

Macchine di Turing non-
deterministiche.
La tesi di Church-Turing

28 aprile 2022

MdT non deterministica

Definizione (Macchina di Turing non deterministica)

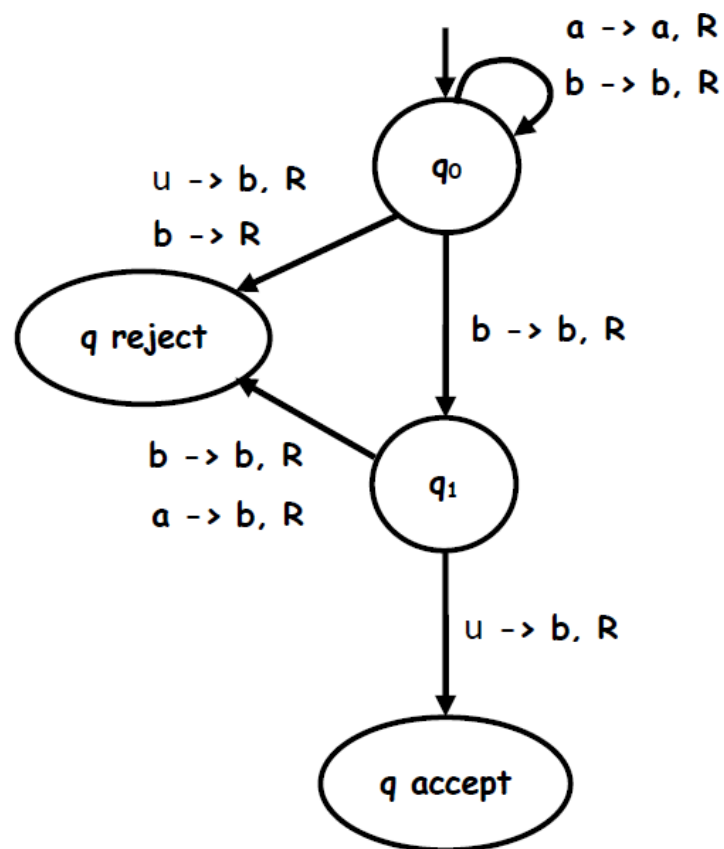
Una Macchina di Turing **non deterministica** è una settupla $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ dove:

- $Q, \Sigma, \Gamma, q_0, q_{\text{accept}}, q_{\text{reject}}$ sono definiti come in una MdT deterministica
- la funzione di transizione δ è definita al modo seguente:

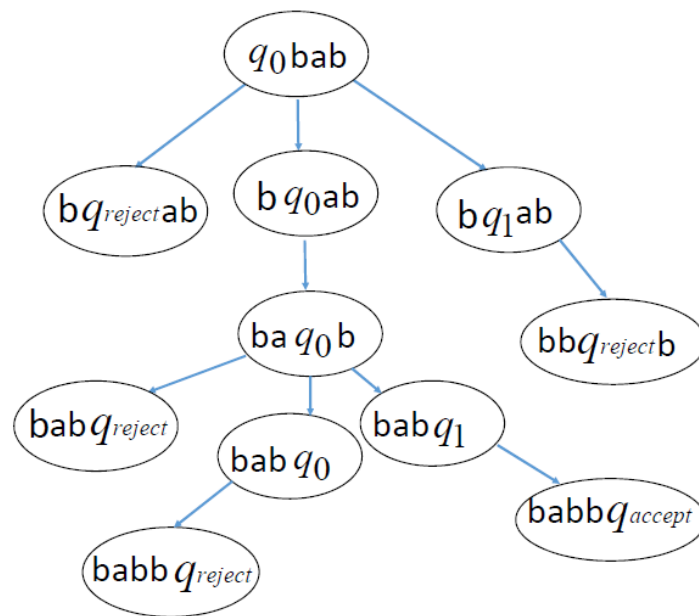
$$\delta : (Q \setminus \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Nota: Se $\delta(q, a)$ è l'insieme vuoto, la macchina si muove nello stato q_{reject} e si ferma.

