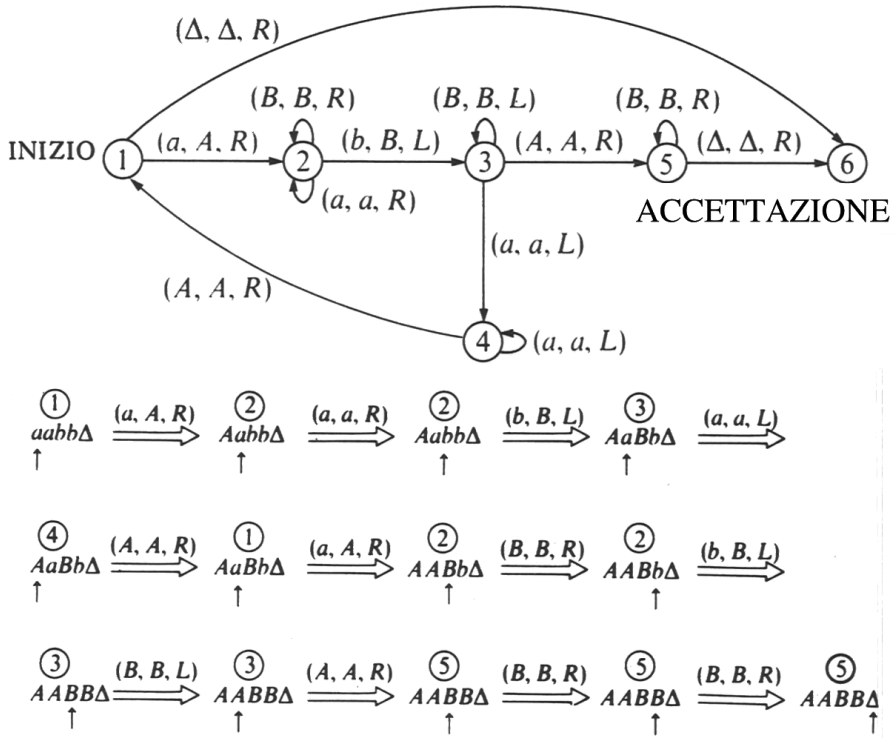
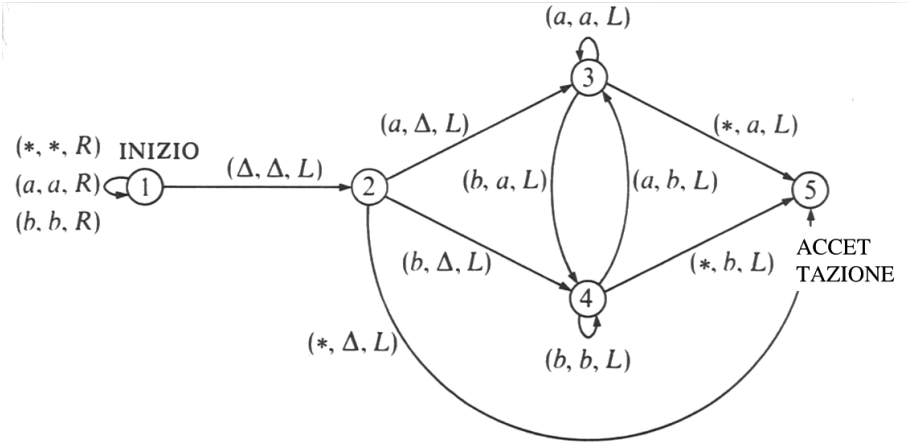


Figura 1.6: Macchina di Turing in grado di riconoscere una stringa della forma $a^n b^n | n \geq 0$.



La macchina di Turing M_7 su $\{a, b, *\}$, descritta nella figura 1.10 calcola la funzione di concatenazione su $\Sigma = \{a, b\}$: essa prende una coppia di parole (w_1, w_2) come entrata e produce la parola $w_1 w_2$

Nastro d'entrata	a	b	a	a	*	b	b	a	b	Δ	Δ	Δ ···
Nastro d'uscita	a	b	a	a	b	b	a	b	Δ	Δ	Δ	Δ ···

Figura 1.7: Macchina di Turing in grado di concatenare due stringhe w_1 e w_2.

