DEFINIZIONE **1.5**

Un ***automa finito*** è una quintupla $(Q, \Sigma, \delta, q_0, F)$, dove

1. Q è un insieme finito chiamato l'insieme degli ***stati***,
2. Σ è un insieme finito chiamato l'***alfabeto***,
3. $\delta: Q \times \Sigma \longrightarrow Q$ è la ***funzione di transizione***,[1]
4. $q_0 \in Q$ è lo ***stato iniziale***, ed
5. $F \subseteq Q$ è l'***insieme degli stati accettanti***.[2]

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un automa finito e sia $w = w_1 w_2 \cdots w_n$ una stringa dove ogni w_i è un elemento dell'alfabeto Σ. Allora M ***accetta*** w se esiste una sequenza di stati $r_0, r_1, \ldots, r_n$ in Q con tre condizioni:

1. $r_0 = q_0$, primo stato = stato iniziale
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, per $i = 0, \ldots, n-1$, e se combio stato con l'input
3. $r_n \in F$. l'ultimo stato $\in$ dell'insieme di tutti gli stati

La condizione 1 afferma che la macchina inizia nello stato iniziale. La condizione 2 afferma che la macchina passa da stato a stato in base alla funzione di transizione. La condizione 3 afferma che la macchina accetta il suo input se termina la lettura in uno stato accettante. Diciamo che M ***riconosce il linguaggio*** A se $A = \{w | M \text{ accetta } w\}$.

DEFINIZIONE **1.16**

Un linguaggio è chiamato un ***linguaggio regolare*** se un automa finito lo riconosce.

TEOREMA 1.25

La classe dei linguaggi regolari è chiusa rispetto all'operazione di unione.

In altre parole, se A_1 e A_2 sono linguaggi regolari, lo è anche $A_1 \cup A_2$.

DIMOSTRAZIONE

Supponiamo che M_1 riconosca A_1, dove $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, ed
M_2 riconosca A_2, dove $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Costruiamo M che riconosce $A_1 \cup A_2$, dove $M = (Q, \Sigma, \delta, q_0, F)$.

1. $Q = \{(r_1, r_2) | r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
   Questo insieme è il ***prodotto cartesiano*** degli insiemi Q_1 e Q_2 ed è

denotato con $Q_1 \times Q_2$. È l'insieme di tutte le coppie di stati, il primo in Q_1 e il secondo in Q_2.

2. Σ, l'alfabeto, è lo stesso di M_1 ed M_2. In questo teorema e in tutti i successivi teoremi simili, assumiamo per semplicità che sia M_1 che M_2 abbiano lo stesso alfabeto di simboli di input Σ. Il teorema resta vero se hanno alfabeti diversi, Σ_1 e Σ_2. Dopo dovremmo modificare la prova ponendo $\Sigma = \Sigma_1 \cup \Sigma_2$.
3. δ, la funzione di transizione, è definita come segue. Per ogni $(r_1, r_2) \in Q$ e ogni $a \in \Sigma$, sia

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)).$$

Quindi δ riceve uno stato di M (che in realtà è una coppia di stati, uno in M_1 e uno in M_2), con un simbolo di input, e restituisce lo stato successivo di M.

4. q_0 è la coppia (q_1, q_2).
5. F è l'insieme delle coppie in cui l'uno o l'altro elemento è uno stato accettante di M_1 o M_2. Possiamo scriverlo come

$$F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ o } r_2 \in F_2\}.$$

Questa espressione è uguale a $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. (Nota che *non* è uguale a $F = F_1 \times F_2$. Cosa ci darebbe invece questo?[3])

Questo conclude la costruzione dell'automa finito M che riconosce l'unione di A_1 e A_2. Questa costruzione è piuttosto semplice, e quindi la sua corret-

---

DEFINIZIONE **1.37**

Un ***automa finito non deterministico*** è una quintupla $(Q, \Sigma, \delta, q_0, F)$, dove

1. Q è un insieme finito di stati,
2. Σ è un alfabeto finito,
3. $\delta\colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ è la funzione di transizione,
4. $q_0 \in Q$ è lo stato iniziale, ed
5. $F \subseteq Q$ è l'insieme degli stati di accettazione.