Esempio – M_1



Albero delle computazioni per bab



$$\delta(q_0, b) = \{(q_0, b, R), (q_1, b, R), (q_{reject}, b, R)\}$$

MdT non deterministica

Sia N una macchina di Turing non deterministica e sia w una stringa.

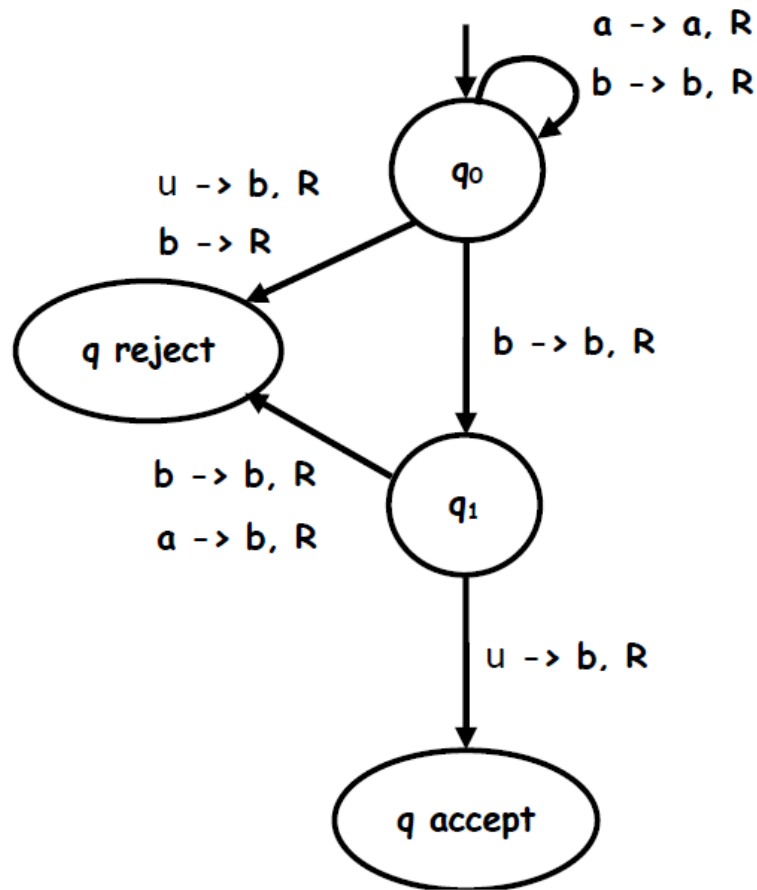
- N **accetta** w se e solo se **esiste almeno una** computazione $q_0w \rightarrow^* uq_{accept}v$, dove q_0w è la configurazione iniziale di N con input w e $uq_{accept}v$ è una configurazione di accettazione. L'albero delle computazioni di N su w contiene almeno una foglia con stato q_{accept} .
- N **rifiuta** w se e solo se **ogni** computazione dalla configurazione iniziale q_0w con input w termina in una configurazione di rifiuto. L'albero delle computazioni di N su w è finito e tutte le foglie sono configurazioni con stato q_{reject} .

Definizione

Sia $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ una MdT non deterministica. Il **linguaggio** $L(N)$ **riconosciuto** da N , è l'insieme:

$$L(N) = \{w \in \Sigma^* \mid \text{esiste una computazione } q_0w \rightarrow^* uq_{accept}v, \\ u, v \in \Gamma^*\}$$

Esempio – M_1



$$L(M_1) = \{wb \mid w \in \{a, b\}^*\}$$

Equivalenza

Teorema

Per ogni macchina di Turing non deterministica N esiste una macchina di Turing deterministica D equivalente ad N , cioè tale che $L(N) = L(D)$.

Idea della prova

- Ogni computazione di N deriva da una sequenza di scelte che D deve riprodurre.
 D simula N provando tutte le possibili scelte che può fare N durante la sua computazione nondeterministica.
- Quindi, per ogni input w , D esplora l'albero delle computazioni di N su w .
- Se D trova lo stato di accettazione su uno qualsiasi dei rami dell'albero, allora D accetta. Altrimenti, la simulazione effettuata da D **non terminerà**.
- Questo assicura che $L(D) = L(N)$.
- Una esplorazione dell'albero tramite **visita BFS** assicura che, se nell'albero c'è un nodo contenente una configurazione di accettazione, lo **scoprirà** in un numero finito di passi.