2 - Le macchine non deterministiche

2.1 - Gerarchie delle potenze di calcolo

Oltre alla macchina di Turing nelle sue varie versioni, sono possibili altre tipologie di macchine, con vari livelli di gerarchia fra potenze di calcolo. Il seguente elenco ne illustra alcune,

- macchine a *stati finiti*: in questo caso, hanno 0 *memorie push-down* - le macchine a stati finiti sono le meno potenti in assoluto dal punto di vista computazionale, e non sono in grado di risolvere il problema del riconoscimento di stringhe $a^n b^n$, espresso in Figura 1.6;
- macchine a 1 *memoria push-down*: dotate di una memoria push-down che implementa una pila FIFO (*first in—first out*), hanno una maggiore potenza di calcolo rispetto alla macchina a stati finiti;
- macchine a 2 *memorie push-down*: dotate di 2 memorie push-down, esse sono equivalenti alla macchina di Turing.
- macchine *di Post*: sono uno speciale tipo di macchine a memorie push-down, aventi una memoria di tipo LIFO (*last in—first out*). Esse sono equivalenti alle macchine a 2 memorie push-down e alle macchine di Turing;
- una macchina con 3 o più pile push-down non fornisce vantaggi dal livello della potenza computazionale, e sono tutte Turing-equivalenti - il

vantaggio è semmai nella semplificazione di calcoli fornita dall'introduzione della pila aggiuntiva.

Perciò, la gerarchia delle potenze di calcolo è la seguente, dall'alto verso il basso:

1. Macchine non deterministiche di Turing, con due memorie push-down - — Macchine di Turing, Modello RAM, Macchine di Post, macchine finite con due memorie push-down (sono in grado di accettare $\{a^n b^n a^n | n \geq 0\}$). Si potrebbe dimostrare che le macchine di Turing deterministiche e non deterministiche hanno la medesima potenza di calcolo;

2. Macchine finite non deterministiche con una memoria push-down, sono in grado di riconoscere $\{ww^R | w \in \{a, b\}^*\}$[1];

3. macchine finite con una memoria push-down, riconoscono $\{a^n b^n | n \geq 0\}$;

4. Macchine finite non deterministiche senza memorie push-down — macchine finite senza memorie push-down — automi finiti.

2.1.1 - Alcune definizioni sulle stringhe

Sia dato l'alfabeto Σ^*. Faremo uso di tre fondamentali funzioni definite su Σ^*:

- l'operazione $testa(x)$, che fornisce la "testa" di una stringa, ovverosia la lettera di estrema sinistra della parola x;
- l'operazione $coda(x)$, la quale invece fornisce la "coda" di una stringa, o la stringa privata della sua testa x;
- l'operazione $\sigma \cdot x$, che *concatena* la lettera σ e la parola x, per formare un'unica parola nuova.

[1] Il *nodo di decisione* che introduce il non determinismo serve per gestire la situazione data dall'incapacità di riconoscere il punto di rottura ww^R, cioè il punto in cui finisce w ed inizia w^R. Con il non determinismo, ogniqualvolta si arriva al nodo di decisione si tengono valide entrambe le possibilità - dunque, è più potente.

2.2 - La macchina di Turing non deterministica

La macchina di Turing effettiva è di tipo *deterministico*. Nella fattispecie, una macchina deterministica definita a grafo vede ciascun arco che parte dallo stesso vertice avere *simboli diversi*. Se ciò non fosse vero, leggendo un unico simbolo α_i e trovandosi nello stato q_j la macchina di Turing non potrebbe "scegliere" il percorso, poiché ne esisterebbe più di uno. Viceversa, una macchina *non deterministica* rompe questa assunzione, e permette alla macchina di compiere una scelta, sia essa arbitraria o del tutto casuale, riguardo quale percorso seguire fra i vari disponibili.

Il *non determinismo* viene introdotto per due ragioni:

- si desidera osservare se una macchina di Turing non deterministica sia o meno più *potente* di una macchina di Turing deterministica;
- si cerca di valutare se possano esistere differenti paradigmi di computazione (ad esempio, computazione parallela).

Una *macchina di Turing non deterministica* può avere nel suo diagramma di flusso dei *nodi di decisione* dove a fronte di un medesimo simbolo vi sono *più archi possibili* - il non determinismo viene dunque introdotto da questo fenomeno: una macchina di Turing non deterministica può scegliere il suo percorso con una legge non deterministica, ma semmai dettata dal caso o da una *scelta arbitraria*.

