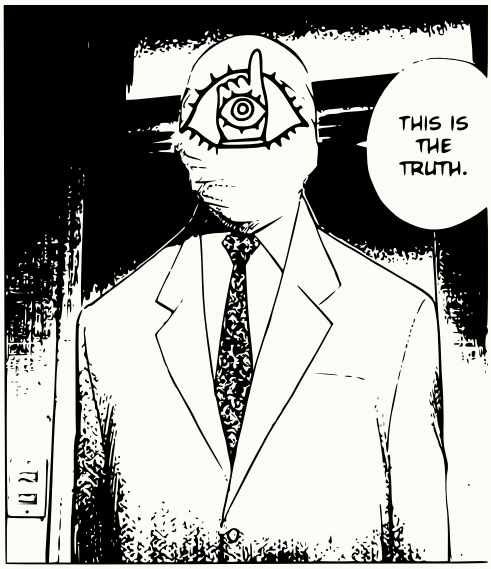
Marco Casu

Automi, Calcolabilità e Complessità

SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica

Questo documento è distribuito sotto la licenza GNU, è un resoconto degli appunti (eventualmente integrati con libri di testo) tratti dalle lezioni del corso di Automi, Calcolabilità e Complessità per la laurea triennale in Informatica. Se dovessi notare errori, ti prego di segnalarmeli.



INDICE

CAPITOLO

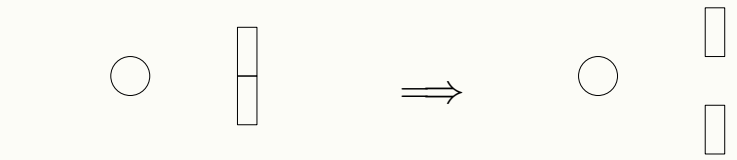
1

AUTOMI

1.1 Linguaggi Regolari

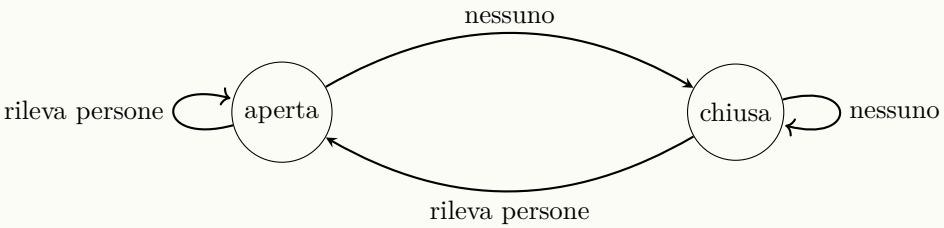
Un *automa a stati finiti* è, seppure limitato nella memoria e nella gestione dell'input, il più semplice modello di computazione. Un automa può interagire con l'input esclusivamente "scorrendolo" in maniera sequenziale.

Esempio : Si vuole modellare una semplice porta con sensore, che si apre quando qualcuno si trova nelle vicinanze.



una persona si trova davanti la porta — la porta si apre

Un automa che modella il problema è il seguente :



Un automa ha alcuni stati speciali, come quello iniziale, indicato con un apposita freccia, e degli stati detti *di accettazione*, ossia stati in cui deve necessariamente terminare la computazione per essere definita valida, vengono rappresentati con un doppio cerchio.

Il modello di calcolo degli automi è riconducibile al concetto di *linguaggio regolare*, che verrà formalizzato in seguito, segue ora una definizione formale di automa.

Definizione (DFA) : : Un DFA (Deterministic Finite Automa) è una 5-tupla, $(Q, \Sigma, \delta, q_0, F)$ di cui

- Q è l'insieme degli stati possibili
- Σ è l'alfabeto che compone le stringhe in input

- δ è una mappa $Q \times \Sigma \to Q$ detta *funzione di transizione.*
- $q_0 \in Q$ è lo stato iniziale.
- $F \subseteq Q$ è l'insieme degli stati di accettazione.

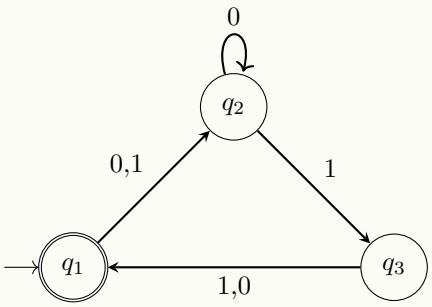


Figura 1.1: semplice automa

Nell'esempio in figura 1.1, si ha che

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $F = \{q_1\}$
- $q_0 = q_1$
- $\delta =$

	0	1
q_1	q_2	q_2
q_2	q_2	q_3
q_3	q_1	q_1

Sia D un DFA, chiamiamo **linguaggio dell'automa**, e denotiamo $L(D)$, l'insieme delle stringhe che date in input a D fanno si che D termini su uno stato di accettazione. Per definire formalmente un linguaggio

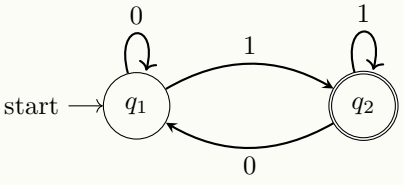


Figura 1.2: il linguaggio di tale automa risulta essere composto dalle stringhe che terminano con 1

di un automa, è necessario introdurre la **funzione di transizione estesa**:

$$\delta^*(q, \epsilon) = \delta(q, \epsilon)$$

$$\delta^*(q, ax) = \delta^*(\delta(q, a), x)$$

dove

$$a \in \Sigma, \quad x \in \Sigma^*, \quad \epsilon = \text{ stringa vuota}$$

Σ^* è l'insieme di tutte le stringhe formate dall'alfabeto Σ. Passiamo ora alla definizione di **configurazione**, essa rappresenta lo stato dell'automa ad un certo punto della computazione, essa è formata da una coppia

$$Q \times \Sigma^*$$

Rappresentante uno stato, ed una stringa di input rimanente da computare.

