Teoriaprimaparteetc - Riassunto Elementi di Teoria della Computazione

Elementi di Teoria della Computazione (Università degli Studi di Salerno)

TEORIA DFA

Definizione formale di DFA

A automa finito deterministico (DFA) è una 5-tupla $(Q, \sum, f, q_1, F)$, dove:

1. Q è un insieme finito chiamato insieme degli stati;
2. $\sum$ è un insieme finito chiamato alfabeto;
3. F: Q x $\sum \rightarrow$ Q è la funzione di transizione;
4. $q_1 \in Q$ è lo stato iniziale;
5. $F \subseteq Q$ è l'insieme degli stati accettati.

Linguaggio della macchina

Se L è l'insieme di tutte le stringhe che la macchina M accetta, allora si dice che:

- L è il linguaggio della macchina M;
- M riconosce (o accetta) L.

Scriviamo anche L(M)=L.

Un linguaggio è regolare se è riconosciuto da qualche DFA. Una macchina può accettare diverse stringhe, ma riconosce sempre e soltanto un linguaggio . Se la macchina non riconosce alcuna stringa, essa riconosce comunque un linguaggio, ossia il linguaggio vuoto ($\oslash$).

Come un DFA accetta una stringa

Sia $M=(Q, \sum, f, q_0, F)$ un DFA e sia $w=w_1 \dots w_n$ una stringa dove w_i con $1 \leq i \leq n$ è un simbolo dell'alfabeto $\sum$. Allora M accetta w se esiste una sequenza di stati $r_0, r_1, \dots, r_n$ con tre condizioni:

1. $r_0 = q_o$;
2. $f(r_i, w_{i+1})=r_{i+1}$, con i=0, …, n-1;
3. $r_n \in F$.

Complemento

Se linguaggio L su alfabeto $\sum$ ha un DFA $M=(Q, \sum, f, q_1, F)$, allora il DFA per il complemento di L è :

$M'=(Q, \sum, f, q_1, Q-F)$

Operazioni regolari

Siano A e B linguaggi, definiamo le operazioni regolari unionme, concatenazione e star:

- Unione: $A \cup B = \{x \mid x \in A \text{ o } x \in B\}$;
- Concatenazione: $AB = \{xy \mid x \in A \text{ e } y \in B\}$;
- Star: $A^* = \{x_0, \ldots, x_k \mid k \geq 0 \text{ e } x_i \in A \quad \forall i \geq 0\}$.

Nota: la stringa vuota è sempre un elemento di A*.

Classe chiusa rispetto una operazione

Una classe di oggetti è chiusa rispetto ad una operazione, se l'applicazione di tale operazione a elementi della classe, restituisce un oggetto ancora nella classe. La classe dei linguaggi regolari è chiusa rispetto a tutte e tre le operazioni regolari.

Teorema

L'insieme dei linguaggi regolari è chiuso per l'operazione di complemento.

Dim.

Dato un DFA M_1 per linguaggio L, possiamo costruire un DFA M_2 per il linguaggio complemento L':

- Rendiamo tutti gli stati accetta in M_1, non accetta in M_2;
- Rendiamo tutti gli stati non accetta in M_1, accetta in M_2 .

Teorema

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

Dim.

Supponiamo che M_1 riconosca A_1, dove $M_1 = (Q_1, \sum, f_1, q_1, F_1)$ ed M_2 riconosca A_2 , dove $M_2 = (Q_2, \sum, f_2, q_2, F_2)$. Costruiamo M in modo che riconosca $A_1 \cup A_2$,

$M = (Q, \sum, f, q_0, F)$, dove :

- $Q = \{(r_1, r_2) \mid r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$;
- $\sum$ è l'alfabeto dell'automa M;
- $f((r_1, r_2), a) = (f_1(r_1, a), f_2(r_2, a))$;
- q_0 è la coppia (q_1, q_2);
- $F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ o } r_2 \in F_2\}$

M riconosce $A_1 \cup A_2$, quindi $A_1 \cup A_2$ è un linguaggio regolare, poiché riconosciuto da un automa.

Teorema

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

Dim.