Una macchina non deterministica accetta l'idea che vi siano più possibilità di sviluppo di una computazione a fronte di un medesimo simbolo identificato sul nastro e associato ad un determinato stato; sono dunque possibili diversi percorsi di computazione. Questo tipo di macchina, ad esempio, potrebbe concepire il *parallelismo* nella computazione, nel senso che più possibilità e percorsi nel calcolo sono possibili. Questa potenzialità, apparentemente di gran lunga migliorativa, **non aumenta** la potenza di calcolo della macchina di Turing. Una macchina di Turing deterministica può, infatti, simulare una macchina non deterministica: è sufficiente compiere una ricerca per ogni diramazione dell'albero dei nodi, con un aumento esponen-

ziale[2] del numero di operazioni da effettuare, ma pur sempre un'operazione realizzabile e computabile da una macchina di Turing. Una macchina non deterministica ha dunque il potenziale vantaggio di poter risolvere problemi in *tempo lineare* che dalla macchinia deterministica sarebbero risolti in in tempo esponenziale.

In questo caso, il parallelismo è insito nella macchina non deterministica; supponendo di voler risolvere un problema di accettazione, la stringa si considera *accettata* qualora uno qualunque fra i percorsi possibili incappa nello stato di ACCETTAZIONE. Una parola $w \in \Sigma^*$ si dice *accettata* da una macchina di Turing non deterministica se esiste una computazione della macchina M che cominci con entrata $x = w$, e che termini ad un arresto con lo stato di ACCETTAZIONE. Se w non viene accettata e lo stato di arresto è quello del RIFIUTO, allora si dice che w è *rifiutata*. In alternativa, siamo di fronte ad un ciclo infinito $w \in ciclo(M)$, cioè dinanzi ad una divergenza.

Possiamo quindi pensare ad una macchina di Turing non deterministica come ad un macchina avente per ogni cella, nella matrice di transizione, un *insieme* $\delta(q, s)$ di triple q_i, s_i, α_i, dove $\alpha_i \in \{L, R\}$ e ad ogni elemento dell'insieme corrisponde una possibile scelta da compiere arbitrariamente o casualmente; dunque da lì è ottenuto il non determinismo della macchina. Ogni possibile scelta genererà un diverso ramo nell'albero della computazione - per l'accettazione di un simbolo è sufficiente che uno *qualsiasi* fra i rami di computazione termini nello stato di ACCETTAZIONE. Dunque, benché una macchina di Turing non deterministica non **aumenti** la potenza di calcolo intrinseca della macchina, essa consente la **parallelizzazione** dei possibili percorsi della computazione, rendendo di fatto possibile risolvere in tempo lineare problemi che sarebbero risolubili (comunque), ma in tempo esponenziale per una macchina deterministica.

[2] L'aumento esponenziale è dovuto al fatto che, ad ogni diramazione effettuata da una macchina di Turing non deterministica, la relativa macchina deterministica dovrà (almeno) sdoppiarsi su 2 percorsi.

2.2.1 - Equivalenza fra macchine non deterministiche e macchine multinastro

Teorema 2 *Ogni macchina di Turing non deterministica ha un equivalente macchina di Turing deterministica multinastro.*

DIMOSTRAZIONE — Una traccia della dimostrazione è una ricerca nell'albero di computazione. Con almeno due nastri, si simula con uno la macchina non deterministica mentre con l'altro si collezionano tutti i possibili k "prossimi passi" della tabella di transizione. La macchina di Turing deterministica allora controllerà tutte le configurazioni stato—simbolo, livello per livello, dell'albero di computazione. Lo stato finale di ACCETTAZIONE terminerà la computazione complessiva.

L'idea è quella, dunque, di *simulare* il non determinismo compiendo un numero crescente in modo esponenziale di passi, uno per ogni ramo dell'albero non deterministico di computazione.

Teorema 3 *Ogni macchina di Turing non deterministica $T_M(n)$ ha un equivalente macchina di Turing deterministica con ordine $2^{O(T_M(n))}$.*

DIMOSTRAZIONE — Una traccia può essere che il numero massimo di foglie è $O(b^{T_M(n)})$, dove b è il numero di figli. Il tempo per viaggiare dalla radice lungo ogni ramo è, per una macchina multinastro,

$$O(T_M(n)b^{T_M(n)}) = 2^{O(T_M(n))},$$

mentre per una macchina a nastro singolo

$$(2^{O(T_M(n))})^2 = 2^{O(2T_M(n))} = 2^{O(T_M(n))},$$

dunque l'ordine è lo stesso.