Idea della prova

Vogliamo rappresentare le computazioni di N su una stringa w , cioè i cammini nell'albero delle computazioni di N su w mediante una stringa.

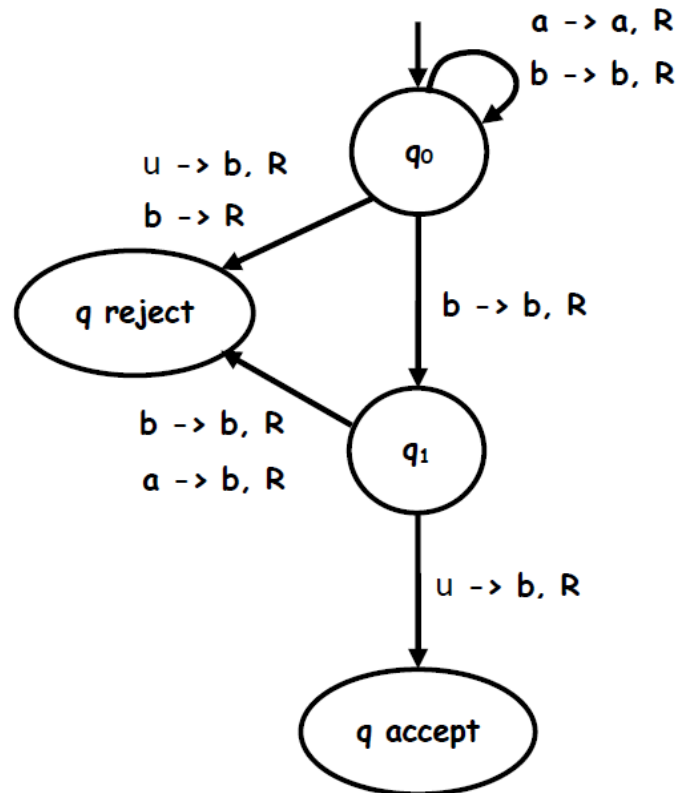
Sia $N = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ una macchina di Turing non deterministica.

Sia B il **massimo numero di scelte** di N su ogni coppia stato-simbolo.

Ovvero B è il massimo numero di figli che può avere un nodo in un albero delle computazioni.

Denotiamo $\Gamma_B = \{1, 2, \dots, B\}$.

Esempio – M_1



B = 3

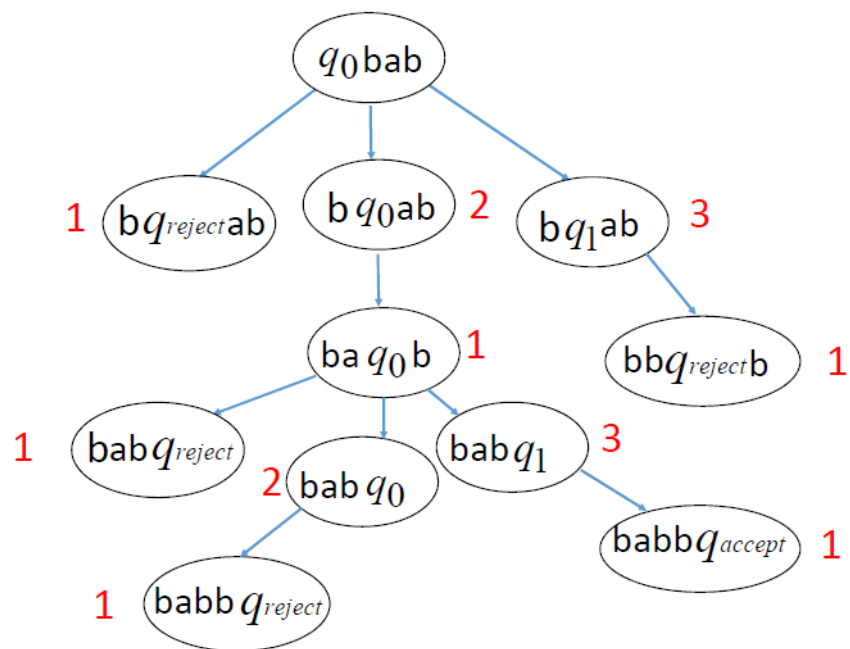
$$\delta(q_0, b) = \{(q_0, b, R), (q_1, b, R), (q_{\text{reject}}, b, R)\}.$$