Supponiamo che M_1 riconosca A_1, dove $M_1=(Q_1, \sum, f_1, q_1, F_1)$ ed M_2 riconosca A_2 , dove $M_2=(Q_2, \sum, f_2, q_2, F_2)$. Costruiamo M in modo che riconosca $A_1 \cap A_2$,

$M=(Q, \sum, f, q_0, F)$, dove :

- $Q=\{ (r_1, r_2) \mid r_1 \in Q_1$ and $r_2 \in Q_2\}$;
- $\sum$ è l'alfabeto dell'automa M;
- $f((r_1, r_2), a)= (f_1(r_1, a) ,f_2(r_2, a))$;
- q_0 è la coppia (q_1, q_2);
- $F= \{ (r_1, r_2) \mid r_1 \in F_1$ and $r_2 \in F_2 \}$

M riconosce $A_1 \cap A_2$, quindi $A_1 \cap A_2$ è un linguaggio regolare, poiché riconosciuto da un automa.

TEORIA NFA

Definizione formale NFA

Un automa finito non deterministico è una quintupla $(Q, \sum, f, q_0, F)$, dove :

1. Q è un insieme finito di stati;
2. $\sum$ è un alfabeto finito;
3. $f: Q \times \sum_\varepsilon \rightarrow P(Q)$ è la funzione di transizione;
4. $q_0 \in Q$ è lo stato iniziale;
5. $F \subseteq Q$ è l'insieme degli stati di accettazione .

Computazione di un NFA

Sia $N=(Q, \sum, f, q_0, F)$ un NFA e sia $w=w_1 \dots w_n$ una stringa, dove w_i è un simbolo dell'alfabeto $\sum_\varepsilon$. Allora M accetta w se esiste una sequenza di stati $r_o, \dots, r_n$ in Q con tre condizioni:

1. $r_0 = q_0$;
2. $r_{i+1} \in f(r_i, w_{i+1})$ con i=0, …,n-1;
3. $r_n \in F$

Definizione macchine equivalenti

Due macchine sono equivalenti se esse riconoscono lo stesso linguaggio.

Teorema

Ogni NFA N ha un equivalente DFA M, cioè , se N è un NFA, allora esiste un DFA M tale che L(N)=L(M).

Dim.

Un DFA è anche un NFA senza ε-transition, dimostriamo il viceversa.

Sia L=L(N) per NFA $N=(Q_N, \sum, f_N, q_N, F_N)$, costruiamo un DFA $D=(Q_D, \sum, f_D, q_D, F_D)$. Consideriamo:

E(R)= R ∪ { q| q stato raggiungibile da stato R con 1 o più ε-archi}.

Definiamo D come:

- Q_D= P(Q_N);
- $f_D(R, a)= ¿ r \in R\, E(f_N(r, a)) \forall R \in Q_D e a \sum$;
- q_D= E({q_n});
- F_D= {R∈ Q_D | R ∩ F_N ≠ ⊘ }.

Nota:

$f^*_N(q_N, x)$= insieme di stati raggiungibili da q_N con input x;

$f^*_N(q_N, ε)$= E({ q_N});

$f^*_N(q_N,xa)= ¿ r \in f^{¿}_N(q_N, x)\, E(f_N(r, a))$

Ipotesi induttiva:

$f^*_D(\{q_o\},w)=f^*_N(q_0, w)$, assumiamo che $q_{0=}q_N$

Base:

w=ε, banalmente vero, poiché q_0=E({q_N})

Passo induttivo:

$f^*_D(\{q_0 \},wa)=f_D(f^*_D(\{q_o\}, w) , a)$ = (per definizione)

= $f_D(f^*_N(q_0, w), a)$ = (per ipotesi induttiva)

= $¿ r\, inf^{¿}_N(q_0, w)\, E(f_N(r, a))$= (per definizione)

= $f^*_N(q_0, wa)$

Corollario

Il linguaggio L è regolare se e solo se esiste un NFA che riconosce L

Dim.

Se L è regolare, allora esiste un DFA che lo riconosce, ma ogni DFA è anche un NFA, quindi esiste un NFA per L.

Se esiste NFA per L, allora esiste anche un DFA per il teorema di uguaglianza fra DFA e NFA, quindi L è regolare.

Teorema

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

Dim.