La definizione formale di computazione per un NFA è simile a quella per un DFA. Sia $N = (Q, \Sigma, \delta, q_0, F)$ un NFA e sia w una stringa sull'alfabeto Σ. Diciamo che N ***accetta*** w se possiamo scrivere w come $w = y_1 y_2 \cdots y_m$, dove ciascun y_i è un elemento di Σ_ε ed esiste una sequenza di stati $r_0, r_1, \ldots, r_m$ in Q con tre condizioni:

1. $r_0 = q_0$,
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \ldots, m-1$, ed
3. $r_m \in F$.

TEOREMA 1.39

Per ogni automa finito non deterministico esiste un automa finito deterministico equivalente.

DIMOSTRAZIONE Sia $N = (Q, \Sigma, \delta, q_0, F)$ l'NFA che riconosce un linguaggio A. Costruiamo un DFA $M = (Q', \Sigma, \delta', q_0', F')$ che riconosce A. Prima di fare la costruzione completa, consideriamo inizialmente il caso più semplice in cui N non ha ε-archi. In seguito considereremo gli ε-archi.

1. $Q' = \mathcal{P}(Q)$.
   Ogni stato di M è un insieme di stati di N. Ricorda che $\mathcal{P}(Q)$ è l'insieme dei sottoinsiemi di Q.
2. Per $R \in Q'$ e $a \in \Sigma$, sia $\delta'(R, a) = \{q \in Q | \, q \in \delta(r, a)$ per qualche $r \in R\}$.
   Se R è uno stato di M, esso è anche un insieme di stati di N. Quando M legge un simbolo a nello stato R, mostra dove a porta ogni stato in R. Poiché da ogni stato si può andare in un insieme di stati, prendiamo l'unione di tutti questi insiemi. Un altro modo per scrivere questa espressione è
   $$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).^4$$

[4] La notazione $\bigcup_{r \in R} \delta(r, a)$ significa: l'unione degli insiemi $\delta(r, a)$ per ogni possibile r in R.

3. $q_0' = \{q_0\}$.
   M inizia nello stato corrispondente alla collezione che contiene solo lo stato iniziale di N.
4. $F' = \{R \in Q' | \, R$ contiene uno stato accettante di $N\}$.
   La macchina M accetta se uno dei possibili stati in cui N potrebbe essere a quel punto è uno stato accettante.

Ora dobbiamo considerare gli ε-archi. Per farlo, introduciamo qualche ulteriore notazione. Per ogni stato R di M, definiamo $E(R)$ come la collezione di stati che possono essere raggiunti dagli elementi di R proseguendo solo con ε-archi, includendo gli stessi elementi di R. Formalmente, per $R \subseteq Q$ sia

$$E(R) = \{q | \, q \text{ può essere raggiunto da } R \text{ attraverso 0 o più } \varepsilon\text{-archi}\}.$$

Poi modifichiamo la funzione di transizione di M ponendo dita supplementari su tutti gli stati che possono essere raggiunti proseguendo attraverso ε-archi dopo ogni passo. Sostituendo $\delta(r, a)$ con $E(\delta(r, a))$ realizziamo questo effetto. Quindi

$$\delta'(R, a) = \{q \in Q | \, q \in E(\delta(r, a)) \text{ per qualche } r \in R\}.$$

Un linguaggio è regolare se e solo se qualche automa finito non deterministico lo riconosce.

TEOREMA 1.45

La classe dei linguaggi regolari è chiusa rispetto all'operazione di unione.

DIMOSTRAZIONE

Sia $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ che riconosce A_1 ed
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ che riconosce A_2.

Costruiamo $N = (Q, \Sigma, \delta, q_0, F)$ per riconoscere $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   Gli stati di N sono tutti gli stati di N_1 ed N_2, con l'aggiunta di un nuovo stato iniziale q_0.
2. Lo stato q_0 è lo stato iniziale di N.
3. L'insieme degli stati accettanti $F = F_1 \cup F_2$.
   Gli stati accettanti di N sono tutti gli stati accettanti di N_1 ed N_2. In questo modo, N accetta se N_1 accetta o N_2 accetta.
4. Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ e } a = \varepsilon \end{cases}$$

TEOREMA 1.47

La classe dei linguaggi regolari è chiusa rispetto all'operazione di concatenatione.

