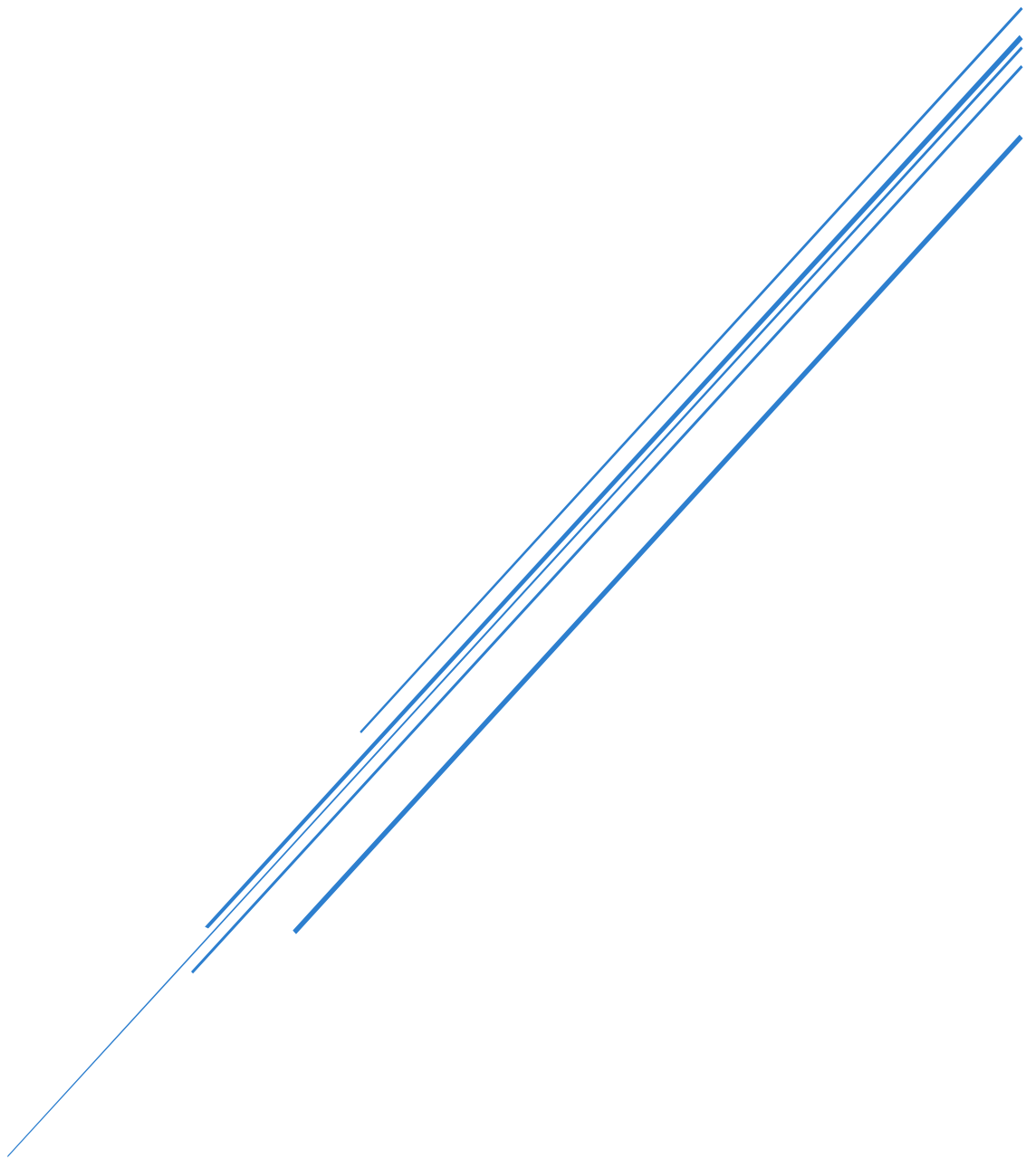


CORSO DI INFORMATICA TEORICA

Luigi Palopoli



A cura di Giovanni Murfuni
Ingegneria informatica laurea magistrale A.A. 2024-2025

Sommario

Programma.....	5
Parte prima.....	6
La gerarchia di Chomsky.....	6
Cos'è un linguaggio naturale?.....	6
Cos'è un linguaggio artificiale?.....	6
Definizione di strutture fondamentali del linguaggio.....	6
Gerarchia di Chomsky.....	6
Cos'è la grammatica?.....	7
Dalla struttura della grammatica alla classe del linguaggio.....	8
Automi.....	8
Definizione di Alfabeto.....	9
Sottostringa e casi particolari (Prefisso e Suffisso).....	10
Definizione di Linguaggio.....	10
Automi riconoscitori.....	11
Deterministic Finite Automation (DFA).....	11
Funzione di transizione estesa alle stringhe per DFA (generalizzazione di δ alle stringhe).....	13
Linguaggio riconosciuto da un automa A.....	13
Equivalenza tra due automi.....	13
Non Deterministic Finite Automation (NFA).....	13
Funzione di transizione estesa alle stringhe per NFA (generalizzazione di δ alle stringhe).....	14
L'automata (NFA) A che linguaggio riconosce.....	15
Problema dell'equivalenza tra DFA e NFA.....	15
Teorema 1.1: Esistenza della controparte deterministica/non deterministica.....	15
Trasformazione.....	15
Teorema 1.2: Appartenenza di una stringa.....	17
Teorema 1.3: DFA e NFA equivalenti riconoscono lo stesso linguaggio.....	17
Teorema 1.4: Condizione necessaria e sufficiente per accettare un linguaggio.....	17
Caso sfavorevole di costruzione per sottoinsiemi e principio della colombaria.....	18
Le macchine con ε -transizioni.....	18
Automi con ε -transizioni.....	19
Espressioni regolari.....	19
Operazioni con i linguaggi.....	20
Casi di un linguaggio regolare.....	20
Teorema 1.5: Esistenza ER per $L(A)$	21

Teorema 1.6: Simmetrico all'1.5	21
Proprietà delle espressioni regolari	22
Pumping Lemma (per i linguaggi regolari o “gioco a due”)	22
Dimostrazione del Pumping Lemma per linguaggi regolari.....	23
Prototipi di linguaggi regolari, liberi o non liberi (da contesto).....	24
Altre proprietà dei linguaggi regolari	25
Quanto costa produrre da una rappresentazione a un'altra dello stesso linguaggio?	26

Programma

Parte prima: Le gerarchie di Chomsky.