Un **passo della computazione** in un automa rappresenta una transizione da una configurazione ad un altra, è una relazione binaria $\vdash_D$: $Q \times \Sigma^*$ tale che

$$(p, ax) \vdash_D (q, x) \iff \delta(p, a) = q \quad \text{dove} \quad p, q \in Q, \;\; a \in \Sigma, \;\; x \in \Sigma^*$$

Si può estendere la definizione di passo di computazione, considerando la sua *chiusura transitiva* $\vdash_D^*$. Essa si ottiene aggiungendo a $\vdash_D$ tutte le coppie in $Q \times \Sigma^*$ che rendono la relazione chiusa rispetto la riflessività e rispetto la transitività.

$$(q, aby) \vdash_D (p, by) \;\wedge\; (p, by) \vdash_D (ry) \implies (q, aby) \vdash_D^* (r, y)$$

Ad esempio, nell'automa in figura 1.2, risulta chiaro che

$$\begin{cases} (q_1, 011) \vdash_D (q_1, 11) \\ (q_1, 11) \vdash_D (q_2, 1) \\ (q_2, 1) \vdash_D (q_2, \epsilon) \end{cases} \implies (q_1, 011) \vdash_D^* (q_2, \epsilon)$$

Inoltre

$$\begin{gathered} \delta^*(q_1, 011) = \\ \delta^*(q_1, 11) = \\ \delta^*(q_2, 1) = \\ \delta^*(q_2, \epsilon) = q_2 \end{gathered}$$

Se non specificato diversamente, con ϵ verrà indicata la stringa vuota. Utilizzando le precedenti definizioni, è possibile definire formalmente quali sono gli input accettati da un DFA.

Definizione : : Sia $D = (Q, \Sigma, \delta, q_0, F)$ un DFA, e sia $x \in \Sigma^*$ una stringa, essa è **accettata** da D se

$$\delta^*(q_0, x) \in F$$

Il **linguaggio riconosciuto** da D è

$$L(D) = \{x \in \Sigma^* | \delta^*(q_0, x) \in F\}$$

Definizione (Linguaggi Regolari) : : L'insieme dei linguaggi regolari, denotato REG, contiene tutti i linguaggi, tali che esiste un DFA che li ha come linguaggi riconosciuti.

$$REG = \{L \mid \exists D = (Q, \Sigma, \delta, q_0, F) \texttt{ t.c. } L \in \Sigma^* \wedge L(D) = L\}$$

Uno fra gli scopi di questo corso riguarda il capire come progettare automi, e capire se, ogni linguaggio è regolare, o ce ne sono alcuni che non possono essere riconosciuti da alcun possibile DFA.

1.1.1 Esempi di DFA

Vediamo in questa sezione alcuni semplici esempi di DFA.

Esempio 1) Si vuole progettare un DFA che accetti il seguente linguaggio

$$\{x \in \{0, 1\}^* \mid w_h(x) \geq 3\}$$

Si ricordi come

$$w_h(x) = \text{ occorrenze di 1 in } x$$

Una volta progettato il DFA, è anche importante dimostrarne la correttezza, ossia dare una prova matematico che l'automa in questione accetti il linguaggio.

- Se $x \in L(D)$ allora D accetta x
- Se D accetta x allora $w_h(x) \geq 3$

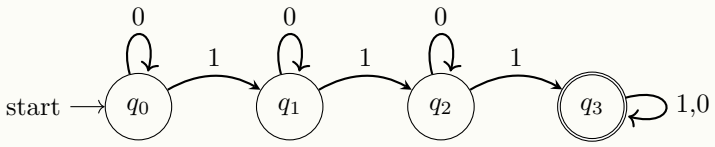


Figura 1.3: Esempio (1) di DFA

In questo, e nei seguenti casi, essendo i DFA estremamente semplici, risulta ovvio che accettino il dato linguaggio, in casi più avanzati, sarà necessario fornire una dimostrazione rigorosa.

Esempio 2) Si vuole progettare un DFA che accetti il seguente linguaggio

$$\{x \in \{0,1\}^* \mid x = 1y \wedge y \in \{0,1\}^*\}$$

Appunto sulla notazione : Se $a \in \Sigma^*$ e $b \in \Sigma^*$, allora con ab si denota la concatenazione di stringhe.

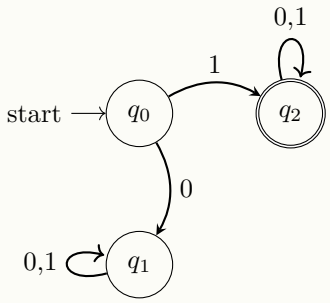


Figura 1.4: Esempio (2) di DFA

Nell'esempio (2), quando dallo stato q_0 il DFA riceve in input 0, la computazione cade su uno stato "buco nero", dalla quale non si può uscire a prescindere dall'input, l'operazione che fa cadere in questo stato è da considerarsi "non definita" in quanto non porterà mai la computazione a terminare su uno stato accettabile, è quindi comodo rimuovere tale stato dal diagramma.

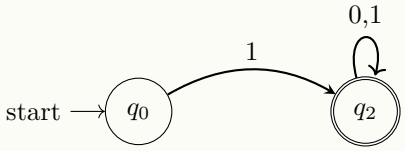


Figura 1.5: Esempio (2.1) di DFA

Anche in questo caso la dimostrazione della correttezza risulta banale.

Esempio 3) Si vuole progettare un DFA che accetti il seguente linguaggio

$$\{x \in \{0,1\}^* \mid x = 0^n 1, \;\; n \in \mathbb{N}\}$$

Con $0^n 1$ si intende una stringa che sia composta esclusivamente da 0, ma con un 1 come ultimo termine, ad esempio :

0000000000000001

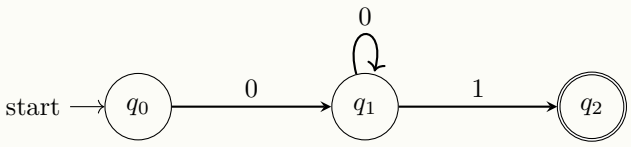


Figura 1.6: Esempio (3) di DFA

1.2 Operazioni sui Linguaggi

Lo studio delle proprietà dei linguaggi regolari può fornire opportune accortezze utili nella progettazione di automi, siccome i linguaggi sono insiemi di stringhe costruiti su un alfabeto Σ, essi godono delle operazioni insiemistiche.