DIMOSTRAZIONE

Sia $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ che riconosce A_1 ed
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ che riconosce A_2.

Costruiamo $N = (Q, \Sigma, \delta, q_1, F_2)$ per riconoscere $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$.
   Gli stati di N sono tutti gli stati di N_1 ed N_2.
2. Lo stato q_1 è uguale allo stato iniziale di N_1.
3. Gli stati accettanti F_2 sono uguali agli stati accettanti di N_2.
4. Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ e } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ e } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2. \end{cases}$$

La classe dei linguaggi regolari è chiusa rispetto all'operazione star.

DIMOSTRAZIONE Sia $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ che riconosce A_1. Costruiamo $N = (Q, \Sigma, \delta, q_0, F)$ per riconoscere A_1^*.

1. $Q = \{q_0\} \cup Q_1$.
   Gli stati di N sono gli stati di N_1 più un nuovo stato iniziale.
2. Lo stato q_0 è il nuovo stato iniziale.
3. $F = \{q_0\} \cup F_1$.
   Gli stati accettanti sono i vecchi stati accettanti più il nuovo stato iniziale.
4. Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ e } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ e } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ e } a = \varepsilon \\ \emptyset & q = q_0 \text{ e } a \neq \varepsilon. \end{cases}$$

Definizione formale di espressione regolare

DEFINIZIONE **1.52**

Diciamo che R è un'***espressione regolare*** se R è

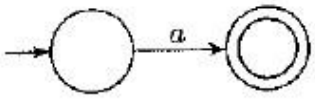
1. a per qualche a nell'alfabeto Σ,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, dove R_1 ed R_2 sono espressioni regolari,
5. $(R_1 \circ R_2)$, dove R_1 ed R_2 sono espressioni regolari, o
6. (R_1^*), dove R_1 è un'espressione regolare.

LEMMA 1.55

Se un linguaggio è descritto da un'espressione regolare, allora esso è regolare.

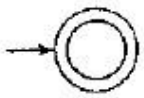
DIMOSTRAZIONE Trasformiamo R in un NFA N. Consideriamo i sei casi nella definizione di espressione regolare.

1. $R = a$ per qualche $a \in \Sigma$. Allora $L(R) = \{a\}$ e il seguente NFA riconosce $L(R)$.



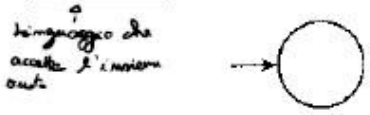
Nota che questa macchina soddisfa la definizione di NFA ma non quella di DFA perché ha qualche stato con nessun arco uscente per ogni possibile simbolo di input. Naturalmente, qui avremmo potuto presentare un DFA equivalente; ma un NFA è tutto quello di cui abbiamo bisogno per ora, ed è più facile da descrivere.
Formalmente, $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, dove descriviamo δ dicendo che $\delta(q_1, a) = \{q_2\}$ e $\delta(r, b) = \emptyset$ per $r \neq q_1$ o $b \neq a$.

2. $R = \varepsilon$. Allora $L(R) = \{\varepsilon\}$ e il seguente NFA riconosce $L(R)$.



Formalmente, $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$, dove $\delta(r, b) = \emptyset$ per ogni r e b.

3. $R = \emptyset$. Allora $L(R) = \emptyset$, e il seguente NFA riconosce $L(R)$.



Formalmente, $N = (\{q\}, \Sigma, \delta, q, \emptyset)$, dove $\delta(r, b) = \emptyset$ per ogni r e b.

4. $R = R_1 \cup R_2$.
5. $R = R_1 \circ R_2$.
6. $R = R_1^*$.

LEMMA 1.60

Se un linguaggio è regolare, allora è descritto da un'espressione regolare.

DEFINIZIONE **1.64**

Un ***automa finito non deterministico generalizzato*** è una quintupla, $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, dove

1. Q è l'insieme finito degli stati,
2. Σ è l'alfabeto di input,
3. $\delta\colon (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \longrightarrow \mathcal{R}$ è la funzione di transizione,
4. q_{start} è lo stato iniziale e
5. q_{accept} è lo stato accettante.