Parte seconda: Compatibilità (cosa si può e cosa non si può calcolare): Turing.

Parte terza: Complessità.

Libri:

1. Introduzione alla teoria della computazione - Michael Sipster
2. Introduzione alla teoria degli automi, linguaggi e calcolo - Hopcroft, Motwani, Ullman

Parte prima

La gerarchia di Chomsky

Iniziamo definendo le differenze tra **linguaggio naturale** e **linguaggio artificiale**.

Cos'è un linguaggio naturale?

Un **linguaggio naturale** è un sistema di comunicazione sviluppato spontaneamente da una comunità umana per l'interazione sociale, caratterizzato da una struttura complessa e in continua evoluzione.

A differenza dei linguaggi formali, come quelli utilizzati nei computer, i linguaggi naturali presentano:

- **Ambiguità:** Una stessa frase può avere molteplici interpretazioni a seconda del contesto, delle sfumature e delle intenzioni comunicative.
- **Contesto-dipendenza:** Il significato di una parola o di un'espressione può variare in base alla situazione in cui viene utilizzata.
- **Ricchezza espressiva:** I linguaggi naturali consentono di esprimere una vasta gamma di concetti, emozioni e sfumature, grazie a una varietà di strutture grammaticali e lessicali.
- **Evoluzione continua:** I linguaggi naturali sono soggetti a cambiamenti nel tempo, influenzati da fattori sociali, culturali e storici.

Cos'è un linguaggio artificiale?

Un **linguaggio artificiale** è un qualsiasi linguaggio non naturale che sia descritto in modo formale e rigoroso, quindi in modo matematico in modo da non lasciare spazio ad ambiguità e alle inconsistenze del linguaggio naturale.

Definizione di strutture fondamentali del linguaggio

1. **Lessico:** *L'insieme delle parole di una lingua, comprese le loro forme flesse e i loro significati. È il vocabolario di base di una lingua.*
2. **Sintassi:** *L'insieme delle regole che governano la combinazione delle parole in frasi o nel nostro caso dei comandi. Si occupa dell'ordine delle parole, della loro funzione all'interno della frase e della costruzione delle frasi complesse.*
3. **Semantica:** *Lo studio del significato delle parole e delle frasi. Si occupa di come le parole si riferiscono al mondo reale e di come si combinano per creare significati complessi*
4. **Pragmatica:** *Lo studio dell'uso del linguaggio in contesti specifici. Si occupa di come il significato di un enunciato varia a seconda della situazione comunicativa, delle intenzioni del parlante e delle conoscenze condivise tra i partecipanti alla comunicazione. Rappresenta il fine ultimo.*

Gerarchia di Chomsky

Per Chomsky i linguaggi formali descrivono una gerarchia, in ordine di potenza, tali classi di linguaggio sono:

1. L3: Linguaggi di tipo 3 o "Regolari"
2. L2: Linguaggi di tipo 2 o "Liberi di contesto" ($L3 \subset L2$)
3. L1: Linguaggi di tipo 1 o "Contestuali" ($L2 \subset L1$)
4. L0: Linguaggi di tipo 0 o "Ricorsivi/Generali" ($L0 \subset L1$)

I linguaggi elencati sono tutti astratti e cadono **sempre in L0 sennò sarebbero incompilabili.*

$$L3 \subset L2 \subset L1 \subset L0$$

Descrivere la semantica di ogni linguaggio è difficile, quindi ci basiamo sui tipi di grammatiche che generano tale linguaggio e sugli automi che lo riconoscono, per tradizione si parte dalle grammatiche.

Cos'è la grammatica?

La grammatica è una quadrupla di tipo $G(V, T, P, S)$ dove:

- V è l'alfabeto dei simboli non terminali o attributi sintattici;
- T è l'alfabeto dei simboli terminali;
- $S \in V$ è il simbolo iniziale (o radice), ovvero, un simbolo da cui deriva tutto il linguaggio;
- P è un insieme di produzioni, ovvero: insieme delle regole di formazione delle frasi.

Esempio di grammatica:

- V : aggettivi o verbi (non terminali);
- T : parole che formano l'italiano o punteggiatura (terminali);
- P : regola di produzione P che dice che è possibile formare una frase di una certa tipologia in uno specifico modo;
- S : simbolo iniziale "frase" (ad esempio).

Composizione di P :

$$r \in P$$

$$r \equiv < testa > \rightarrow < corpo >$$

$$< testa >, < corpo > \in (V \cup T)^*$$

**L'asterisco indica una sequenza di simboli*

Esempio di grammatica:

$$G = (V, T, P, S)$$

$$V = \{S, A, B, C\}$$

$$T = \{0, 1, 2\}$$

S simbolo iniziale

P regole di produzione:

1. $S \rightarrow 0SBC$
2. $S \rightarrow 0BC$
3. $CB \rightarrow BC$
4. $0B \rightarrow 01$
5. $1B \rightarrow 11$
6. $1C \rightarrow 12$
7. $2C \rightarrow 22$

Esempio d'uso:

001122 G grammatica $\rightarrow L(G)$ è il linguaggio generato G.

Vediamo se posso ottenere 001122 con la mia grammatica:

$S \rightarrow^1 0SBC \rightarrow^2 0(0BC)BC \rightarrow^3 00BBCC \rightarrow^4 001BCC \rightarrow^5 0011CC \rightarrow^6 00112C \rightarrow^7 001122$

Ho dimostrato che **001122** $\in L(G)$: $L(G) = \{0^n 1^n 2^n \mid n \geq 1\}$ *n non è una potenza ma la ripetizione

$$L(G) \in L1, \quad L(G) \notin L2$$

Notazione corretta:

$G(V, T, P, S)$

$(\alpha \rightarrow \beta) \in P$

$\gamma\alpha\delta \rightarrow_G \gamma\beta\delta$

FOCUS: \rightarrow_G è la derivazione diretta: $\gamma\alpha\delta$ è trasformato direttamente in $\gamma\beta\delta$ oppure $\gamma\beta\delta$ deriva direttamente da $\gamma\alpha\delta$.

$$\gamma\alpha\delta \rightarrow_G^* \gamma\beta\delta \text{ se } \begin{cases} (\alpha \rightarrow \beta) \in P \\ \text{oppure } \exists \mu \text{ t. c.} \\ (\gamma\alpha\delta \rightarrow_G^* \gamma\mu\delta) \wedge (\gamma\mu\delta \rightarrow_G \gamma\beta\delta) \end{cases}$$

$$L(G) = \{\omega \in T^* \mid S \rightarrow_G^* \omega\}$$

FOCUS: \rightarrow_G^* è la derivazione classica: $\gamma\alpha\delta$ è trasformato dalle regole della grammatica in $\gamma\beta\delta$ oppure $\gamma\beta\delta$ deriva per le regole della grammatica da $\gamma\alpha\delta$.