In altre parole, una macchina non deterministica è in grado di "evocare" un numero esponenziale ed arbitrariamente elevato di macchine di Turing deterministiche - tale funzionalità però è irrealizzabile, poiché corrisponderebbe a dotare la propria macchina di un numero arbitrario di unità di calcolo, ciascuna per ogni possibile passo della computazione, per farle lavorare in parallelo. Se ciò fosse invece possibile nel mondo materiale, potremmo effet-

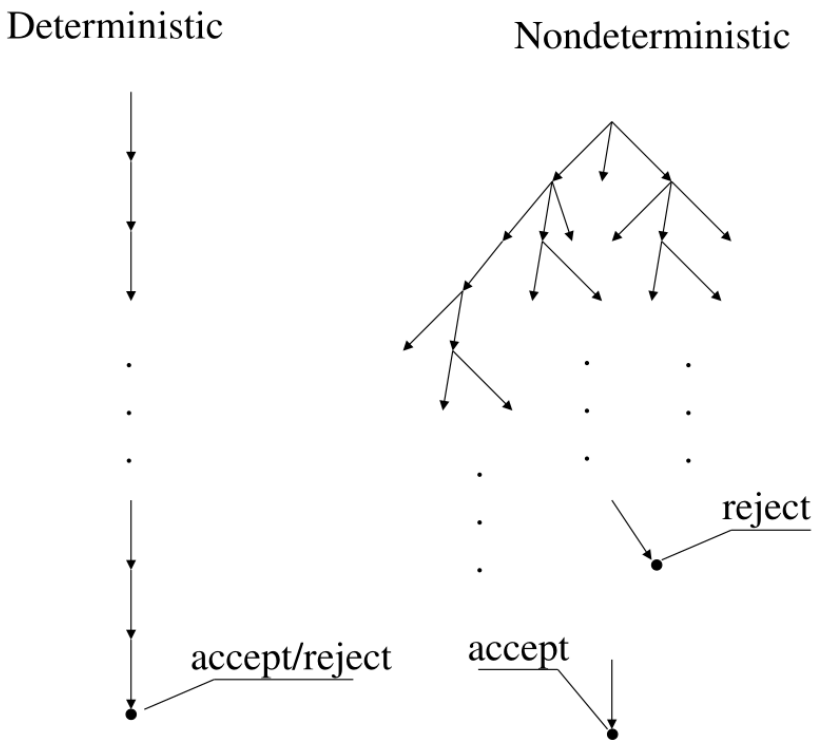


Figura 2.1: Esempio di computazione “in parallelo” effettuata dalla macchina non deterministica (a destra), e simulazione da parte della macchina deterministica (a sinistra).

tivamente risolvere problemi difficilmente trattabili in tempo polinomiale, rendendoli di fatto trattabili.

3 - Le reti neuronali di Hopfield

Le *reti neuronali* incarnano un diverso paradigma rispetto a quello della computazione procedurale. Storicamente, esse prendono spunto dalla natura, in particolare dal concetto di *neurone*, inteso come singola unità di calcolo, dotata di input, di output e di una *funzione caratteristica*. Diversamente dall'idea della computazione procedurale, dove la *complessità* è definita tramite la complessità di tipo computazionale (o temporale), le reti neuronali esprimono la loro complessità attraverso la **complessità strutturale**, o **complessità circuitale**. La computazione è di tipo *collettivo*, ed emerge come proprietà dell'*evoluzione dinamica* della rete. Ogni decisore locale, detto *neurone*, non ha visibilità della computazione globale, e svolge il proprio compito localmente — ogni decisore locale concorre alla soluzione globale in modo sfumato: la rete è **robusta** rispetto a malfunzionamenti locali.

Dal punto di vista strettamente logico, il paradigma di computazione a reti neuronali è l'unico paradigma radicalmente differente da quello procedurale, mentre il paradigma *a DNA*[1] è una tipologia di calcolo che, nella realtà, non differisce sostanzialmente dal metodo procedurale.

Esistono due filoni di reti neuronali; il primo associato ai **perceptron**, macchine a strati che fungono da riconoscitori di pattern e configurazioni. Le variabili di ingresso rappresentano un ente, una configurazione del sistema esterno, e il perceptron è in grado di fornire in output una risposta che

[1] TEST FIXME TODO Trattasi di un paradigma di calcolo dove si va a cercare lo spazio delle soluzioni, e si eleggono quelle più "performanti" — talune soluzioni saranno la base dalla quale saranno definite le successive soluzioni, convergendo dunque alla soluzione. La potenza di questo metodo è quello di potersi permettere una ricerca esauriente della soluzione, poiché ciascun DNA è infinitesimamente piccolo, e dunque può essere esaminato in parallelo.

consente di riconoscere tale configurazione o pattern in base a come essa sia stata configurata e ai suoi parametri. Il secondo filone, invece, è quello della *rete di Hopfield*; tale filone sarebbe in grado, in linea di principio, di **risolvere problemi** di natura matematica. Mediante una rete di Hopfield è possibile risolvere ad esempio il *travelling salesman problem*, un problema presumibilmente intrattabile[2].