TEOREMA 1.70

Pumping lemma Se A è un linguaggio regolare, allora esiste un numero p (la lunghezza del pumping) tale che se s è una qualsiasi stringa in A

di lunghezza almeno p, allora s può essere divisa in tre parti, $s = xyz$, soddisfacenti le seguenti condizioni:

1. per ogni $i \geq 0$, $xy^iz \in A$,
2. $|y| > 0$, e
3. $|xy| \leq p$.

y deve essere presente tra i primi P caratteri

DIMOSTRAZIONE Sia $M = (Q, \Sigma, \delta, q_1, F)$ un DFA che riconosce A e sia p il numero di stati di M.

Sia $s = s_1 s_2 \cdots s_n$ una stringa in A di lunghezza n, dove $n \geq p$. Sia $r_1, \ldots, r_{n+1}$ la sequenza di stati attraversati da M mentre elabora s, quindi $r_{i+1} = \delta(r_i, s_i)$ per $1 \leq i \leq n$. Questa sequenza ha lunghezza $n + 1$, che è almeno $p + 1$. Due tra i primi $p + 1$ elementi nella sequenza devono essere lo stesso stato, per il principio della piccionaia. Chiamiamo il primo di questi r_j e il secondo r_l. Poiché r_l si presenta tra le prime $p + 1$ posizioni in una sequenza che inizia in r_1, abbiamo $l \leq p+1$. Ora sia $x = s_1 \cdots s_{j-1}$, $y = s_j \cdots s_{l-1}$ e $z = s_l \cdots s_n$.

Poiché x porta M da r_1 a r_j, y porta M da r_j a r_j e z porta M da r_j a r_{n+1}, che è uno stato accettante, M deve accettare xy^iz per $i \geq 0$. Sappiamo che $j \neq l$, perciò $|y| > 0$; e $l \leq p+1$, perciò $|xy| \leq p$. Quindi tutte le condizioni del pumping lemma sono rispettate.

DEFINIZIONE **2.2**

Una ***grammatica context-free*** è una quadrupla (V, Σ, R, S), dove

Variabili, Terminali, Regole, Regola Iniziale

1. V è un insieme finito i cui elementi sono chiamati ***variabili***,
2. Σ è un insieme finito, disgiunto da V, i cui elementi sono chiamati ***terminali***,

Non hanno elementi in comune

3. R è un insieme finito di ***regole***, dove ciascuna regola è una variabile e una stringa di variabili e terminali, ed
4. $S \in V$ è la variabile iniziale.

Se u, v e w sono stringhe di variabili e terminali e $A \to w$ è una regola della grammatica, diciamo che uAv ***produce*** uwv, e lo denotiamo con

$uAv \Rightarrow uwv$. Diciamo che u ***deriva*** v, e lo denotiamo con $u \stackrel{*}{\Rightarrow} v$, se $u = v$ o se esiste una sequenza $u_1, u_2, \ldots, u_k$, con $k > 0$ e

Un compilatore traduce codice scritto in un linguaggio di programmazione in un'altra forma, solitamente una più adatta per l'esecuzione. Per farlo, il compilatore estrae il significato del codice da compilare in un processo chiamato ***parsing***. Una rappresentazione di questo significato è l'albero sintattico per il codice, nella grammatica context-free per il linguaggio di

DEFINIZIONE **2.7**

Una stringa w è derivata ***ambiguamente*** in una grammatica context-free G se essa ha due o più diverse derivazioni a sinistra. Una grammatica G è ***ambigua*** se essa genera qualche stringa ambiguamente.

Forma normale di Chomsky

Quando si lavora con le grammatiche context-free, è spesso conveniente averle in forma semplificata. Una delle forme più semplici e utili è chiamata la forma normale di Chomsky. La forma normale di Chomsky è utile nel dare algoritmi che lavorano con grammatiche context-free, come facciamo nei Capitoli 4 e 7.

DEFINIZIONE **2.8**

Una grammatica context-free è in ***forma normale di Chomsky*** se ogni regola è della forma

$$A \to BC$$
$$A \to a$$

dove a è un terminale e A, B e C sono variabili qualsiasi – tranne che B e C non possono essere la variabile iniziale. Inoltre, permettiamo la regola $S \to \varepsilon$, dove S è la variabile iniziale.

TEOREMA 2.9

Ogni linguaggio context-free è generato da una grammatica context-free in forma normale di Chomsky.