Dalla struttura della grammatica alla classe del linguaggio

In generale: una grammatica di tipo n genera un linguaggio di tipo n .

- Tipo 0: nessuna restrizione su G: Grammatica di tipo 0
- Tipo 1: $(\forall (\alpha \rightarrow \beta) \in P): |\alpha| \leq |\beta|$ (dimensione della testa \leq dimensione del corpo): Grammatica di tipo 1
- Tipo 2: $(\forall (\alpha \rightarrow \beta) \in P)(\alpha \in V)(\beta \in (V \cup T)^*)$: $(|\beta| \geq 1)$ (la testa è formata da un solo simbolo non terminale, mentre il corpo può aumentare sia terminali che non terminali, ma almeno un terminale)
Senza il $(|\beta| \geq 1)$ si perde $L2 \subset L1$ (inclusione) potremmo avere $\dim(\text{corpo}) < \dim(\text{testa})$:
Grammatica di tipo 2
- Tipo 3: $(\forall (\alpha \rightarrow \beta) \in P)(\alpha \in V)$ e $\beta: \begin{cases} a \in T \\ aA \text{ con } a \in T, A \in T \end{cases}$: Grammatica di tipo 3

Automi

Ogni classe di linguaggio ha il suo automa riconoscitore:

- L3 sono riconosciuti da **automati a stati finiti** o SFA (State Final Automation)
In questo caso specifico automa deterministico equivale ad automa non deterministico.
- L2 sono riconosciuti da **automati a pila** o FSA (Final Stack Automation) hanno memoria stack infinita, l'accesso è in testa o in coda in lettura o scrittura.

In questo caso l'automa non deterministico è più potente di quello deterministico poiché non tutti i linguaggi $L2$ possono essere riconosciuti da un automa a pila, ad esempio:

$\{\omega\omega^R \mid \omega \in T^*\}$ dove T è l'alfabeto e R indica "reverse"

20 stringhe \rightarrow 20 stati finiti, invece con ∞ stringhe tutto ciò non vale, quindi un linguaggio finito è di tipo 3, ma uno infinito può essere $L2 \vee L1 \vee L0$.

- $L1$ sono riconosciuti da **automi linearmente limitati** non deterministici poiché sono in grado di riconoscere ogni tipo di $L1$, possiamo anche definire la versione deterministica; tuttavia, non sappiamo che relazione c'è tra deterministico e non deterministico, se uno equivale all'altro o se uno è più potente di un altro.
Gli automi linearmente limitati sono simili agli automi a pila, ma permettono accesso in lettura e scrittura ovunque; tuttavia, danno a disposizione una quantità di memoria pari a una funzione lineare sulla dimensione dell'input.
- $L0$ sono riconosciuti dalla macchina di Turing, simile agli automi linearmente limitati come tipologia di accesso, ma mette a disposizione memoria infinita.
Essendo $L3 \subset L2 \subset L1 \subset L0$ la macchina di Turing può riconoscere ogni tipo di linguaggio.
In questo caso deterministico equivale a non deterministico.

Definizione di Alfabeto

Un alfabeto Σ è un insieme finito di simboli: $\Sigma = \{0,1\}$; $\Sigma = \{a, b, ?, c, ciao, M\}$.

In generale possiamo vedere l'alfabeto come un insieme matematico.

Una **stringa** σ su Σ è una sequenza finita di simboli tali che appartengano a Σ .

Preso $\Sigma^* = \{w \mid w \text{ è una stringa di } \Sigma\}$ abbiamo la giustapposizione di simboli di Σ , ovvero l'**insieme di tutte le stringhe** che possiamo costruire con Σ : $\epsilon \in \Sigma^* \forall \epsilon$.

La lunghezza di una stringa si indica con la notazione di modulo (o cardinalità): $|\sigma| = n$.

La stringa sopracitata ϵ corrisponde alla **stringa nulla**, di fatto $|\epsilon| = 0$.

Un esempio di tale notazione può essere: $|0101| = 4$.

È importante evidenziare che la dimensione di una stringa dipende da come è stato definito l'alfabeto al quale fa riferimento.

Ad esempio:

$$\Sigma = \{010, 1\} \rightarrow |0101| = 2$$

In generale abbiamo che Σ^k è l'insieme di tutte le stringhe tali che la loro lunghezza sia pari a k :

$$\Sigma^k = \{w \in \Sigma^* : |w| = k\}$$

E di conseguenza:

$$\Sigma^k = \{\epsilon\} \neq \emptyset$$

È facile notare che $\Sigma^1 \neq \Sigma$ poiché il primo contiene stringhe di lunghezza unitaria mentre il secondo contiene caratteri, di fatto sono isomorfi, ma differiscono per tipo.