Sia N_1= (Q_1, ∑, f_1, q_1, F_1) che riconosce A_1,

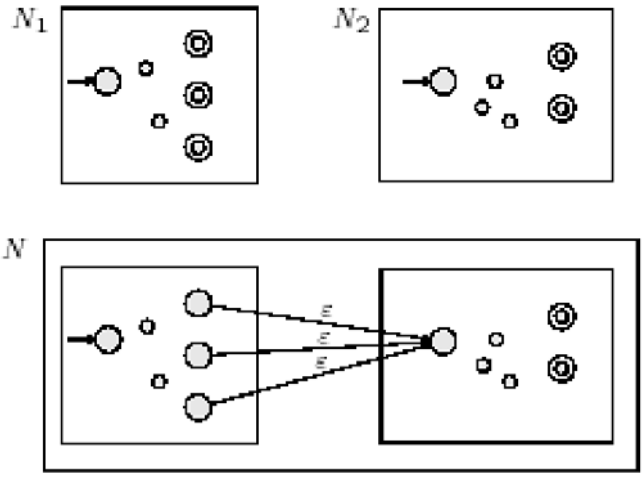
N_2=(Q_2, ∑, f_2, q_2, F_2) che riconosce A_2.

Costruiamo N=(Q, ∑, f, q_1, F_2) un NFA per riconoscere A_1A_2, allora risulta:

- $Q=Q_1 \cup Q_2$;
- Definiamo f in modo che $\forall\, q \in Q$ e $a \in \sum_\varepsilon$, risulti:

$$f(q,a)=\begin{cases} f_1(q,a) \text{ if } q \in Q_1 \wedge q \notin F_1 \\ f_1(q,a) \text{ if } q \in F_1 \wedge a \neq \varepsilon \\ f_1(q,a) \cup \{q_2\} \text{ if } q \in F_1 \wedge a = \varepsilon \\ f_2(q,a) \text{ if } q \in Q_2 \end{cases}$$

N riconosce A_1A_2, quindi A_1A_2 è un linguaggio regolare.



Teorema

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

Dim.

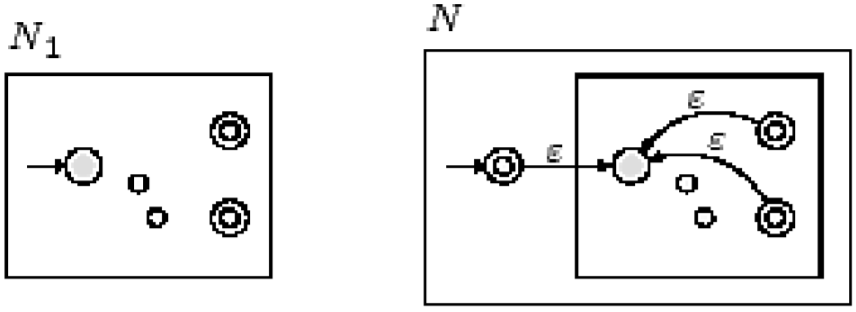
Sia $N_1= (Q_1, \sum, f_1, q_1, F_1)$ che riconosce A_1,

Costruiamo $N=(Q, \sum, f, q_1, F_2)$ un NFA per riconoscere A^*_1 , allora risulta:

- $Q=\{q_0\} \cup Q_1$;
- $F=\{ q_0\} \cup F_1$;
- Definiamo f in modo che $\forall$ q$\in$ Q e a$\in \sum_\varepsilon$, risulti:

$$f(q,a)=\begin{cases} f_1(q,a) \text{ if } q\in Q_1 \wedge q\notin F_1 \\ f_1(q,a) \text{ if } q\in F_1 \wedge a\neq\varepsilon \\ f_1(q,a)\cup\{q_0\} \text{ if } q\in F_1 \wedge a=\varepsilon \\ q1 \text{ if } q=q_0 \wedge a=\varepsilon \\ \text{if } q=q_0 \wedge a\neq\varepsilon \end{cases}$$

N riconosce A_1*, quindi A_1* è un linguaggio regolare.



TEORIA ESPRESSIONI REGOLARI

Definizione induttiva di espressione regolare

Base:

ε e ⊘ sono espressioni regolari: L(ε)={ε} e L(⊘)=⊘, se a ∈∑, allora a è un'espressione regolare L(a)={a}.

Ipotesi induttiva:

R_1 e R_2 sono espressioni regolari.