Risulta utile definire formalmente la concatenazione fra stringhe, siano

$$x = a_1, a_2 \ldots, a_n \qquad y = b_1, b_2 \ldots, b_n$$

due stringhe, esse possono essere concatenate

$$xy = a_1, a_2 \ldots, a_n, b_1, b_2 \ldots, b_n$$

L'operazione di concatenazione non è commutativa, può essere definita ricorsivamente in tal modo :

$$x(ya) = (xy)a$$

dove

$$x, y \in \Sigma^* \qquad a \in \Sigma$$

Siano L_1, L_2 due linguaggi regolari in REG (per semplicità, definiti su uno stesso alfabeto Σ), e sia n un numero naturale, sono definite su di essi le seguenti operazioni :

- **unione** : $L_1 \cup L_2 = \{x \in \Sigma^* \mid x \in L_1 \vee x \in L_2\}$
- **intersezione** : $L_1 \cap L_2 = \{x \in \Sigma^* \mid x \in L_1 \wedge x \in L_2\}$
- **complemento** : $\neg L_1 = \{x \in \Sigma^* \mid x \notin L_1\}$
- **concatenazione** : $L_1 \circ L_2 = \{xy \mid x \in L_1 \wedge y \in L_2\}$
- **potenza** : $L_1^n = \underbrace{L_1 \circ L_1 \circ L_1, \cdots \circ L_1}_{n \text{ volte}}$
- **star** : $L_1^* = \{x_1, x_2 \ldots, x_k \mid k \in \mathbb{Z}^+ \wedge x_i \in L_1\}$
  Si può definire anche diversamente
  $$L_1^* = \bigcup_{k=0}^{\infty} L_1^k$$

Esempio di concatenazione e potenza :

$$\Sigma = \{a, b\} \quad L_1 = \{a, ab, ba\} \quad L_2 = \{ab, b\} \quad L = \{a, ab, ba\}$$

$$L_1 \circ L_2 = \{aab, ab, abab, abb, baab, bab\}$$

$$L^2 = \{aa, aab, aba, abab, abba, baa, baba\}$$

Teorema (Chiusura di REG) : La classe dei linguaggi regolari REG, è chiusa rispetto a tutte le operazioni appena elencate, siano L_1 ed L_2 due linguaggi regolari, allora :

$$L_1 \cup L_2 \in REG \qquad L_1 \circ L_2 \in REG \qquad L_1 \cap L_2 \in REG$$

$$L_1^n \in REG \qquad \neg L_1 \in REG \qquad L_1^* \in REG$$

Dimostrazione (unione ed intersezione) : Siano L_1 ed L_2 due linguaggi regolari, considero due DFA, per semplicità, con lo stesso alfabeto

$$D_1 = (Q_1, \Sigma, \delta, q_1, F_1)$$

$$D_2 = (Q_2, \Sigma, \delta, q_2, F_2)$$

tali che

$$L(D_1) = L_1 \ \wedge \ L(D_2) = L_2$$

Si costruisce un DFA che simula contemporaneamente l'esecuzione di D_1 e D_2, in cui gli stati possibili saranno le possibili combinazioni di coppie di stati. Si definisce $D = (Q, \Sigma, \delta, q_0, F)$ tale che

- $Q = Q_1 \times Q_2 = \{(r_1, r_2) \| \ r_1 \in Q_1 \wedge r_2 \in Q_2\}$
- $\delta((r_1, r_2), a) = (\delta(r_1, a), \delta(r_2, a))$ dove $a \in \Sigma$ e $(r_1, r_2) \in Q$
- $q_0 = (q_1, q_2)$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(r_1, r_2) \| \ r_1 \in F_1 \vee r_2 \in F_2\}$

Nel caso si dovesse dimostrare la proprietà dell'intersezione, si avrebbe che

- $F = F_1 \times F_2$

A questo punto risulta chiaro che

(i) $x \in L_1 \cup L_2 \implies x \in L(D)$

(ii) $x \in L(D) \implies x \in L_1 \cup L_2$ ■

Dimostrazione (complemento) : : Sia L un linguaggio regolare, e D un automa che lo accetta $L(D) = L$. Si vuole dimostrare che esiste un automa che accetti $L^C = \{w \in \Sigma^* \mid w \notin L\}$. Essendo

$$D = (Q, \Sigma, \delta, q_0, F)$$

considero

$$D' = (Q, \Sigma, \delta, q_0, F^C)$$

dove $F^C = Q \backslash F$. Supponiamo che $w \in L^C$, allora sicuramente, se data come input a D', la computazione terminerà in uno stato che non è in F, dato che, per definizione, se terminasse in F, sarebbe accettato da D, ma L^C contiene tutte le stringhe che non sono accettate da D, quindi

- $w \in L^C$
- la computazione termina in uno stato $q \in F^C$
- D' accetta w
- L^C è un linguaggio regolare. ■

Per dimostrare la proprietà di concatenazione, è necessario introdurre un nuovo concetto.

1.3 Non Determinismo

Si può generalizzare il modello di DFA, in modo tale che la lettura di un input non scaturisca il passaggio da uno stato ad un'altro, ma da uno stato ad un insieme di stati, tale generalizzazione è detta *Non-Deterministic Finite Automa.*

Definizione (NFA) : : Un NFA è una tupla $N = (Q, \Sigma, \delta, q_0, F)$ tale che

- Q, Σ, q_0, F condividono la definizione con i loro corrispettivi nel DFA
- $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$

- $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$
- $\mathcal{P}(Q)$ è l'insieme delle parti di Q

Una computazione in un NFA è paragonabile ad una computazione parallela, in cui un input può risultare in diversi *rami* di computazione. Una funzione di transizione di un NFA inoltre accetta la stringa vuota ϵ, se la computazione finisce in uno stato in cui è presente un arco di questo tipo, verrà considerata anche una diramazione verso quell'arco a prescindere dall'input.

Una computazione si può rappresentare graficamente con un albero, se una delle diramazioni possibili termina in uno stato accettabile, allora la stringa in input è accettata.