Proprietà 1.1

$$(\forall \Sigma)(\forall \sigma \in \Sigma^*)(\sigma \epsilon = \epsilon \sigma = \sigma)$$

Proprietà 1.2

$$\Sigma^* = \bigcup_{i \geq 0} \Sigma^i \quad \Sigma^+ = \bigcup_{i > 0} \Sigma^i$$

Unione di stringhe

$$\sigma = abc \quad \sigma' = a1 \quad \sigma\sigma' = abca1$$

Cardinalità di un alfabeto

Posso ottenere infinite parole di dimensione sempre diversa.

$$|\Sigma^*| = |\Sigma^+| = \infty$$

Sottostringa e casi particolari (Prefisso e Suffisso)

$\sigma' \neq \epsilon$ è **sottostringa** di σ se esistono σ'' e σ''' (anche vuote): $\sigma = \sigma''\sigma'\sigma'''$

Casi particolari ($\sigma''' = \epsilon$):

- Data una stringa σ , $\exists \sigma' \neq \epsilon$ che è **prefisso** di σ e se $\exists \sigma''$ (anche vuota): $\sigma = \sigma'\sigma''$
- Data una stringa σ , $\exists \sigma' \neq \epsilon$ che è **suffisso** di σ e se $\exists \sigma''$ (anche vuota): $\sigma = \sigma''\sigma'$

Definizione di Linguaggio

Fissato un alfabeto Σ , L è un **linguaggio** (su Σ) se $L \subset \Sigma^*$ (L è sottoinsieme di sigma star).

Problema standard:

Dati $\Sigma, \sigma, L \rightarrow \sigma \in L?$

Tale problema è un tipo di problema decisionale il cui codominio è binario.

Definizione: Un problema decisionale è un problema il cui dominio è binario e composto esclusivamente da due valori di booleani, ovvero "vero" o "falso".

Definizione: Un problema di calcolo è un problema computazionale che richiede un output specifico in risposta a un dato input. A differenza di un problema decisionale, che richiede una risposta sì o no, un problema di calcolo richiede una risposta più complessa, che può essere un numero, una stringa, una struttura dati o qualsiasi altro tipo di oggetto computazionale.

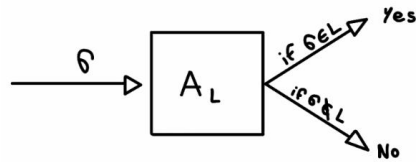
Esempi:

Problema decisionale: "Un grafo è tricolorabile?"

Problema di calcolo: "Mi colori un grafo?"

Automi riconoscitori

Gli automi precedentemente citati sono automi riconoscitori.



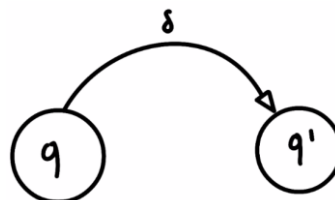
Tali automi possono essere di due tipi:

1. **Deterministici:** un automa viene chiamato deterministico se dato un input esiste una singola computazione, quindi, a parità di input avremo lo stesso output.
2. **Non deterministici:** un automa non deterministico a differenza di uno deterministico non è “probabilistico”, infatti, non esegue una singola computazione, ma tutte le combinazioni possibili, tutto ciò avviene grazie alla clonazione della macchina che sta eseguendo la computazione, per ogni punto di sdoppio.

Deterministic Finite Automation (DFA)

Un automa a stati finiti deterministico è una quintupla $A = (Q, \Sigma, \delta, q_0, F)$, dove:

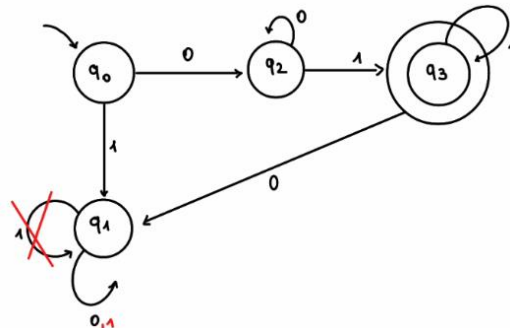
- Q è un insieme **finito** di stati (accettabili);
- Σ è un insieme **finito** dei possibili simboli dell'alfabeto in ingresso all'automa;
- $q_0 \in Q$ è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali marcati come tali (accettanti);
- $\delta: Q \times \Sigma \rightarrow Q$ è la funzione di transizione, restituisce lo stato successivo a partire dallo stato attuale e dall'input letto in quell'istante di tempo: $\delta(q_i, x) = q'$



Quando termina la computazione di un automa?

La computazione di un automa termina nel momento in cui termina la lettura dell'input, in questo modo tale automa si “ferma” su uno stato, e nel nostro caso, se tale stato è finale allora la stringa viene accettata, altrimenti la stringa non viene accettata.

Esempio:



Prendendo in considerazione l'automa in foto, innanzitutto notiamo che lo stato q_1 prende il nome di **stato pozzo** (a causa del loop che a prescindere dall'input rimanda allo stato stesso) e q_3 rappresenta lo stato finale e lo notiamo graficamente dalla rappresentazione mediante due circonferenze concentriche.

Tale automa riconosce linguaggi così definiti $\{0^n, 1^m | n, m \geq 1\}$ (per l'ultima volta sottolineerò che n e m non sono esponenti ma rappresentano la frequenza di 0 e 1 nella stringa).

Per verificare la veridicità di ciò che abbiamo detto effettuiamo dei test iterativi sull'automa:

- Caso positivo $\sigma = 00111$

$$q_0 \xrightarrow{0} q_2 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{1} q_3 \xrightarrow{1} q_3$$

L'automa dopo cinque transizioni ha terminato la lettura dell'input e quindi, trovandosi in uno stato finale, la stringa viene accettata ($\sigma \in L$).

- Caso negativo $\sigma = 11111$

$$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1 \xrightarrow{1} q_1$$

L'automa dopo cinque transizioni ha terminato la lettura dell'input e quindi, **non** trovandosi in uno stato finale, la stringa **non** viene accettata ($\sigma \notin L$), σ di fatto viola le regole di produzione prestabilite, in cui viene espressamente specificata la presenza di almeno uno (≥ 1) zero.

- Caso negativo $\sigma = 00101$

$$q_0 \xrightarrow{0} q_2 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{0} q_1 \xrightarrow{1} q_1$$

L'automa dopo cinque transizioni ha terminato la lettura dell'input e quindi, **non** trovandosi in uno stato finale, la stringa **non** viene accettata ($\sigma \notin L$), σ di fatto viola le regole di produzione prestabilite, in cui viene espressamente specificato che la stringa debba essere costituita da n zeri e poi m uni, quindi uno zero in input dopo x uni è illegale.

Funzione di transizione estesa alle stringhe per DFA (generalizzazione di δ alle stringhe)

1. $\hat{\delta}(q, \epsilon) = q$
2. Se $w = xa, x \in \Sigma^*, a \in \Sigma$

$$\text{Allora } \hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$$

Linguaggio riconosciuto da un automa A

$L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q, w) \in F\}$ corrisponde al linguaggio conosciuto dall'automa A.