Associamo 1 a $(q_{\text{reject}}, b, R)$, 2 a (q_0, b, R) e 3 a (q_1, b, R) .

Esempio – M_1

$$\delta(q_0, b) = \{(q_0, b, R), (q_1, b, R), (q_{reject}, b, R)\}.$$

Associamo 1 a (q_{reject}, b, R) , 2 a (q_0, b, R) e 3 a (q_1, b, R) .

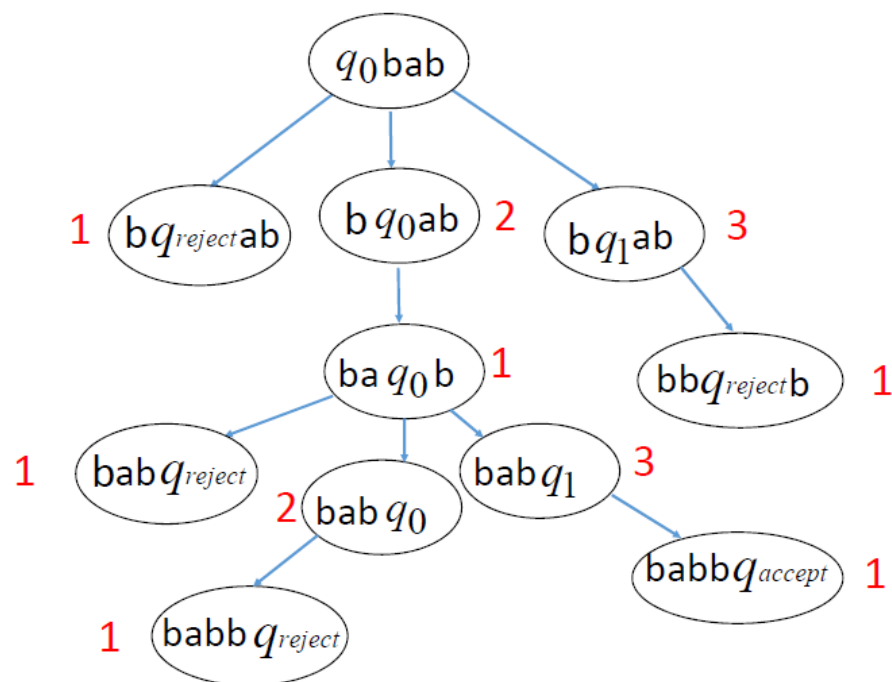


La computazione che mostra che bab è accettata.

2131 :

$$q_0bab \rightarrow bq_0ab \rightarrow baq_0b \rightarrow babq_1 \rightarrow babbq_{accept}$$

Esempio – M_1



21 : $q_0bab \rightarrow bq_0ab \rightarrow baq_0b$
 22, 23 senza sbocco, muore.

L'ordine in cui BFS scopre i nodi di un albero in cui ogni nodo ha B figli è l'**ordine radix** (o per lunghezza) sulle stringhe in $\{1, 2, \dots, B\}$.