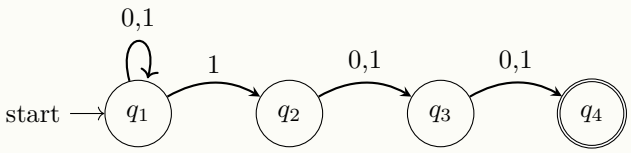


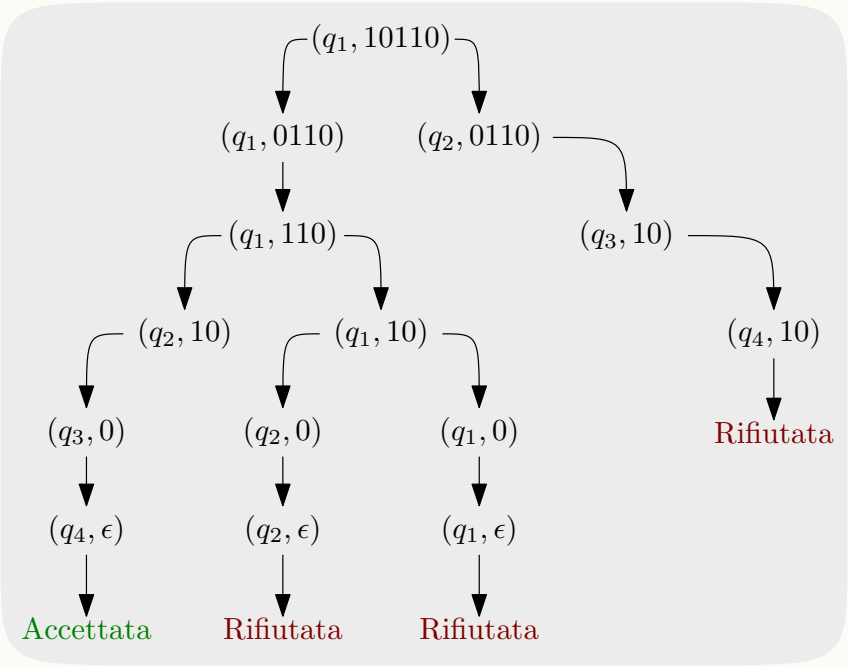
Figura 1.7: Esempio di NFA

Sia N l'NFA mostrato in figura 1.7, si ha

$$L(N) = \{x \in \{0,1\}^* \| \ x \text{ ha 1 come terzultimo valore } \}$$

Si può visualizzare il seguente albero di computazione data come input una stringa w :

$$w = 10110$$



Essendo che la traccia di sinistra accetta w, allora N accetta w.

È necessario estendere il concetto di *configurazione* per gli NFA, essa, rappresentante uno stato della computazione, sarà una coppia

$$(q, x) \in Q \times \Sigma_\epsilon$$

E diremo che

$$(p, ax) \vdash_N (q, x) \iff q \in \delta(p, a)$$

dove

$$p, q \in Q \quad a \in \Sigma_\epsilon \quad x \in \Sigma_\epsilon^*$$

Si considera ora la chiusura transitiva di $\vdash_N$, denotata $\vdash_N^*$, se w è una stringa ed N un NFA, si ha che

$$w \in L(N) \iff \exists q \in F \| \ (q_0, w) \vdash_N^* (q, \epsilon)$$

Consideriamo adesso l'unione di due NFA, risulta particolarmente semplice da definire, siano N_1 e N_2 due NFA, che per semplicità, condividono l'alfabeto

$$N_1 = \{Q_1, \Sigma_\epsilon, \delta_1, q_1, F_1\}$$

$$N_2 = \{Q_2, \Sigma_\epsilon, \delta_2, q_2, F_2\}$$

Definisco un nuovo NFA $N = (Q, \Sigma_\epsilon, \delta, q_0, F)$ tale che

- $Q = Q_1 \cup Q_2$
- $F = F_1 \cup F_2$
- Siano $q \in Q$ e $a \in \Sigma_\epsilon$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a) \text{ se } q \in Q_1 \\ \delta_2(q,a) \text{ se } q \in Q_2 \\ \{q_1, q_2\} \text{ se } q = q_0 \ \wedge \ a = \epsilon \\ \emptyset \text{ se } q = q_0 \ \wedge \ a \neq \epsilon \end{cases}$$

Si avrà che $L(N) = L(N_1) \cup L(N_2)$

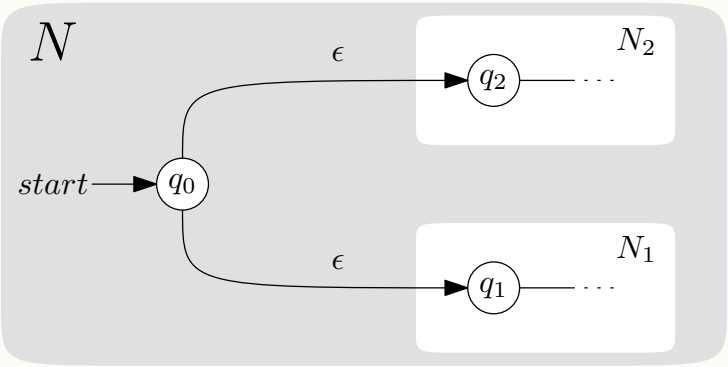


Figura 1.8: Unione di due NFA

Denotiamo $\mathcal{L}(DFA)$ l'insieme dei linguaggi accettati da un qualsiasi DFA, che per definizione è REG, e denotiamo, in maniera analoga $\mathcal{L}(NFA)$.

Teorema : : L'insieme dei linguaggi accettati da un qualsiasi DFA, e quello dei linguaggi accettati da un qualsiasi NFA coincidono

$$\mathcal{L}(DFA) = \mathcal{L}(NFA) = REG$$

Dimostrazione : Il caso $\mathcal{L}(DFA) \subseteq \mathcal{L}(NFA)$ è banale e non verrà dimostrato. Si vuole dimostrare che se L è accettato da un generico NFA, allora esiste un DFA che accetta L, l'idea è quella di "simulare" un NFA tramite un DFA che rappresenti ogni possibile stato di computazione.

Sia $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ un NFA, e sia $L = L(N)$. Si considera un DFA $D = (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$ tale che

- $Q_D = \mathcal{P}(Q_N)$
- $q_{0_D} = \{q_{0_N}\}$
- $F_D = \{R \in Q_D | R \cap F_N \neq \emptyset\}$, ovvero, D accetta tutti gli insiemi in cui compare almeno un elemento accettato da N.