Facciamo degli esempi:

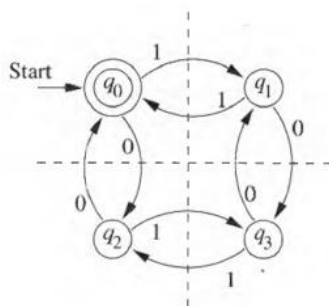
$L = \{w \in \{0,1\}^* \mid w \text{ contiene lo stesso numero di 0 e 1}\}$ **non** è riconoscibile da un automa a stati finiti poiché servirebbe di fatto memoria infinita o perlomeno riusabile.

$L = \{w \in \{0,1\}^* \mid w \text{ contiene numero pari di 0 e 1}\}$ è riconoscibile da un automa a stati finiti:

Tabella degli stati:

	0	1
q_0	P	P
q_1	P	D
q_2	D	P
q_3	D	D

Automa (soluzione grafica):



Automa (tabellare):

	0	1
$\rightarrow q_0$	q_2	q_1
$* q_1$	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

N.B. \rightarrow e $*$ indicano rispettivamente stato iniziale e stato finale.

Equivalenza tra due automi

Due automi A_1 e A_2 si dicono equivalenti quando riconoscono lo stesso linguaggio.

Ogni automa ha $\infty\mathbb{N}$ automi equivalenti in quanto possono sempre esserci stati clone o stati irraggiungibili (inutili).

Non Deterministic Finite Automaton (NFA)

Un automa a stati finiti deterministico è una quintupla $A = (Q, \Sigma, \delta, q_0, F)$, dove:

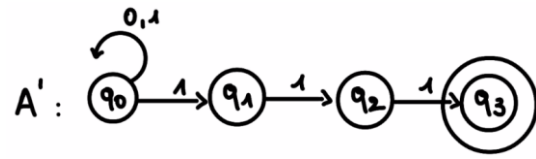
- Q è un insieme **finito** di stati (accettabili);
- Σ è un insieme **finito** dei possibili simboli dell'alfabeto in ingresso all'automa;
- $q_0 \in Q$ è lo stato iniziale;
- $F \subseteq Q$ è l'insieme degli stati finali marcati come tali (accettanti);
- $\delta: Q \times \Sigma \rightarrow Q$ è la funzione di transizione, restituisce l'insieme potenza (**power set**) di Q .

Come salta all'occhio, sintatticamente, NFA e DFA si equivalgono, ma la differenza sta nel fatto che la funzione di transizione di un DFA restituisce uno stato, mentre quella di un NFA restituisce un insieme di stati (**il power set di Q**).

~pag. 56 Libro

L'NFA può essere considerato, a differenza del DFA, un'entità che sceglie la giusta computazione.

Facciamo un esempio di NFA che riconosce il linguaggio $L(A') = \{w \mid 111 \text{ è un suffisso di } w\}$:

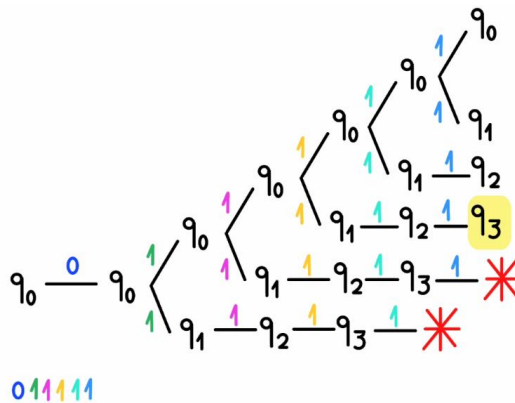


Subito si nota che nello stato iniziale (q_0) sono presenti due transizioni per lo stesso stato (punto di sdoppio), in corrispondenza di questo specifico punto si genera un clone della macchina, infatti, un clone supporterà che l'uno in input sia casuale e nell'altro che l'uno in input sia il primo dei tre uni del suffisso; inoltre, lo stato finale (q_3) non presenta transizioni e la funzione di transizione $\delta(q_0, 1) = \{q_0, q_1\}$ non produrrà più uno stato bensì un insieme di stati.

Quando viene accettata una stringa?

Una stringa σ viene accettata quando almeno uno dei cloni arriva in uno stato finale.

Ma cosa succede durante la lettura dell'input 011111?



Gli “asterischi rossi” nell’immagine rappresentano quella che è una caratteristica intrinseca degli automi non deterministici, ovvero, quando per un dato input la relazione di transizione di stato non è definita (in questo caso $\nexists \delta(q_3, 1), \delta(q_3, 0)$), **il clone in questione muore**.

Visto che nell'esempio fatto il clone numero due si trova nello stato finale al termine della computazione, la stringa 011111 viene accettata in quanto, infatti, rispetta la regola di produzione.

Funzione di transizione estesa alle stringhe per NFA (generalizzazione di δ alle stringhe)

Generalizzazione del dominio δ nel campo delle stringhe: $\hat{\delta}(q, \epsilon) = \{q\}$.

Sia $w = xa, x \in \Sigma^*, a \in \Sigma$

Sia $\hat{\delta}(q, x) = \{p_1, \dots, p_k\}$

Sia $\cup_{i=1}^k \delta(p_i, a) = \{r_1, \dots, r_n\}$

Allora: $\hat{\delta}(q, w) = \{r_1, \dots, r_n\}$

**Se abbiamo una stringa w applicando δ sul primo simbolo otteniamo un sistema di stati, e stessa cosa avviene con il secondo, con il terzo e così via, alla fine unisco l'insieme di tutti gli stati ottenuti.*

L'automa (NFA) A che linguaggio riconosce

$$A = (Q, \Sigma, \delta, q_0, F) \\ L(A) = \{w \in \Sigma^* \mid \delta^*(q_0, w) \wedge F \neq \emptyset\}$$

Nell'esempio precedente $\hat{\delta}(q_0, 011111) = \{q_0, q_1, q_2, q_3\} \neq \emptyset$

Definizione semantica:

Esiste almeno un path (cammino) che ha come stato d'arrivo lo stato finale.