Esempio, $B=3$:

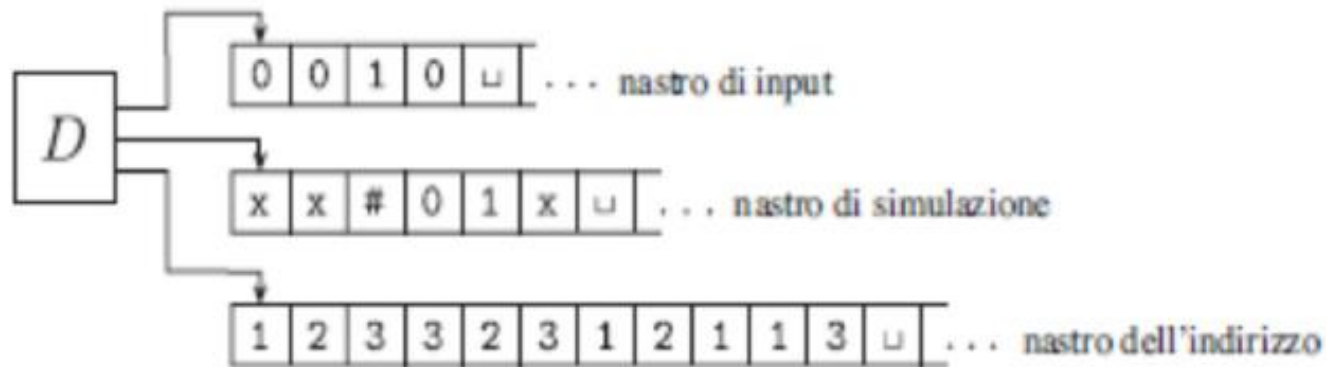
Epsilon, 1, 2, 3, **11, 12, 13**, 21, **22, 23**, 31, **32, 33**, **111, 112**,

Esercizio

Descrivere una MdT che data una stringa **w** su $\{1, 2\}$ calcoli la **successiva** di **w** nell'ordine radix.

Ordine radix = ordine per lunghezza e a parità di lunghezza ordine numerico secondo $1 < 2$

Idea della prova



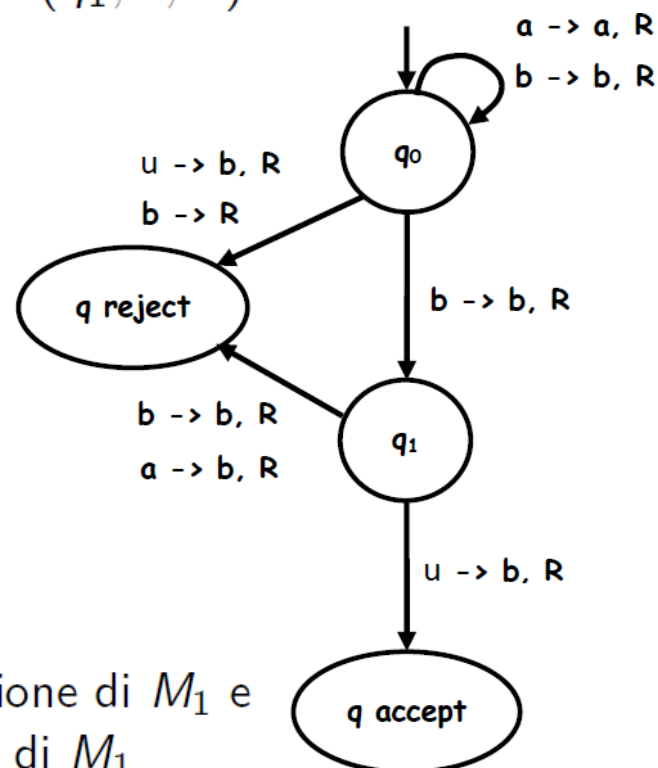
- Sul **primo nastro** è memorizzata la stringa input w (sulla parte più a sinistra) e il contenuto del primo nastro non verrà alterato dalle computazioni di D .
- Sul **secondo nastro** viene eseguita la simulazione di N . Il nastro 2 mantiene una copia del nastro di N corrispondente a qualche diramazione dell'albero delle computazioni.
- Sul **terzo nastro** vengono generate le **codifiche** delle possibili computazioni di N con input w .

Il nastro 3 tiene traccia della posizione di D nell'albero delle computazioni di N .