- Sia $R \in Q_D$ e $a \in \Sigma$, si definisce $\delta_D(R, a) = \bigcup_{r \in R} \delta_N(r, a)$

Questo caso non tiene conto di un NFA in cui sono presenti degli ϵ-archi. Supponiamo che vi siano, sia $R \in Q_D$, si definisce la funzione estesa E definita come segue

$$E(R) = \{q \in Q_N \| \ q \text{ può essere raggiunto da un qualsiasi stato } r \in R \text{ attraverso zero o più } \epsilon\text{-archi } \}$$

Cambia la definizione del DFA utilizzato per la dimostrazione

- $q_{0_D} = E(\{q_{0_N})\}$
- $\delta_D(R, a) = \bigcup_{r \in R} E(\delta_N(r, a))$

È chiaro che D tiene traccia di tutte le possibili computazioni di N, ed accetta L, ossia $L(D) = L(N)$. ■

Esempio : Si consideri l'NFA N definito come segue

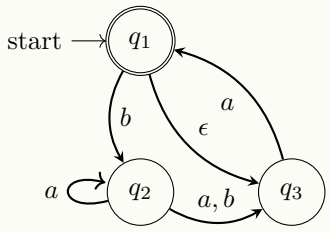


Figura 1.9: $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$

Si costruisce un DFA $D = (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$ con le seguenti specifiche (per comodità, l'elemento $\{q_i, q_j \ldots, q_k\}$ verrà denotato $p_{ij\ldots k}$), mostrato in figura 1.10

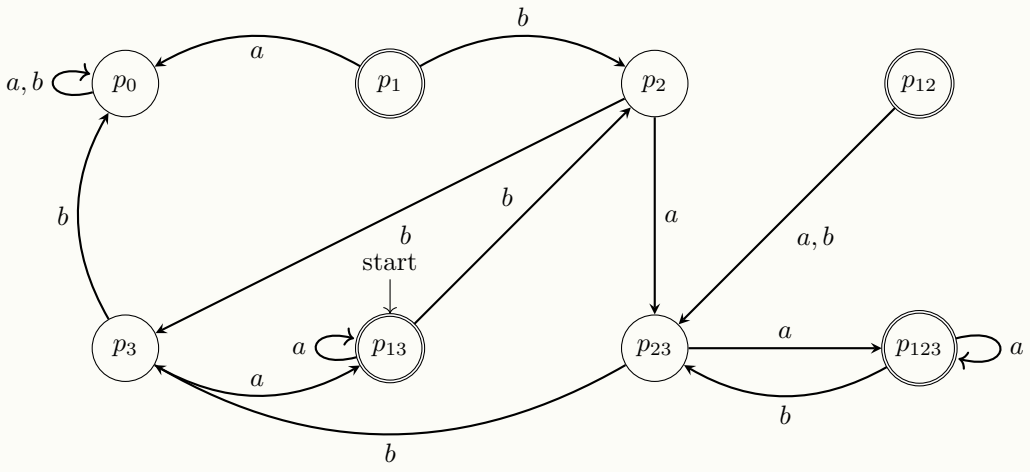


Figura 1.10: grafico di D

- $Q_D = \{p_0, p_1, p_2, p_3, p_{12}, p_{13}, p_{23}, p_{123}\}$
- $E(q_{0_N}) = \{q_1, q_3\} \implies q_{0_D} = p_{13}$
- $F_D = \{p_1, p_{12}, p_{13}, p_{123}\}$
- la funzione δ_D si definisce osservando il grafico di N
  - $\delta_N(q_2, a) = \{q_2, q_3\} \implies \delta_D(p_2, a) = p_{23}$
  - $\delta_N(q_2, b) = \{q_3\} \implies \delta_D(p_2, a) = p_3$

- $\delta_N(q_1, a) = \emptyset \implies \delta_D(p_1, a) = p_0$
- $\delta_N(q_1, b) = \{q_2\} \implies \delta_D(p_1, b) = p_2$
- etc...

L'introduzione degli automi non deterministici è stata necessaria in principio per la dimostrazione della chiusura di *REG* rispetto le operazioni di concatenazione e star.

Teorema : *REG* è chiusa per concatenazione.

Dimostrazione : Siano L_1 ed L_2 due linguaggi regolari, esistono quindi due NFA

$$N_1 = (Q_1, \Sigma_\epsilon, \delta_1, q_0^1, F_1) \qquad N_2 = (Q_2, \Sigma_\epsilon, \delta_2, q_0^2, F_2)$$

tali che $L(N_1) = L_1 \ \wedge \ L(N_2) = L_2$. Si costruisce un NFA $N = (Q, \Sigma_\epsilon, \delta, q_0, F)$, l'idea è quella di concatenare le ramificazioni di N_1 ad N_2.

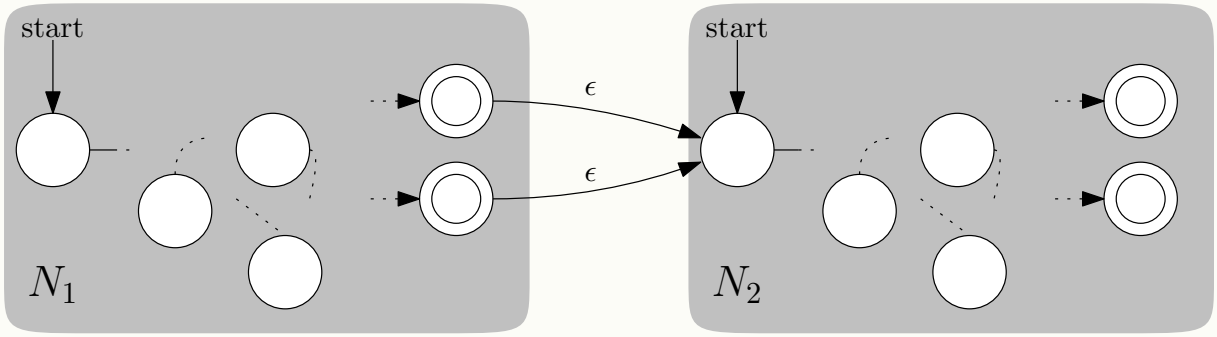


Figura 1.11: schema di N

Un NFA di questo tipo computerà una stringa in L_1, se finirà in uno stato di F_1, andrà nello stato iniziale di N_2, è chiaro che l'automa accetta solamente una concatenazione di stringhe fra L_1 ed L_2.

- $Q = Q_1 \cup Q_2$
- $q_0 = q_0^1$
- $F = F_2$
- per $a \in \Sigma_\epsilon$ e $q \in Q$ si ha $\delta(q, a) = \begin{cases} \delta_1(q, a) \text{ se } q \in Q_1 \wedge q \notin F_1 \\ \delta_1(q, a) \text{ se } q \in F_1 \wedge a \neq \epsilon \\ \delta_1(q, a) \cup \{q_0^2\} \text{ se } q \in F_1 \wedge a = \epsilon \\ \delta_2(q, a) \text{ se } q \in Q_2 \end{cases}$

Si ha quindi che $L(N) = L_1 \circ L_2$. ■

Teorema : *REG* è chiusa per star.