Problema dell'equivalenza tra DFA e NFA

Accezione di equivalenza ricorrente nel corso:

Due automi qualsiasi si dicono equivalenti se riconoscono lo stesso linguaggio, ovvero, se calcolano la stessa funzione booleana.

Teorema 1.1: Esistenza della controparte deterministica/non deterministica

$$DFA \equiv NFA \rightarrow \forall DFA (\text{o } FA) \exists NFA \wedge \forall NFA \exists DFA$$

Dato un NFA il corrispondente DFA è più grande, e dato un DFA il corrispondente NFA è più piccolo (in genere).

Solitamente la dimensione coincide ma in casi sfavorevoli può capitare che per un NFA di n stati avremo un DFA di dimensione 2^n .

Cosa cambia tra DFA e NFA a livello notazionale?

- DFA: $A_D = (Q, \Sigma, \delta_D, q_0, F_D)$
- NFA: $A_N = (Q, \Sigma, \delta_N, q_0, F_N)$
- $\forall q, a (\delta_N(q, a) = \{\delta_D(q, a)\})$ *la differenza è solo di due graffe in più*

Trasformazione

Gli automi finiti deterministici e non riconoscono la stessa classe di linguaggi. Tale equivalenza è sorprendente in quanto gli NFA sembrano avere maggiore potere computazionale rispetto ai DFA.

- N (Non deterministico) = $(Q_N, \Sigma, \delta_N, q_0, F_N)$, dove $F_N \subseteq Q_N$
- D (Deterministico) = $(Q_D, \Sigma, \delta_D, q_0, F_D)$, dove $F_D \subseteq Q_D$

Abusiamo della notazione insiemistica per descrivere ogni **singolo stato** (non insieme di stati) dell'automa D:

$$Q_D \rightarrow \{q_0, q_1\} \text{ oppure } q_0, q_1\{$$

$L(D) = L(N) \rightarrow$ costruiamo D tale che sia **equivalente** ad N.

$$Q_D = 2^{Q_N} \quad F_D = \{S \subseteq Q_N \mid S \wedge F \neq \emptyset\}$$

Dove:

- Q_D è l'insieme dei sottoinsiemi di Q_N vale a dire; Q_D è l'insieme potenza di Q_N . Osserviamo che se Q_N ha n stati, allora Q_D avrà 2^n stati. Spesso dallo stato iniziale di Q_D non è possibile raggiungere tutti questi stati. Gli stati inaccessibili possono essere eliminati. Dunque, in effetti, il numero di stati di D può essere molto inferiore a 2^n ;

- S è un sottoinsieme di stati e $\forall S \subseteq Q_N \quad \delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$;
- F_D è l'insieme dei sottoinsiemi S di Q_N , in tale sottoinsieme saranno presenti alcuni stati di Q_N . Tra gli stati presenti all'interno di F_N saranno finali tutte quelle rappresentazioni di stato che avranno intersezione non vuota con F_N : $\{S \wedge F_N \neq \emptyset\} \subseteq F_D$.

$$w \rightarrow \{q_0, \dots, q_n\} \quad D \rightarrow \{\emptyset, \{q_0\}, \{q_1\}, \dots, \{q_0, q_3, q_7\} \dots\}$$

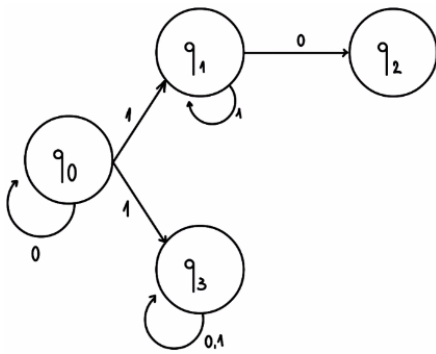
*Ricordiamo che se N ha n stati D ne avrà 2^n

Inoltre, se abbiamo un insieme di stati in Q_N che formano uno stato in Q_D sull'input a applichiamo δ_D per ogni stato tra quelli in questione e li uniamo volta per volta:

$$(\forall s \subseteq Q_N)(\forall a \in \Sigma) \quad \delta_D(s, a) = \bigcup_{p \in s} \delta_N(p, a)$$

Dove δ_D è definita su ogni stato di D , ma $Q_D = 2^{Q_N}$ insieme di sottoinsiemi di rappresentazioni di stato di N .

Esempio 1:

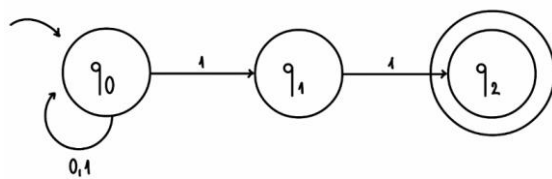


Funzioni di transizione di stato:

$$\delta_D(\{q_0\}, 1) = \bigcup_{p \in \{q_0\}} \delta_N(p, 1) = \{q_1, q_3\}$$

$$\delta_D(\{q_1, q_3\}, 0) = \bigcup_{p \in \{q_1, q_3\}} \delta_N(p, 0) = \{q_2, q_3\}$$

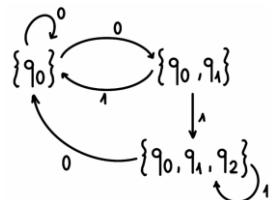
Esempio 2:



Un NFA a $n = 3$ stati avrà un DFA equivalente a $2^n = 8$ stati:

$Q_D \setminus \Sigma$	0	1
\emptyset	\emptyset	\emptyset
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_1\}$	\emptyset	$\{q_2\}$
$* \{q_2\}$	\emptyset	\emptyset
$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$
$* \{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$* \{q_1, q_2\}$	\emptyset	$\{q_2\}$
$\{q_0, q_1, q_2\}$	$\{q_0\}$	$\{q_0, q_1, q_2\}$

Grafo di accessibilità:



La tabella rappresenta il corrispondente DFA (infatti ha 8 righe, ovvero 8 stati) e, come si può notare, gli unici tre stati raggiungibili sono $\{q_0\}, \{q_0, q_1\}, \{q_0, q_1, q_2\}$ perciò, come si nota dal grafo di accessibilità, possiamo riportare solo questi tre stati nel nostro DFA, a riconferma della frase: "Solitamente la dimensione coincide ma in casi sfavorevoli può capitare che per un NFA di n stati avremo un DFA di dimensione 2^n ".