Passo induttivo:

- R_1 U R_2 è un'espressione regolare che rappresenta il linguaggio $L(R_1 \cup R_2)= L(R_1) U L(R_2)$;
- R_1R_2 è un'espressione regolare che rappresenta il linguaggio $L(R_1R_2)= L(R_1)L(R_2)$;
- R_1* è un'espressione regolare che rappresenta il linguaggio $(L(R_1)^*)= (L(R_1))^*$

Nota: Non bisogna confondere le espressioni regolari ε e ⊘. L'espressione ε rappresenta il linguaggio che contiene una sola stringa, ossia, la stringa vuota. Mentre ⊘ rappresenta il linguaggio che non contiene alcuna stringa.

Teorema di Kleene

Il linguaggio L è regolare se e solo se L ammette un'espressione regolare.

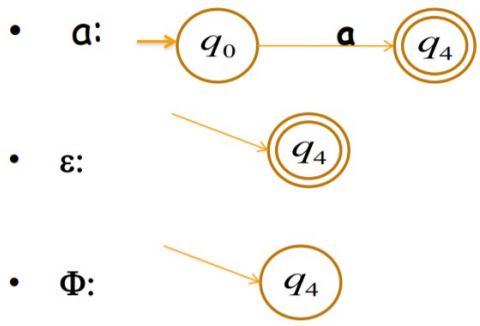
Dim.

Bisogna dimostrare che:

1. Per ogni DFA A possiamo costruire una espressione regolare R con L(R)=L(A);
2. Per ogni espressione regolare R esiste un NFA A tale che L(R)=L(A).

Partiamo dal dimostrare il punto 2.

Base: Dimostriamo per le espressioni regolari dei casi base a, ε, ⊘ che possono essere riconosciute da un NFA.



Ipotesi induttiva:

Assumiamo la correttezza per le espressioni regolari R_1 e R_2, quindi:

- ∃ un NFA N_1 tale che $L(N_1)=L(R_1)$;
- ∃ un NFA N_2 tale che $L(N_2)=L(R_2)$.

Passo induttivo:

- $R=R_1 \cup R_2$, per definizione induttiva risulta: $L(R)=L(R_1 \cup R_2)= L(R_1) \cup L(R_2)$.
  ∃ un NFA N tale che $L(N)=L(N_1) \cup L(N_2)= L(R_1) \cup L(R_2)= L(R)$;
- $R=R_1R_2$, per definizione induttiva risulta: $L(R)=L(R_1R_2)= L(R_1)L(R_2)$.
  ∃ un NFA N tale che $L(N)=L(N_1)L(N_2)= L(R_1)L(R_2)= L(R)$;
- $R=R_1^*$, per definizione induttiva risulta: $L(R)=[L(R_1)^*]= [L(R_1)]^*$.
  ∃ un NFA N tale che $L(N)=[L(N_1)]^*= [L(R_1)^*]= L(R)$.

Ora dimostriamo il punto 1.

Procedura di conversione dei DFA in espressione regolari. La procedura viene divisa in due parti, nella prima viene mostrato come trasformare un DFA in un GNFA e poi come trasformare un GNFA in un'espressione regolare. I GNFA sono una variazione di NFA, nei quali gli archi delle transizioni possono avere espressioni regolari come etichette. Tali GNFA possono leggere blocchi di simboli alla volta anziché uno alla volta. Per comodità assumiamo che :

- Lo stato iniziale ha archi di transizione uscenti verso un qualsiasi altro stato, ma nessun arco entrante;
- Esiste un solo stato accettante, ed esso ha archi entranti provenienti da qualsiasi altro stato, ma nessun arco uscente. Inoltre lo stato accettante non è uguale allo stato iniziale.

Possiamo facilitare quindi la costruzione dell'GNFA aggiungendo un nuovo stato iniziale con un ε-arco che entra nel vecchio stato iniziale e un nuovo stato accettante con ε-archi entranti provenienti dai vecchi stati di accettazione.

Ora passiamo alla seconda fase della dimostrazione, quella in cui si trasforma un GNFA in un'espressione regolare.