Le reti neuronali hanno avuto grande successo in 3 periodi storici:

- all'inizio degli anni 40 - nel 1948 fu fornito il primo modello di neurone e rete neuronale. L'idea fu quella di avvalersi di una macchina fisica per simulare il comportamento dei neuroni naturali, presenti nel nostro cervello (Figura 3.1. All'epoca uscirono anche articoli su memorie a breve e lungo termine adoperando reti neuronali;

- durante gli anni 80 - nei primi anni 80, Hopfield individuò un modello di reti neuronali che associava le reti neuronali ad un particolare modello matematico; qualità che prima era assente. Si tratta di una caratteristica fondamentale: quando si cerca di risolvere problemi con le reti neuronali, l'assenza di teoremi (ad esempio, quelli asintotici) non ci permette di stabilire la *qualità* di una soluzione, oppure se la computazione andrà in qualche maniera a buon fine. Dunque, l'assenza di un buon apparato matematico che faccia da base alle reti neuronali è il principale svantaggio di tale paradigma. Hopfield riuscì a fornire la soluzione di alcuni fra questi problemi matematici, suggerendo la possibilità di risolvere problemi, come ad esempio quello sopra citato del *travelling salesman problem*. Ci fu allora una corsa dal punto di vista scientifico riguardo la possibilità di risolvere in maniera efficiente i problemi presumibilmente intrattabili. Si può dimostrare, tuttavia, che la potenza di computazione di una macchina di Hopfield **non è superiore** alla capacità di computazione della macchina di Turing. Con una macchina di Hopfield non è dunque possibile risolvere in tempo polinomiale problemi che non sono risolubili in tale maniera già dalla macchina di Turing - vi è dunque l'*equivalenza* fra le due macchine. Le reti di Hopfield caddero pertanto ben presto in disuso;

[2] "Presumibilmente" si riferisce al fatto che, fino ad ora, nessuno è riuscito a dimostrare né che tale problema può essere risolto in tempo polinomiale, né che non può esserlo.

- il terzo ed ultimo periodo d'oro delle reti neuronali è al giorno d'oggi, dove la potenza di calcolo superiore delle macchine moderne apre la strada ad un uso maggiormente ampio delle reti neuronali, che si avvalgono della pura potenza computazionale di gran lunga maggiore rispetto al passato per produrre risultati in tempo apprezzabile.

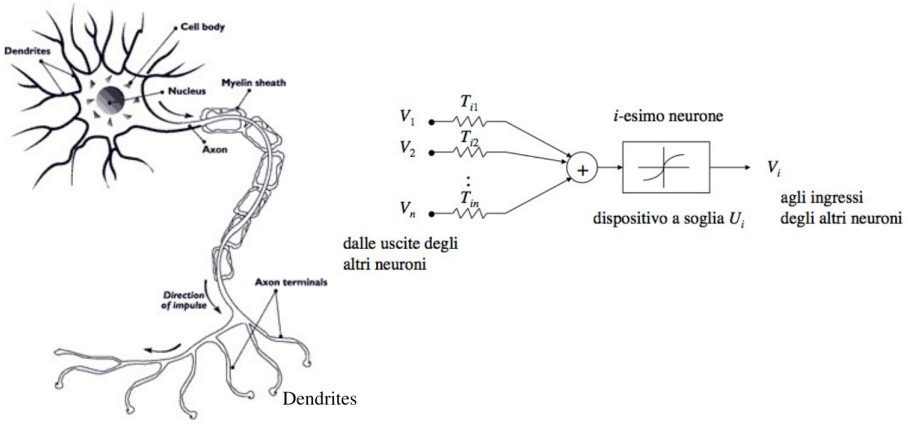


Figura 3.1: Paragone fra neurone naturale e neurone di Hopfield.

3.1 - Il neurone reale ed il neurone simulato

Una macchina di Hopfield simula un neurone reale. Vi sono all'incirca 10^{11} neuroni (dieci miliardi) nel cervello, i quali formano una rete neuronale di una complessità disarmante. Il cervello è in grado di svolgere una quantità di compiti enorme, in modo efficiente - il cervello dunque è la struttura più complessa che esista nell'universo noto. I segnali elettrici nel cervello si sviluppano nell'ambito dei *segnali continui* (benché vi siano degli *spike*), tuttavia il suo funzionamento si svolge nell'ambito dei *segnali discreti*; un neurone si "eccita" o si "rilassa", manifestando dunque un comportamento del tutto discreto da questo punto di vista.