Dimostrazione : Sia $L \in REG$ e sia $N = (Q, \Sigma_\epsilon, \delta, q_0, F)$ un NFA tale che $L(N) = L$. Considero un NFA $N^* = (Q^*, \Sigma_\epsilon, \delta^*, q_0^*, F^*)$, identico ad N, con opportune modifiche, lo stato q_0 iniziale di N non è iniziale in N^*, ed ogni stato finale ha una ϵ-arco verso q_0.

- $Q^* = Q \cup \{q_0^*\}$
- q_0^* è un nuovo stato
- $F^* = F \cup \{q_0^*\}$ questo perché in L^* è presente la stringa vuota

- per $a \in \Sigma_\epsilon$ e $q \in Q^*$ si ha $\delta^*(q,a) = \begin{cases} \delta(q,a) \text{ se } q \in Q \wedge q \notin F \\ \delta(q,a) \text{ se } q \in F \wedge a \neq \epsilon \\ \delta(q,a) \cup \{q_0\} \text{ se } q \in F \wedge a = \epsilon \\ \{q_0\} \text{ se } q = q_0^* \wedge a = \epsilon \\ \emptyset \text{ se } q = q_0^* \wedge a \neq \epsilon \end{cases}$

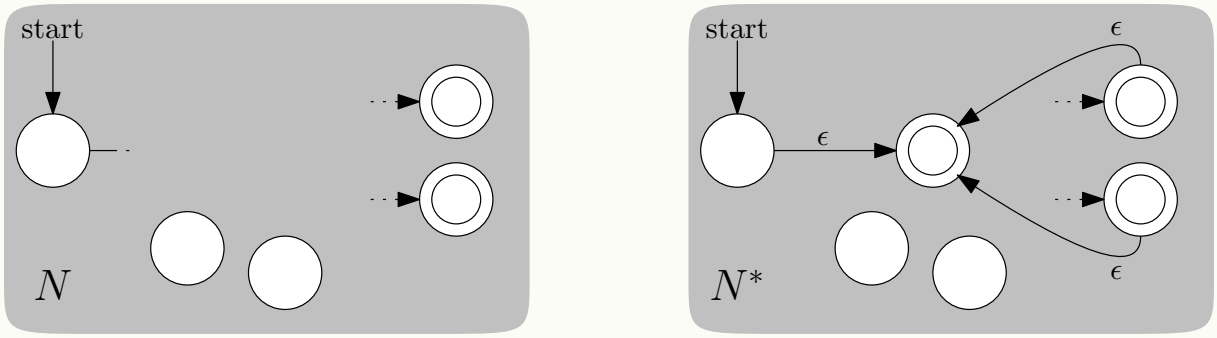


Figura 1.12: schema di N^*

1.4 Espressioni Regolari

Un espressione regolare è simile ad un espressione algebrica ma opera sulle stringhe, dato un alfabeto, un'espressione su tale alfabeto rappresenta un insieme di stringhe, un esempio è

$$(0|1)0^*$$

Dove $(0|1) \equiv \{0\} \cup \{1\} = \{0,1\}$ e $0^* \equiv \{0\}^*$ quindi $(0|1)0^* \equiv \{0,1\} \circ \{0\}^*$.

Definizione (espressione regolare) : : Sia Σ un alfabeto, un espressione regolare r su Σ, denotata $r \in re(\Sigma)$, è definita per induzione

Caso base

- $r = \emptyset \in re(\Sigma)$
- $r = \epsilon \in re(\Sigma)$
- $r = a \in re(\Sigma)$

Caso induttivo

- $r = r_1 \cup r_2$ dove $r_1, r_2 \in re(\Sigma)$
- $r = r_1 \circ r_2$ dove $r_1, r_2 \in re(\Sigma)$
- $r = r_1^*$ dove $r_1 \in re(\Sigma)$

L'insieme delle stringhe definite da $r \in re(\Sigma)$ è il *linguaggio* di r ed è denotato $L(r)$.

Esempio : Sia $\Sigma = \{0,1\}$

- $0^*10^* = \{w \mid w \text{ ha esattamente un } 1\}$
- $\Sigma^*1\Sigma^* = \{w \mid w \text{ ha almeno un } 1\}$
- $\Sigma^*001\Sigma^* = \{w \mid w \text{ ha la sottostringa } 001\}$

Per convenzione si definisce

$$1^*\emptyset = \emptyset \qquad \emptyset^* = \epsilon$$

Teorema Fondamentale : Sia $\mathcal{L}(DFA) = REG$ l'insieme dei linguaggi accettati da un qualsiasi DFA, e sia $\mathcal{L}(re)$ l'insieme dei linguaggi accettati da una qualsiasi espressione regolare, è vero che

$$\mathcal{L}(re) = \mathcal{L}(DFA) = REG$$

Dimostrazione : è necessario dimostrare due direzioni

$\boxed{\mathcal{L}(re) \subseteq \mathcal{L}(DFA)}$: Sia r un espressione regolare, si considera un DFA D_r definito come segue, a seconda dei casi

Caso base

- $r = \emptyset \implies D_r$ non accetta alcuna stringa
- $r = \epsilon \implies D_r$ accetta la stringa vuota
- $r = a \implies D_r$ accetta la $a \in \Sigma$

Caso induttivo

- $r = r_1 \cup r_2$, esistono due automi D_{r_1} e D_{r_2} che accettano rispettivamente $L(r_1)$ e $L(r_2)$, ma allora esiste necessariamente un automa D_r che accetta $L(r_1) \cup L(r_2)$.
- $r = r_1 \circ r_2$, esistono due automi D_{r_1} e D_{r_2} che accettano rispettivamente $L(r_1)$ e $L(r_2)$, ma allora esiste necessariamente un automa D_r che accetta $L(r_1) \circ L(r_2)$.
- $r = r_1^*$, esiste un automa D_{r_1} che accetta $L(r_1^*)$, ma allora esiste necessariamente un automa D_r che accetta $L(r_1)$.

Tali tesi sono vere dato che la classe dei linguaggi regolari è chiusa per le operazioni di star, concatenazione ed unione. □

$\boxed{\mathcal{L}(DFA) \subseteq \mathcal{L}(re)}$: Sia L un linguaggio regolare, e sia N l'NFA tale che $L(N) = L$. Si costruisce un nuovo tipo di NFA che sarà equivalente ad N. Tale automa è detto *GNFA*, dove la G sta per "Generalizzato", tale automa ha una *forma canonica*, ossia, rispetta le seguenti proprietà

- Lo stato iniziale, ha solo archi uscenti
- Vi è un singolo stato finale, ed ha solo archi entranti
- Per ogni coppia di stati (non necessariamente distinti), c'è esattamente un arco.
- Ogni arco è etichettato da un espressione regolare.