Riduciamo il numero di stati da iterare finché il numero di stati è k>2 (sempre garantito per la considerazione di prima, nel creare due stati di inizio e di accettazione diversi fra loro), nel seguente modo:

1. Selezioniamo uno stato, isolando e sistemando il resto in modo che lo stesso linguaggio sia ancora riconosciuto. Possiamo rimuovere qualsiasi stato, a condizione che non sia lo stato start o accetta. Abbiamo garanzia che tale stato esista poiché k>2. Chiamiamo lo stato rimosso "q_{rip}";
2. Dopo aver rimosso q_{rip} modifichiamo la macchina cambiando le espressioni regolari che etichettano le restanti frecce. Le nuove etichette controbilanciano l'assenza di q_{rip} integrando le computazioni perse. La nuova etichetta che va dallo stato q_i a q_j è una espressione regolare che descrive tutte le stringhe che porterebbero la macchina da q_i a q_j direttamente o tramite q_{rip}.

Pumping Lemma

Il teorema del pumping lemma afferma che tutti i linguaggi regolari hanno una proprietà speciale. Se noi possiamo mostrare che un linguaggio non ha questa proprietà, siamo sicuri che esso non sia regolare. La proprietà afferma che tutte le stringhe nel linguaggio possono essere "replicate" se la loro lunghezza raggiunge almeno un valore speciale , chiamato lunghezza del pumping . Questo significa che ogni stringa contiene una parte che può essere ripetuta un numero qualsiasi di volte, ottenendo una stringa che appartiene ancora al linguaggio. Enunciamo ora il teorema:

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. $\forall\, i > 0,\ xy^iz \in A$;
2. $|y| > 0$;
3. $|xy| \leq p$.

Ricorda: $|s|$ rappresenta la lunghezza della stringa s, y^i indica i copie di y concatenate insieme e y^0 è uguale a ε. Quando s è diviso in xyz, x o z potrebbero

essere ε, ma la condizione 2 dice che y≠ε. La condizione 3 afferma che le parti x e y messe insieme, hanno al più la lunghezza di p.

Riassumendo:

Siano: M automa che riconosce L e p=numero stati di M.

$|w| \geq p$ dove w è una stringa del linguaggio L. Di solito prendiamo $|w|>p$, poiché vogliamo che almeno uno stato r si ripeta.

Per il pumping lemma, possiamo dividere w in tre parti:

- x porta da stato iniziale a r;
- y da r a r (cicla);
- z da ra a stato finale.

Bisogna rispettare le condizioni ora (se abbiamo contraddizione, possiamo dire che il linguaggio non è regolare), quindi deve sussistere che:

1. $|xy|<p$ (r primo stato che si ripete);
2. $|y|>0$ (non deve esserci stringa vuota sul ciclo);
3. xy^iz (porta da stato iniziale a r, da r a r per i volte, e infine da r a stato finale).

TEORIA MACCHINA DI TURING

Definizione macchina di turing

Una macchina di Turing è una 7-upla $(Q, \sum, \Gamma, \delta, q_0, q_{accept}, q_{reject})$,dove Q, ∑, Γ sono tutti insiemi finiti e risulta che:

1. Q è l'insieme di stati;
2. ∑ è l'alfabeto di input non contente il simbolo blank ("␣");
3. Γ è l'alfabeto del nastro, con ␣ ∈ Γ e ∑ ⊆ Γ;
4. δ: Q x Γ → Q x Γ x {L,R} è la funzione di transizione;
5. $q_0 \in Q$ è lo stato iniziale;
6. q_{accept} è lo stato d'accettazione;
7. q_{reject} è lo stato di rifiuto, nota: $q_{reject} \neq q_{accept}$.

Configurazioni macchine di Turing

Per uno stato q e due stringhe u e v sull'alfabeto Γ del nastro, scriviamo uqv 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.

Si dice che la configurazione C_1 produce la configurazione C_2, se la macchina di Turing può passare da C_1 a C_2 in un unico passo. Formalmente, supponiamo di avere:

- a, b, c $\in \Gamma$;
- u, v $\in \Gamma *$;
- qi , qj $\in$ Q.

Definiamo:

- uaqibv produce uqjacv se δ(qi , b) = (qj , c, L) Mossa a sinistra
- uaqibv produce uacqjv se δ(qi , b) = (qj , c, R) Mossa a destra

Tipologia di configurazioni

Esistono vari tipi di configurazioni:

- Configurazione di start di M su input w, è la configurazione q_0w che indica che la macchina è nello stato iniziale q_0 con la posizione della testina più a sinistra del nastro;
- Configurazione di accept se lo stato della configurazione è q_{accept};
- Configurazione di reject se lo stato della configurazione è q_{reject};
- Configurazione di Halt, qualsiasi configurazione accept o reject. Importante per i loop che non raggiungono mai uno stato accept o reject.