Descrizione di D :

- ① Inizialmente il nastro 1 contiene l'input w e i nastri 2 e 3 contengono solo \sqcup . (D inizializza la stringa sul nastro 3 a ϵ .)
- ② D copia il contenuto del nastro 1 sul nastro 2.
- ③ Utilizza il nastro 2 per simulare N con input w sulla ramificazione della sua computazione non deterministica corrispondente alla stringa sul nastro 3, cioè riproduce la successione di scelte del corrispondente cammino (se tale stringa corrisponde a una computazione). Prima di ogni passo di N , consulta il simbolo successivo sul nastro 3 per determinare quale scelta fare tra quelle consentite dalla funzione di transizione di N . Se si raggiunge una configurazione di accettazione, D accetta l'input. Altrimenti (se si raggiunge una configurazione di rifiuto, se la stringa non corrisponde a una computazione, o al termine della simulazione sulla stringa) D passa al passo 4.
- ④ D genera sul nastro 3 la stringa successiva a quella corrente in ordine radix e torna al passo 2.

Esempio. $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ è tale che $\delta(q_0, b) = \{(q_0, b, R), (q_1, b, R), (q_{reject}, b, R)\}$. Supponiamo di associare 2 alla scelta (q_0, b, R) , 3 alla scelta (q_1, b, R) e 1 alla scelta (q_{reject}, b, R) .



Poniamo $D = (Q', \Sigma, \Gamma', \delta', q'_0, q_{accept}, q'_{reject})$.

Sia p_0 lo stato che corrisponde a q_0 nella simulazione di M_1 e p_1 lo stato che corrisponde a q_1 nella simulazione di M_1 (eventualmente $p_0 = q_0$ e $p_1 = q_1$). Allora

$$\delta'(p_0, \sigma, b, 2) = (p_0, \sigma, b, 2, S, R, R),$$

$$\delta'(p_0, \sigma, b, 3) = (p_1, \sigma, b, \mathbf{3}, S, R, R) \text{ e}$$

$$\delta'(p_0, \sigma, b, 1) = (q_{reject}, \sigma, b, 1, S, R, R), \text{ per ogni carattere } \sigma.$$

Equivalenza

Teorema

Per ogni macchina di Turing non deterministica N esiste una macchina di Turing deterministica D equivalente ad N , cioè tale che $L(N) = L(D)$.

Ricorda:

per ogni MdT multinastro esiste una MdT che riconosce lo stesso linguaggio

Riconoscere

Corollario

Un linguaggio L è Turing riconoscibile se e solo se esiste una macchina di Turing non deterministica N che lo riconosce, cioè tale che $L(N) = L$.

Nota: se L è Turing riconoscibile allora esiste una MdT deterministica

$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ che lo riconosce, cioè tale che $L(M) = L$.

È facile vedere che la macchina di Turing non deterministica $M' = (Q, \Sigma, \Gamma, \delta', q_0, q_{accept}, q_{reject})$, dove $\delta'(q, \gamma) = \{\delta(q, \gamma)\}$, per ogni $q \in Q$ e $\gamma \in \Gamma$, è tale che $L(M') = L(M) = L$.

Decidere

- Una macchina di Turing non deterministica è un **decisore** se, per ogni stringa input w , tutte le computazioni a partire da q_0w terminano in una configurazione di arresto.
- Una macchina di Turing non deterministica N **decide** L se N è un decisore e $L = L(N)$.

Nota. Sia N una macchina di Turing non deterministica tale che, per ogni input w , N accetta w o N rifiuta w .

N non è necessariamente un decisore.

Una stringa accettata potrebbe avere computazioni che non terminano

Riconoscere e decidere

- Una macchina di Turing non deterministica è un **decisore** se, per ogni stringa input w , tutte le computazioni a partire da q_0w terminano in una configurazione di arresto.
- Una macchina di Turing non deterministica N **decide** L se N è un decisore e $L = L(N)$.

Anche per le macchine di Turing non deterministiche sono definite due famiglie di linguaggi:

- i linguaggi L **riconosciuti** da macchine di Turing non deterministiche;
- i linguaggi L **decisi** da macchine di Turing non deterministiche.