DEFINIZIONE **2.13**

Un ***automa a pila*** è una sestupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$, dove Q, Σ, Γ ed F sono tutti insiemi finiti, e

1. Q è l'insieme degli stati,
2. Σ è l'alfabeto dell'input,
3. Γ è l'alfabeto della pila,
4. $\delta: Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \longrightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ è la funzione di transizione,
5. $q_0 \in Q$ è lo stato iniziale, ed
6. $F \subseteq Q$ è l'insieme degli stati accettanti.

TEOREMA 2.20

Un linguaggio è context free se e solo se esiste un automa a pila che lo riconosce.

TEOREMA 2.34

Il pumping lemma per i linguaggi context-free Se A è un linguaggio context-free, allora esiste un numero p (la lunghezza del pumping) tale che, se s è una qualsiasi stringa in A di lunghezza almeno p, allora s può essere divisa in cinque parti $s = uvxyz$ che soddisfano le condizioni

1. per ogni $i \geq 0$, $uv^ixy^iz \in A$,
2. $|vy| > 0$ e
3. $|vxy| \leq p$.

1. Una macchina di Turing può sia scrivere che leggere sul nastro.
2. La testina di lettura-scrittura può muoversi sia verso sinistra che verso destra.
3. Il nastro è infinito.
4. Gli stati speciali di accettazione e rifiuto hanno effetto immediato.

DEFINIZIONE **3.3**

Una ***macchina di Turing*** è una 7-tupla, $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$, dove Q, Σ, Γ sono tutti insiemi finiti e

1. Q è l'insieme degli stati,
2. Σ è l'alfabeto di input non contenente il simbolo ***blank*** $\sqcup$,
3. Γ è l'alfabeto del nastro, con $\sqcup \in \Gamma$ e $\Sigma \subseteq \Gamma$,
4. $\delta: Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$ è la funzione di transizione,
5. $q_0 \in Q$ è lo stato iniziale,
6. $q_{\text{accept}} \in Q$ è lo stato di accettazione e
7. $q_{\text{reject}} \in Q$ è lo stato di rifiuto, con $q_{\text{reject}} \neq q_{\text{accept}}$.

Definizione formale di macchina di Turing

Il cuore della definizione di una macchina di Turing è la funzione di transizione δ perché essa ci dice come la macchina effettua un passo. Per una macchina di Turing, δ prende la forma $Q \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$. Cioè,

Un'impostazione di questi tre elementi è chiamata una ***configurazione*** della macchina di Turing. Le configurazioni sono spesso rappresentate in modo speciale. Per uno stato q e due stringhe u e v sull'alfabeto Γ del nastro, scriviamo $u\,q\,v$ per indicare la configurazione, dove lo stato corrente è q, il contenuto corrente del nastro è uv e la posizione attuale della testina è il primo simbolo di v. Dopo l'ultimo simbolo di v, il nastro contiene solo simboli blank. Per esempio, $1011q_701111$ rappresenta la configurazione in cui il nastro contiene 101101111, lo stato corrente è q_7 e la testina è posizionata sul secondo 0. La figura 3.4 mostra una macchina di Turing con tale configurazione.

equivalentemente definire la funzione di transizione in una forma più complicata $\delta: Q' \times \Gamma \longrightarrow Q \times \Gamma \times \{\text{L}, \text{R}\}$, dove Q' è Q senza q_{accept} e q_{reject}. Una macchina di Turing M ***accetta*** l'input w se esiste una sequenza di configurazioni $C_1, C_2, \ldots, C_k$ tale che

1. C_1 è la configurazione iniziale di M su input w,
2. ogni C_i produce C_{i+1} e
3. C_k è una configurazione di accettazione.

L'insieme di stringhe che M accetta rappresenta il ***linguaggio di M***, o il ***linguaggio riconosciuto da M***, denotato con $L(M)$.