Come computa la macchina di Turing

Una macchina di Turing M accetta l'input w se esiste una sequenza di configurazioni tale che:

1. C_1 è la configurazione iniziale di M su input w;
2. Ogni C_i produce C_{i+1} $\forall$ i=1,…, n-1;
3. C_n è una configurazione accept.

Nota: δ(q,a)=(r,b,d) dove:

- r$\in$Q;
- b$\in\Gamma$ (il simbolo scritto dalla testina sulla cella del nastro su cui la testina si trova all'inizio della transizione;
- d$\in${L,R} (è la direzione in cui la testina muove un passo).

Definizione Turing riconoscibile

Un linguaggio si dice Turing riconoscibile se esiste una macchina di Turing che lo riconosce, ovvero le stringhe del linguaggio sono accettate da tale macchina.

Definizione Turing decidibile

Un linguaggio è Turing decidibile se esiste una macchina di Turing che lo decide. Tale macchina non accetta altre stringhe oltre quelle del linguaggio che decide.

Nota: La differenza fra turing decidibile e turing riconoscibile non sta nell'accettazione, bensì sulle stringhe che non appartengono al linguaggio.

Ricorda: L'input per una macchina di Turing è sempre una stringa. Se vogliamo dare in input altri oggetti, questi devono essere codificati come stringhe. Rappresenteremo i problemi di decisione mediante linguaggi.

Stayer

Macchina di Turingg in cui la testina può restare sulla stessa cella del nastro durante una transizione. Quindi la definizione formale resta la stessa, cambia soltanto la funzione di transizione che diventa: δ: Q x Γ →Q x Γ x {L, S, R}, dove S serve ad indicare che la testina resta ferma.

Potere computazionale

Il potere computazionale di due modelli è lo stesso se riconoscono la stessa classe di linguaggi.

Macchina di Turing multinastro

Una macchina di Turing a k nastri è una normale macchina di Turing avente K 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. Formalmente nella definizione cambia soltanto come è definita la funzione di transizione, che diventa:

$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

Dove abbiamo k simboli letti a cui associamo uno stato, k simboli da scrivere e k movimenti delle k testine.

Teorema

Se un linguaggio è riconosciuto da una macchina di Turing, esiste uno stayer che lo riconosce, viceversa, se un linguaggio è riconosciuto da uno stayer, esiste una macchina di Turing che lo riconosce.

Dim.

La macchina di Turing può essere facilmente simulata da uno stayer, basta non usare la possibilità di non muovere la testina. Quindi il potere computazionale dello

stayer, è almeno pari al potere computazionale della macchina di Turing convenzionale.

Ora bisogna mostrare il viceversa, cioè che per ogni stayer esiste una macchina di Turing che riconosce lo stesso linguaggio. Assumiamo che M sia uno stayer e mostriamo come una macchina di Turing che denominiamo con M' simuli M. Definiamo M' come M, tranne che ogni transizione S (indica che la testina non si muove) è sostituita da due transizioni in M' :

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

Di conseguenza M e M' riconoscono lo stesso linguaggio, quindi abbiamo dimostrato che il potere computazionale della macchina di Turing è almeno pari al potere computazionale di Stayer.

Teorema

La macchina di Turing e la macchina di Turing multinastro sono modelli equivalenti.

Dim.

In un verso è ovvio, poiché la macchina di Turing multinastro con k che indica il numero di nastri uguale a uno, è una Macchina di turing.

Bisogna dimostrare il viceversa, cioè che una macchina di Turing è anche una macchina di Turing multinastro. Per fare ciò basta codificare l'informazione su un solo nastro concatenando il contenuto dei k nastri su k blocchi consecutivi, i k blocchi saranno separati da "#", quindi risulta che:

- Ogni blocco ha una lunghezza variabile che dipende dal contenuto del nastro corrispondente ;
- Un elemento marcato nel blocco i-mo indica la posizione della testina i-ma. Dato che marchiamo la posizione della testina e poiché potrebbe spostarsi su tutte le lettere dell'alfabeto, risulta un alfabeto esteso del nastro che denominiamo con Γ_2, definito come segue:
  Γ_2 tale che å $\in \Gamma_2$ $\forall$ a$\in\Gamma$.