Esercizio 3.3

L'esercizio 3.3 di [Sipser] chiede di dimostrare il seguente **Corollario**, supponendo valido il seguente Teorema sugli alberi

Teorema

Se ogni nodo in un albero ha un numero finito di figli e ogni cammino dell'albero ha un numero finito di nodi, allora l'albero ha un numero finito di nodi.

Corollario

Un linguaggio L è decidibile se e solo se esiste una macchina di Turing non deterministica N che lo decide.

Esercizio 3.3

Corollario

Un linguaggio L è decidibile se e solo se esiste una macchina di Turing non deterministica N che lo decide.

Prova

Se un linguaggio L è decidibile, esiste una macchina di Turing deterministica M che è un decisore e che accetta L . M può essere facilmente trasformata in una macchina di Turing non deterministica che decide L .

Viceversa, se un linguaggio L è deciso da una macchina di Turing non deterministica N , definiamo una macchina di Turing deterministica D' , modificando la precedente definizione di D come segue.

Descrizione di D' :

- ① Inizialmente il nastro 1 contiene l'input w e i nastri 2 e 3 contengono solo \sqcup . (D' inizializza la stringa sul nastro 3 a ϵ .)
- ② D' copia il contenuto del nastro 1 sul nastro 2.
- ③ Utilizza il nastro 2 per simulare N con input w su una ramificazione della sua computazione non deterministica, riproducendo la successione di scelte del cammino corrispondente alla stringa sul nastro 3 (se tale stringa corrisponde a una computazione). Se si raggiunge una configurazione di accettazione, D' accetta l'input. Altrimenti (se si raggiunge una configurazione di rifiuto, se la stringa non corrisponde a una computazione, o al termine della simulazione sulla stringa) D' passa al passo 4.
- ④ Rifiuta se tutti i cammini dell'albero delle computazioni di N su w hanno portato a una configurazione di rifiuto. Altrimenti D' passa al passo 5.
- ⑤ D' genera sul nastro 3 la stringa successiva a quella corrente in ordine radix e torna al passo 2.

Esercizio 3.3

Possiamo dedurre che D' è un decisore per L .

Se N accetta il suo input w , allora D' troverà un cammino che termina in una configurazione di accettazione e D' accetta w .

Se N rifiuta il suo input w , tutte le sue computazioni su w terminano in una configurazione di rifiuto perché è un decisore. Quindi ciascuno dei cammini ha un numero finito di nodi poiché ogni arco nel cammino rappresenta un passo di una computazione di N su w . Pertanto l'intero albero di computazione di N su w è finito. Quindi, D' si fermerà e rifiuterà quando l'intero albero sarà stato esplorato.

Esercizio 3.3

Nota che la macchina D' deve individuare se tutte le computazioni su w conducono a una configurazione di rifiuto.

Quindi D' deve avere un controllo sulle stringhe che rappresentano codifiche di computazioni su w che terminano in q_{reject} . Se x è una stringa che codifica una tale computazione, D' deve bloccare la generazione di stringhe in ordine radix con prefisso x .

Analogamente, se la stringa x è una stringa non valida, cioè non corrisponde a una computazione, D' deve bloccare la generazione di stringhe in ordine radix con prefisso x .

Varianti equivalenti di MdT

Abbiamo visto alcune **varianti** di MdT

- MdT con possibilità di S (Stayer)
- MdT multi- nastro
- MdT non deterministiche

E abbiamo dimostrato che sono tutte **equivalenti** alla MdT.

Altre varianti equivalenti di MdT

- MdT a singola scrittura (Ex. 3.17)
- MdT a nastro doppiamente infinito (ex. 3.18)
- MdT con Reset a sinistra (Ex. 3.19)

Sono tutte equivalenti alla MdT.

Dagli anni '30 (del secolo scorso), quando ancora l'informatica e i computer NON esistevano, quindi indipendentemente da essi, sono stati proposti molti altri modelli di computazione (anche molto differenti dalle MdT).