Quando uno stato è raggiungibile?

Gli stati raggiungibili sono tutti quelli che producono da essi su un input qualsiasi, ovvero tutti quelli che non producono in nessun caso come output \emptyset .

Teorema 1.2: Appartenenza di una stringa

Data una stringa w se appartiene al linguaggio riconosciuto dall'automa D allora apparterrà anche al linguaggio riconosciuto dall'automa N .

1. $\forall w \in \Sigma^* \quad w \in L(D) \rightarrow w \in L(N)$
2. $\forall w \in \Sigma^* \quad (\delta_D(\{q_0\}, w) = \delta_N(q_0, w))$

*La 2 non è altro che un proxy della 1.

Tale teorema si costruisce per induzione strutturale su w .

Dimostrazione:

- Caso base $|w| = 0$ ($w = \epsilon$):

$$\delta_v(q_0, \epsilon) = \{q_0\} = \delta_D(\{q_0\}, \epsilon)$$

- Ipotesi induttiva:

$$(\forall w \in \Sigma^*) (|w'| \leq n) \quad \widehat{\delta}_N(q_0, w') = \widehat{\delta}_D(\{q_0\}, w')$$

- Caso base $|w| = n + 1$ (w stringa generica):

$$w = xa, \forall x \in \Sigma^*, a \in \Sigma, |x| = n$$

$$\begin{aligned} \widehat{\delta}_N(q_0, x) &= \widehat{\delta}_D(\{q_0\}, x) \\ &= \{p_1, \dots, p_k\} \\ \widehat{\delta}_N(q_0, w) &= \widehat{\delta}_N(q_0, xa) = \bigcup_{i=1}^k \delta_N(p_i, a) \end{aligned}$$

*Dalla funzione di transizione estesa alle stringhe per NFA

$$\delta_D(\{p_1, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a)$$

$$\widehat{\delta}_D(\{q_0\}, w) = \delta_D(\widehat{\delta}_D(\{q_0\}, x), a) = \delta_D(\{p_1, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a) = \widehat{\delta}_N(q_0, w)$$

Teorema 1.3: DFA e NFA equivalenti riconoscono lo stesso linguaggio

Dato un DFA possiamo banalmente costruire un NFA, viceversa per ogni NFA in modo più complesso possiamo costruire un DFA.

$$L(D) = \{w \in \Sigma^* \mid \widehat{\delta}_D(\{q_0\}, w) \in F_D\} = \{w \in \Sigma^* \mid \widehat{\delta}_D(\{q_0\}, w) \wedge F_N \neq \emptyset\} = \{w \in \Sigma^* \mid \widehat{\delta}_N(q_0, w) \wedge F_N \neq \emptyset\} = L(N)$$

*Dalla definizione di F_D

Teorema 1.4: Condizione necessaria e sufficiente per accettare un linguaggio

Un linguaggio L viene accettato da un automa a stati finiti deterministico (DFA) se e solo se viene accettato da un automa a stati finiti non deterministico (NFA).

Dal libro 2 a pagina 68, il teorema 2.12 la dimostrazione del teorema 1.4:

La parte "se" è costituita dalla costruzione per sottoinsiemi e dal teorema 1.3.

La parte “solo se” è facile, bisogna solamente convertire un DFA in un NFA identico.

In modo induttivo, se abbiamo il diagramma di transizione di un DFA, possiamo anche interpretarlo come il diagramma di transizione di un NFA, in cui c'è esattamente una scelta di transizione in qualunque caso. In termini più formali, sia $D = (Q, \Sigma, \delta_D, q_0, F)$ un DFA. Definiamo l'NFA equivalente $N = (Q, \Sigma, \delta_N, q_0, F)$, dove δ_N è definita dalla seguente regola.

Se $\delta_D(q, a) = p$, allora $\delta_N(q, a) = \{p\}$.

È facile dimostrare, per induzione su $|w|$, che se $\widehat{\delta_D}(q_0, w) = p$ allora $\widehat{\delta_N}(q_0, w) = \{p\}$, infatti l'ultimo step della dimostrazione del teorema 1.2 è:

$$\widehat{\delta_D}(\{q_0\}, w) = \delta_D(\widehat{\delta_D}(\{q_0\}, x), a) = \delta_D(\{p_1, \dots, p_k\}, a) = \bigcup_{i=1}^k \delta_N(p_i, a) = \widehat{\delta_N}(q_0, w)$$

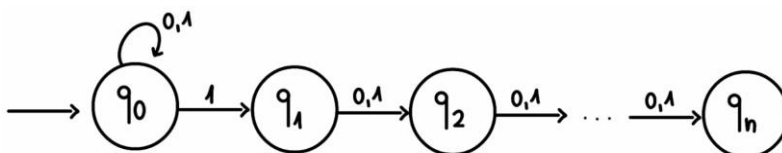
Di conseguenza la stringa w viene accettata da D se e solo se viene accettata da N , cioè $L(D) = L(N)$.

Caso sfavorevole di costruzione per sottoinsiemi e principio della colombaria

Per ricollegarci alla frase, già riportata più volte: “Solitamente la dimensione coincide ma in casi sfavorevoli può capitare che per un NFA di n stati avremo un DFA di dimensione 2^n ” citiamo il **principio della colombaria**:

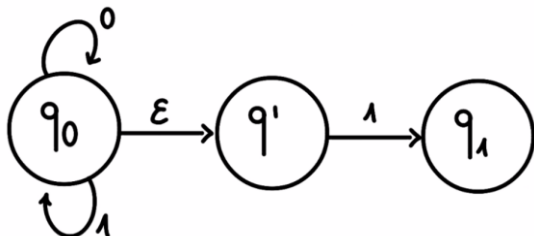
“Se abbiamo una colombaria con k cassette e n colombi ($n > k$), esisterà sempre un foro che contiene almeno due colombi.”

Esempio di costruzione di un NFSFA il cui corrispondente DSFA ha almeno 2^n stati, dove n sono gli stati dell'NSFA:



L'automa in foto riconosce stringhe per cui l'ennesimo carattere prima della fine sia un uno, e quindi, dovendo riconoscere almeno 2^n caratteri deve avere almeno 2^n stati.