DEFINIZIONE **3.5**

Un linguaggio si dice ***Turing-riconoscibile*** se esiste una macchina di Turing che lo riconosce.[1]

Quando attiviamo una macchina di Turing su un input, tre risultati sono possibili. La macchina può *accettare*, *rifiutare*, oppure andare in *loop* (ciclare). Per ***loop*** si intende semplicemente che la macchina non si ferma. Un ciclo, in quanto tale, comporta un qualche comportamento semplice o complesso che non porta ad uno stato di arresto. Una macchina di Turing M può non accettare un input sia raggiungendo lo stato q_{reject} e rifiutando oppure andando in loop. A volte distinguere una macchina che è entrata in un loop da una che sta semplicemente impiegando molto tempo risulta difficile. Per questo motivo preferiamo macchine di Turing che si fermano su ogni singolo input; tali macchine non ciclano mai. Una tale macchina è detta ***decisore*** perché prende in ogni caso una decisione, sia essa di accettare o di rifiutare. Diciamo che un decisore ***decide*** un certo linguaggio se riconosce tale linguaggio.

DEFINIZIONE **3.6**

Un linguaggio si dice ***Turing-decidibile*** o semplicemente ***decidibile*** se esiste una macchina di Turing che lo decide.

In questo diagramma di stato, nella transizione da q_1 a q_2 compare l'etichetta 0→␣,R. Questa etichetta indica che quando siamo nello stato q_1 e la testina legge 0, la macchina si porta nello stato q_2, scrive ␣ e sposta la testina a destra. In altre parole, $\delta(q_1,0) = (q_2,␣,\text{R})$. Per chiarezza utilizziamo l'abbreviazione 0→R per indicare la transizione da q_3 a q_4, il che significa che la macchina si sposta verso destra durante la lettura di 0 quando si trova nello stato q_3 ma non modifica il nastro, così $\delta(q_3,0) = (q_4,0,\text{R})$.

Macchina di Turing multinastro

Una ***macchina di Turing multinastro*** è come una normale macchina di Turing con vari nastri. Ogni nastro ha la sua testina per la lettura e la scrittura. Inizialmente l'input si trova sul nastro 1, mentre gli altri nastri sono vuoti. La funzione di transizione viene modificata per consentire la lettura, la scrittura e lo spostamento della testina contemporaneamente su alcuni o tutti i nastri. Formalmente,

$\delta: Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{\text{L},\text{R},\text{S}\}^k$, dove k è il numero di nastri. L'espressione

$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, \text{L}, \text{R}, \ldots, \text{L})$

significa che, se la macchina si trova nello stato q_i e le testine da 1 a k leggono i simboli da a_1 ad a_k, la macchina va nello stato q_j, scrive i simboli da b_1 a b_k e muove ogni testina a sinistra o a destra, o la fa restare ferma, come specificato. Le macchine di Turing multinastro sembrano più potenti delle macchine di Turing ordinarie, ma possiamo dimostrare che sono equivalenti in potenza di calcolo. Ricordiamo che due macchine sono equivalenti se riconoscono lo stesso linguaggio.

TEOREMA 3.13

Per ogni macchina di Turing multinastro esiste una macchina di Turing a nastro singolo equivalente.

DIMOSTRAZIONE. Mostriamo come convertire una macchina di Turing multinastro M in una TM S equivalente a nastro singolo. L'idea chiave è quella di mostrare come simulare M con S. Supponiamo che M abbia k nastri. Allora S simula l'effetto di k nastri memorizzando le loro informazioni sul suo singolo nastro. Essa utilizza il nuovo simbolo # come delimitatore per separare i contenuti dei diversi nastri. Oltre al contenuto dei nastri, S deve tenere traccia delle posizioni delle testine. Lo fa scrivendo un simbolo con un punto sopra per contrassegnare la posizione in cui si troverebbe la testina di quel nastro. Pensate a questi come nastri e testine "virtuali". Come in precedenza, i simboli del nastro "puntati" sono

S = "Su input $w = w_1 \cdots w_n$:

1. Inizialmente S mette il suo nastro nel formato che rappresenta tutti i k nastri di M. Il nastro formattato contiene
$$\#\dot{w}_1 w_2 \cdots w_n \#\dot{\sqcup}\#\dot{\sqcup}\# \cdots \#.$$
2. Per simulare una singola mossa, S scansiona il suo nastro dal primo #, che segna l'estremità sinistra, al $(k+1)$mo #, che segna l'estremità di destra, per determinare i simboli puntati dalle testine virtuali. Successivamente S fa un secondo passaggio per aggiornare i nastri in accordo alla funzione di transizione di M.
3. Se in qualsiasi momento S sposta una delle testine virtuali a destra su un #, questa azione significa che M ha spostato la testina corrispondente sulla parte di nastro vuota non letta in precedenza. Quindi S scrive un simbolo blank in questa cella del nastro e sposta il contenuto del nastro, da questa cella fino al simbolo # più a destra, di una unità a destra. Poi prosegue la simulazione come prima."