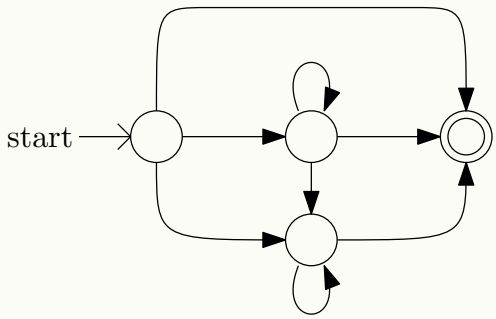


Figura 1.13: forma di un GNFA

Più precisamente, sia G un GNFA definito $G = (Q, \Sigma, \delta, q_{start}, q_{acc})$ dove

$$\delta : Q\backslash\{q_{acc}\} \times Q\backslash\{q_{start}\} \to re(\Sigma)$$

Dato un generico NFA, è possibile trasformarlo in un GNFA aggiungendo al più due stati (iniziale e finale), ed utilizzando gli ϵ-archi per riempire le coppie di stati che non sono collegate.

La funzione $Convert : GNFA \to GNFA$ modifica un GNFA restituendone uno equivalente, ma con uno stato in meno. Tale funzione è definita in tal modo, sia k il numero di archi di G, si esegue $Convert(G)$

- Se $k = 2$, allora esiste un solo arco fra questi etichettato con un espressione regolare r, la funzione restituirà r.
- Se $k > 2$, viene selezionato un qualsiasi nodo in $Q\backslash\{q_{start}, q_{acc}\}$, sia questo q_{rip}, si avrà $Convert(G) = G' = (Q\backslash\{Q_{rip}\}, \Sigma, \delta', q_{start}, q_{acc})$ dove

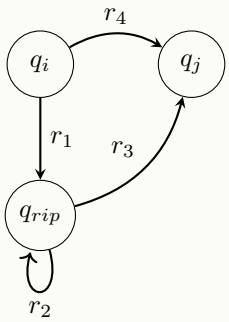
  $$\delta' : Q\backslash\{q_{acc}, q_{rip}\} \times Q\backslash\{q_{start}, q_{rip}\} \to re(\Sigma)$$

  Inoltre ogni etichetta di G' viene aggiornata secondo la seguente procedura, siano $q_i \in Q\backslash\{q_{acc}, q_{rip}\}$ e $q_j \in Q\backslash\{q_{start}, q_{rip}\}$ due stati qualsiasi

  $$\delta'(q_i, q_j) = r_1 r_2^* (r_3 | r_4)$$

  Dove

  - $r_1 = \delta(q_i, q_{rip})$
  - $r_2 = \delta(q_{rip}, q_{rip})$
  - $r_3 = \delta(q_{rip}, q_j)$
  - $r_4 = \delta(q_i, q_j)$



Bisogna ora dimostrare che un generico GNFA G è equivalente a $Convert(G)$. Si dimostra per induzione su k numero di stati.

caso base $k = 2$: In tal caso la procedura $Convert$ restituisce l'espressione regolare r sull'unico arco che descrive ogni stringa accettata da G. $L(r) \equiv L(G)$

passo induttivo : si assume che G è equivalente a $Convert(G)$ per $k - 1$ stati.

- Se G accetta w, allora esiste un ramo di computazione $C = \{q_{start}, q_1 \ldots, q_{accept}\}$, se q_{rip} che è stato rimosso in $G' = Convert(G)$ non appartiene a C, allora la computazione non è alterata e G' accetta w, altrimenti ci sarà una differente sequenza di stati, ma gli stati q_i, q_j adiacenti a q_{rip} sono ora uniti da un arco etichettato da un espressione regolare che comprende le stringhe per andare da q_i a q_j passando per q_{rip}.
- Se G' accetta w, anche G lo accetta dato che per ogni coppia di stati in C si è aggiornata l'etichetta tenendo conto della transazione che porta da uno stato all'altro passando per q_{rip}.

Quindi $Convert$ restituisce un automa equivalente con $k - 1$ stati, quindi l'asserto è vero. ■

1.4.1 Esempi

Esempio 1) Si trasformi $r = (ab|a)^*$ in un NFA.

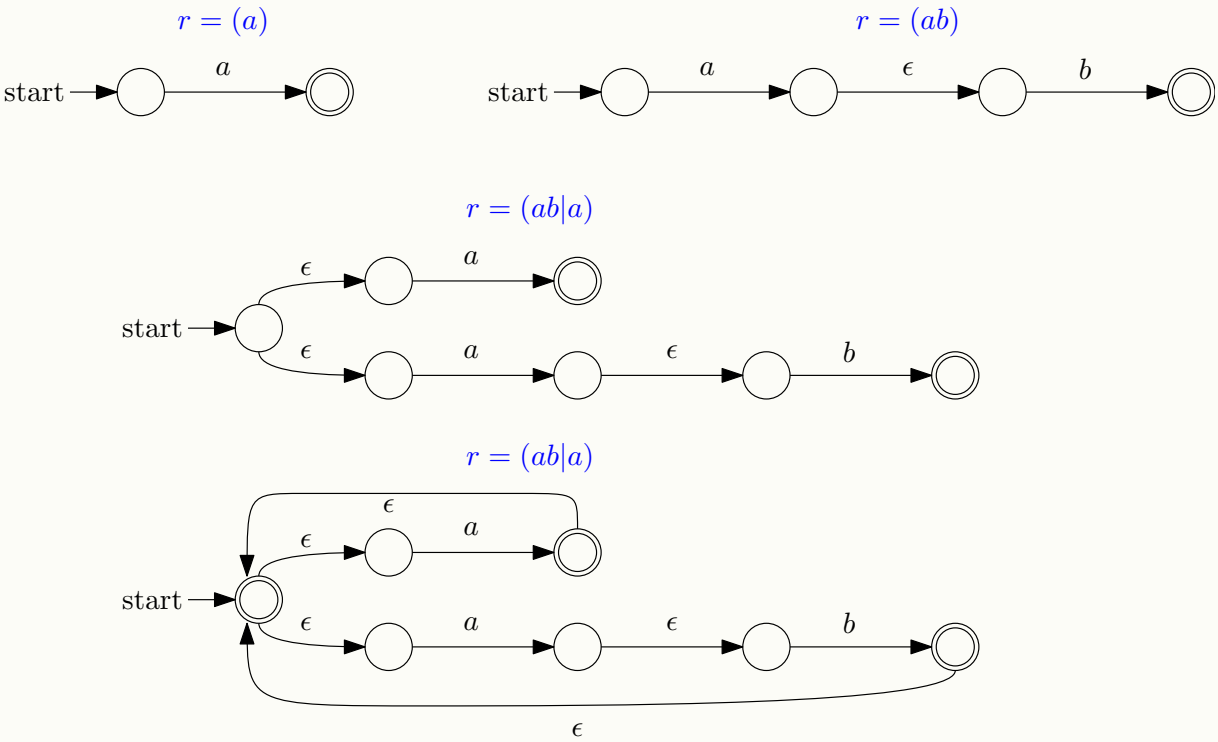
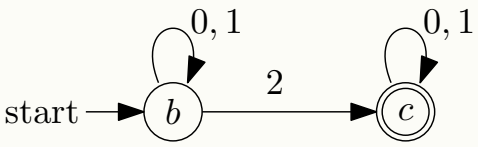
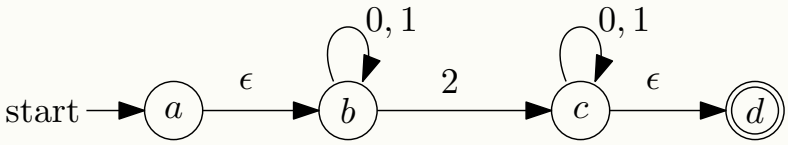