Calcolabilità

Negli anni '30 (del secolo scorso), quando ancora l'informatica e i computer NON esistevano, quindi indipendentemente da essi, alcuni **logici**, motivati anche dalla lista dei **23 problemi** proposti da **Hilbert** nel 1900, si proposero questo progetto:

Formalizzare in modo esatto la nozione intuitiva di problema e di procedura effettiva di calcolo, così da definire quando un problema è risolvibile o no.

Questo progetto fu realizzato indipendentemente e con una diversa risposta da

- **Alan Turing** (1936) con il suo modello noto come **Macchina di Turing**
- **Alonso Church** (1936) con il **λ -calcolo**.

Tesi di Church - Turing

Fu dimostrato che le due risposte erano **equivalenti**, cioè definivano la stessa classe di problemi risolubili. Tutto ciò che poteva essere calcolato con una **Macchina di Turing** poteva essere calcolato (simulando la MdT) col **λ -calcolo**, e viceversa.

In seguito, ogni altra "**ragionevole**" formalizzazione della nozione di **procedura effettiva** ha condotto agli stessi risultati.

Per questo è possibile parlare di un concetto di "calcolabilità", **indipendente** dal particolare formalismo.

Tesi di Church-Turing:

Tutto ciò che può essere intuitivamente calcolato tramite una procedura effettiva può essere calcolato con una **Macchina di Turing**.

Tenendo conto che essa è antecedente alla costruzione dei primi computer, costituì un enorme salto intellettuale.

Tesi di Church - Turing

La Tesi di Church – Turing non è un teorema, non dovrete studiarne la dimostrazione!

Anche perché non è proprio dimostrabile.

Può essere visto come definizione di **procedura effettiva** / algoritmo.

Questo significa anche che qualsiasi modello di «calcolatore» si possa mai progettare oggi o in futuro, con tutte le risorse e le tecnologie possibili, non potrà che risolvere i problemi risolti con una Macchina di Turing!

Per esempio, anche i “quantum computer” obbediscono alla tesi, quindi non possono fornire vantaggi sui computer classici in termini di **computabilità**. Possono invece fornirli in termini di **complessità** di tempo.

Per esempio, per la **fattorizzazione** di interi in numeri primi come dimostrato da Shor nel 1994.

Procedura effettiva = Algoritmo = MdT (e sue varianti)

- Algoritmi per eseguire calcoli in svariati campi erano noti già nell'antichità (esempio: Algoritmo di Euclide per il calcolo del MCD, circa 300 A.C.).
Turing ha il merito di aver definito in modo formale il concetto di algoritmo.
- L'esigenza nacque a causa di un problema posto da Hilbert, al Congresso internazionale di Matematica del 1900 (Parigi), il decimo di una lista di 23 problemi.
- Il decimo problema di Hilbert si può riformulare come la richiesta di una soluzione algoritmica a un problema relativo ai polinomi.

Il X problema di Hilbert

Nel **1900**, D. Hilbert presentò al Secondo Congresso Internazionale di Matematica a Parigi, una lista di 23 problemi come sfida per il secolo nascente.

Il **decimo problema di Hilbert** era il seguente:

“ Progettare un processo in base al quale può essere determinato in un numero finito di operazioni se un polinomio a coefficienti interi ha radici intere”

Questo problema fu risolto nel **1970** da Matijasevic che - grazie a risultati precedenti di Davis, Putnam e Robinson – provò che **non esiste un algoritmo** che risolve tale problema.

Nota: per provare che un problema è risolubile mediante un algoritmo basta esibire l'algoritmo.

Per provare che per un problema **non esiste nessun algoritmo** che lo risolve, è necessario definire in modo formale la nozione di algoritmo.

Esercizi

Discusse le soluzioni proposte sulla piattaforma elearning ai seguenti esercizi:

MdT per il successore

Progettare una macchina di Turing che calcola il successore in binario.

Esempio:

input: 01101

output: 01110

MdT a 2 nastri (1)

Sia L il linguaggio di tutte le stringhe sull'alfabeto $\{a, b\}$ che abbiano lo stesso numero di occorrenze di a e di b .

Progettare una macchina di Turing a 2 nastri che decide L .