COROLLARIO 3.15

Un linguaggio è Turing-riconoscibile se e solo se qualche macchina di Turing multinastro lo riconosce.

Macchina di Turing non deterministica

Una macchina di Turing non deterministica è definita come ci si aspetta. In qualsiasi punto della computazione la macchina può procedere effettuando varie scelte. La funzione di transizione di una macchina di Turing non deterministica è della forma

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

La computazione di una macchina di Turing non deterministica è un albero i cui rami corrispondono alle diverse scelte possibili previste per la macchina. Se qualche ramo della computazione porta allo stato di accettazione, allora la macchina accetta l'input. Se sentite la necessità di rivedere il non determinismo, andate alla Sezione 1.2 (pagina 50). Ora mostriamo che il non determinismo non influisce sulla potenza di calcolo del modello macchina di Turing.

TEOREMA 3.16

Per ogni macchina di Turing non deterministica esiste una macchina di Turing deterministica equivalente.

DIMOSTRAZIONE La TM D deterministica che simula ha tre nastri. Dal Teorema 3.13, questa disposizione equivale ad avere un singolo nastro. La macchina D utilizza i suoi tre nastri in maniera particolare, come illustrato nella figura seguente. Il nastro 1 contiene sempre la stringa di input e non viene mai modificato. Il nastro 2 mantiene una copia del nastro di N corrispondente a qualche diramazione della sua computazione non deterministica. Il nastro 3 tiene traccia della posizione di D nell'albero delle computazioni di N.

COROLLARIO 3.18

Un linguaggio è Turing-riconoscibile se e solo se esiste una macchina di Turing non deterministica che lo riconosce.

DIMOSTRAZIONE. Qualsiasi TM deterministica è automaticamente una TM non deterministica e quindi una direzione di questo teorema è già dimotrata. L'altra direzione segue dal Teorema 3.16.

COROLLARIO 3.19

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

Enumeratori

Come accennato in precedenza, alcune persone usano il termine *linguaggio ricorsivamente enumerabile* per indicare un linguaggio Turing-riconoscibile. Questo termine deriva da una variante di macchina di Turing chiamata ***enumeratore***. Definito in modo informale, un enumeratore è una macchina di Turing con una stampante collegata. La macchina di Turing può

Un enumeratore E inizia con un nastro di input vuoto. Se l'enumeratore non si ferma, esso può stampare un elenco infinito di stringhe. Il linguaggio enumerato da E è la collezione di tutte le stringhe che esso stampa. Inoltre,

TEOREMA 3.21

Un linguaggio è Turing-riconoscibile se e solo se esiste un enumeratore che lo enumera.

DIMOSTRAZIONE. Come prima cosa, dimostriamo che se abbiamo un enumeratore E che enumera un linguaggio A, allora esiste una TM M che riconosce A. La TM M funziona come segue.

M = "Su input w:

1. Esegue E. Ogni volta che E genera una stringa, la confronta con w.
2. Se w appare nell'output di E, *accetta*."

Chiaramente, M accetta quelle stringhe che compaiono sulla lista di E.

Ora occupiamoci dell'altra direzione. Se la TM M riconosce un linguaggio A, possiamo costruire il seguente enumeratore E per A. Sia $s_1, s_2, s_3, \ldots$ una lista di tutte possibili stringhe in Σ^*.

E = "Ignora l'input.

1. Ripete i seguenti passi per $i = 1, 2, 3, \ldots$.
2. Esegue M per i passi su ogni input, $s_1, s_2, \ldots, s_i$.
3. Se una qualche computazione accetta, stampa la corrispondente s_j."

Se M accetta una particolare stringa s, alla fine s apparirà nella lista generata da E. In realtà, essa apparirà nella lista un numero infinito di volte, perché M ritorna all'inizio su ogni stringa per ogni ripetizione del passo 1.