Come computa?

La macchina di Turing scorre il nastro verso destra fino alla fine per determinare i simboli puntati dalle testine virtuali, li memorizza alla fine del nastro per capire in un secondo momento quale azioni applicarvi . Si riposiziona all'inizio del nastro e lo scorre di nuovo eseguendo su ogni sezioni le azioni che simulano quelle delle testine della macchina di Turing multinastro.

Quindi per ogni istruzione della macchina di Turing multinastro, una macchina di Turing può:

- Scorrere i k nastri e "raccogliere" informazioni sui simboli letti;
- Applicare transazioni scorrendo i k nastri e applicando su ciascun simbolo marcato scrittura e spostamento;
- Entrare nello stato che ricorda il nuovo stato della macchina di Turing multinastro e riposizionare la testina all'inizio del nastro per poter computare di nuovo.

Quindi la simulazione è possibile.

Macchina di Turing non deterministica

Una macchina di Turing non deterministica è definita come una normale macchina di Turing, ciò che cambia è la funzione di transizione, poiché può computare in diversi modi. Formalmente definiamo la funzione di transizione come segue:

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

Ogni configurazione non produce più una singola configurazione, bensì può produrne molteplici.

Teorema

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

Dim.

In un verso è semplice da dimostrare, poiché la macchina di Turing è anche una macchina di Turing non deterministica.

Dimostriamo il viceversa, guardiamo alla computazione di una macchina di Turing non deterministica N come a configurazioni raggiungibili dalla configurazione iniziale C_0 su input w. Mostriamo che esiste una macchina di Turing M che simula N. Le

computazioni di N possono essere viste come un albero, dove ogni ramo dell'albero rappresenta una scelta non deterministica ed ogni nodo dell'albero è una configurazione. La radice dell'albero è la configurazione iniziale. La macchina di Turing M essenzialmente esplora l'albero alla ricerca di una configurazione di accettazione. Per esplorare l'albero utilizza la visita in ampiezza (per evitare i loop che porterebbe la visita in profondità). Ricordiamo che la visita in ampiezza esplora tutti i cammini che terminano alla stessa profondità prima di andare a esplorare ogni cammino che termina alla profondità successiva. La macchina di Turing M ha tre nastri, che hanno la seguenti funzioni:

1. Il nastro 1 contiene sempre la stringa di input e non viene mai modificato;
2. Il nastro 2 mantiene una copia del nastro di N, su tale nastro viene applicata la computazione di qualche diramazione di N;
3. Il nastro 3 tiene traccia delle posizioni di M nell'albero delle computazioni di N.

Inizialmente il nastro 1 contiene l'input w e i nastri 2 e 3 sono vuoti .Consideriamo la rappresentazione dei dati sul nastro 3. Ogni nodo dell'albero può avere al massimo b figli, dove b è la dimensione del più grande insieme di scelte possibili date dalla funzione di transizione di N. Per ogni configurazione di N, la macchina di Turing M codifica tutte le possibili transizioni e le enumera. Ogni nodo della struttura ha un indirizzo che è una stringa sull'alfabeto Γ_b={1, …,b} , quindi il nastro 3 serve a far decidere alla macchina di Turing quale percorso scegliere nel livello k (dovrà percorrere tutte le possibili scelte alla ricerca dello stato di accettazione). Possiamo quindi riassumere la simulazione di N tramite M nel seguente modo:

1. Copia il nastro 1 sul nastro 2 ed inizializza la stringa sul nastro 3 a ε;
2. Utilizza il nastro 2 per simulare N con input w su una ramificazione della sua computazione non deterministica, consulta il simbolo successivo sul nastro 3 per determinare quale scelta fare tra quelle consentite dalla funzione di transizione di N. Se trovo una configurazione di accettazione, accetta l'input, altrimenti va alla fase 3;
3. Sostituisci la stringa sul nastro 3 con la stringa successiva rispetto all'ordine sulle stringhe (precedentemente enumerate). Se si passa al livello successivo, la stringa ha un simbolo in più, poiché la profondità aumenta di 1, quindi partendo dalla radice, bisognerà fare una scelta in più in confronto alle stringhe del livello precedente. Simula la ramificazione successiva delle computazioni andando al passo 1.

Tesi di Church-Turing

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