Figura 1.14: Esempio 1

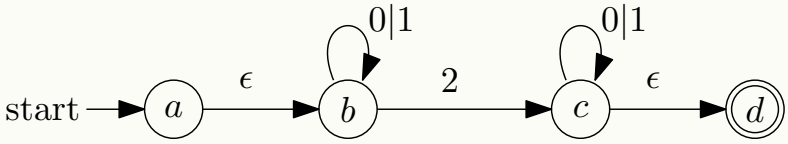
Esempio 2) Dato il seguente automa, si trovi l'espressione regolare associata



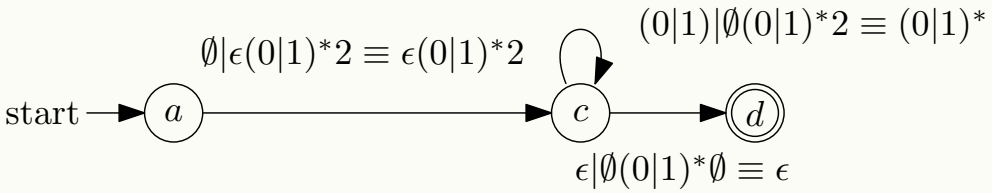
passo 1 : si trasforma in forma canonica (ignorati gli archi etichettati con $\emptyset$)



passo 2 : si trasformano le etichette in espressioni regolari



passo 3 : si rimuove b



passo 4 : si rimuove b

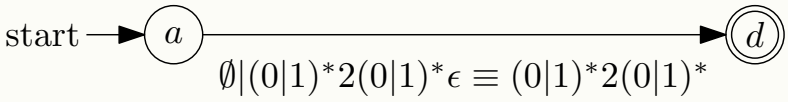


Figura 1.15: Esempio 2

1.5 Linguaggi non regolari

1.5.1 Il Pumping Lemma

A questo punto della lettura, è naturale porsi una domanda : Tutti i linguaggi sono regolari? Esistono linguaggi che non possono essere accettati da alcun DFA? Nel caso solamente un sottoinsieme dei linguaggi fosse regolare, quali proprietà soddisfa? Si consideri il seguente linguaggio

$$L = \{0^n 1^n \mid n \geq 0\}$$

Si può provare a disegnare un automa che accetti L, rendendosi ben presto conto che è *impossibile*, L non è regolare, esistono quindi dei linguaggi che non sono regolari. L'automa a stati finiti è un modello semplice, non può "ricordare" quanti caratteri di un certo tipo sono stati letti.

Essendo che solamente alcune stringhe possono essere accettate da un qualsiasi automa, è importante caratterizzare tali stringhe e definirne le proprietà in tal merito.

Appunto sulla notazione : Se w è una stringa, allora $|w|$ è il numero dei suoi caratteri.

Osservzione : Se un DFA con n stati legge una stringa di $k > n$ caratteri, allora ci sarà almeno uno stato che verrà considerato due volte durante la computazione.

Teorema (Pumping Lemma) : Sia L un linguaggio regolare, sia D l'automa tale che $L(D) = L$, si considera una stringa $w \in L(D)$, ed una sua decomposizione in 3 stringhe concatenate $w = xyz$. Esiste un intero $p \leq |w|$, denotato *pumping* tale che

1. $\forall i \geq 0$, $xy^i z \in L(D)$
2. $|y| > 0$
3. $|xy| \leq p$

Dimostrazione : Sia $D = (Q, \Sigma, \delta, q_{start}, F)$ un automa, e sia $p = |Q|$. Sia w una stringa su Σ di $n \geq p$ caratteri definita $w = w_1 w_2 \dots w_n$. Sia $\{r_1, r_2 \dots, r_{n+1}\}$ la sequenza di stati che D computa su input w, ossia

$$\delta(r_i, w_i) = r_{i+1}$$

Tale sequenza è lunga $n+1 \geq p+1$ stati, fra i primi $p+1$ elementi c'è necessariamente uno stato ripetuto, sia r_j la prima occorrenza di tale stato, e sia r_l la seconda.

Siccome la ripetizione avviene fra le prime $p+1$ computazioni, si ha che $l \leq p+1$. Si consideri la seguente scomposizione di w

- $x = w_1, w_2 \dots, w_{j-1}$
- $y = w_j, w_{j+1} \dots, w_{l-1}$
- $z = w_l, w_{l+1} \dots, w_n$

1. $xy^i z \in L(D)$ perché x parte da $r_1 = q_{start}$ e arriva a r_j, y^i parte da r_j e ritorna su r_l, che è lo stesso stato, e z porta da r_l allo stato finale di accettazione.
2. Essendo che $i \neq l$, $|y| > 0$.
3. $l \geq p + 1$ ovvero $l - 1 = |xy| \geq p$.

I tre punti sono dimostrati. ■