Le macchine con ϵ -transizioni



Ogni macchina con ϵ -transizioni può essere ridotta a una senza, tali macchine sono intrinsecamente **non deterministiche**.

Le ϵ -transizioni vengono usate negli analizzatori lessicali, in quanto sono molto comode in questi casi.

Automi con ϵ -transizioni

Questi speciali automi non vengono usati solo negli analizzatori lessicali, ma trovano ampio utilizzo anche nell'ambito delle analisi di DNA, RNA e catene amminoacidiche.

Tali automi si ricollegano subito a un tipo di notazione particolare per definire linguaggi, ovvero le **espressioni regolari**.

Espressioni regolari

Le "Espressioni Regolari" o Regular Expression" (in java: regex), sono una notazione per definire linguaggi che, a differenza delle produzioni, a meccanismo dichiarativo, funzionano a meccanismo algebrico.

Oltre a definire linguaggi, tali espressioni sono in grado di compattare la notazione rimanendo ben leggibili, a differenza di altre notazioni, per esempio nell'ambito degli indirizzi ai siti web.

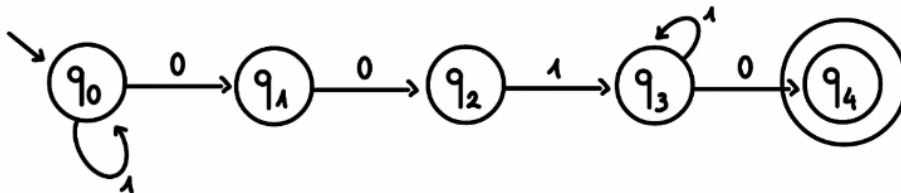
Facciamo subito un esempio:

$\Sigma = \{0, 1\}$ $e = \underline{1^*} \underline{00} \underline{0} \underline{1^+} \underline{0}$ (dove "e" è l'espressione regolare)

Perché sono sottolineati?

È una distinzione che verrà usata solo in questo esempio, tuttavia risulta utile per comprendere che, in realtà non abbiamo a che fare con gli 0,1 di Σ , ma con delle denotazioni di tali simboli (terminali e non terminali).

Di fatto e denota una semantica che è un linguaggio regolare il cui automa è:



Prima di procedere, analizziamo l'espressione e :

- $\underline{1^*}$: il simbolo "*" prende il nome di "**Star di Kleene**" e indica la ripetizione di un oggetto (in questo caso 1) zero o più volte;
- $\underline{00}$: concatenazione o giustapposizione, tali oggetti devono apparire "consecutivamente" nelle stringhe generate dal linguaggio;
- $\underline{1^+}$: il simbolo "+" indica la ripetizione di un oggetto (in questo caso 1) una o più volte.

Un alfabeto è un insieme di stringhe, mentre, un'espressione è una meta-stringa che denota un linguaggio.

Operazioni con i linguaggi

- In quanto insiemi matematici abbiamo:
 - \cup : unione
 - \cap : intersezione
 - $()^c$: complemento, ovvero: $(L)^c = \Sigma^* - L$
- In quanto insiemi di stringhe abbiamo:
 - Concatenazione di linguaggi: $L.M = \{xy | x \in L, y \in M\}$
 - Star di Kleene applicata al linguaggio: $L^* = \{x^k | x \in L, k \geq 0\}$ ad esempio
 $\{0, 11\}^* = \{\epsilon, 0, 11, 00, 1111, 011, \dots\}$
 - $L^+ = \{x^k | x \in L, k \geq 1\}$

Casi di un linguaggio regolare

- ϵ è un'espressione e denota $\{\epsilon\}$
- \emptyset è un'espressione e denota $\{\emptyset\}$
- Per $a \in \Sigma$, a è un'espressione e denota $\{a\}$
- Ogni variabile denota un linguaggio arbitrario:
$$\begin{cases} L(\epsilon) = \{\epsilon\} \\ L(\emptyset) = \{\emptyset\} \\ L\{a\} = \{a\} \end{cases}$$

Tramite il potente strumento dell'induzione possiamo dire che:

- $(E + F)$ è un'espressione regolare e denota $L(E) \cup L(F) \rightarrow L(E + F) = L(E) + L(F)$
- $(E.F)$ è un'espressione regolare e denota $L(E) \cap L(F) \rightarrow L(E.F) = L(E).L(F)$
- E^* è un'espressione regolare e denota $L(E^*) = (L(E))^*$
- E è un'espressione regolare e denota $L((E)) = L(E)$

Ricordiamo che la priorità è:

1. Star di Kleene
2. Intersezione
3. Unione

Esercizio:

“Costruire un'espressione regolare che denoti il linguaggio costituito da tutte le stringhe di 0 e 1 alternati”

Idea 1: $((01)^*0^+(10)^*)^*((10)^*1^+(01)^*)^*$

Sbagliata! Posso ottenere stringhe di tipo “01010000 ...” oppure di tipo “01011111 ...” che non appartengono al linguaggio.

Idea 2: $(01)^* + (10)^*$

Parzialmente sbagliata! Non vengono generate tutte le stringhe possibili, è giusta invece:

$$(01)^* + (10)^* + 1(01)^* + 0(10)^*$$

Idea 3 (parte giusta dell'idea 2 ma più elegante): $(\epsilon + 1)(01)^*(\epsilon + 0)$

Genera tutte le stringhe possibili.

Teorema 1.5: Esistenza ER per $L(A)$

NB: D'ora in poi mi avvarrò della licenza poetica, abusando della notazione insiemistica per semplificare l'espressione "è un'espressione regolare" in " $\in ER$ ".

Sia $L = L(A)$, con $A \rightarrow DFA$, allora esisterà $R \in ER$ tale che $L = L(R)$

La dimostrazione si trova a pagina 98 del libro 2, la riporteremo per completezza ma **non fa parte del programma**.

Supponiamo che gli stati di A siano $\{1, 2, \dots, n\}$ per un intero n .

Comunque siano definiti gli stati di A , ce ne saranno n per un n finito, e rinominandoli ci si può riferire agli stati in questo modo, come se fossero i primi n interi positivi.

Il nostro primo e più difficile compito consiste nel costruire una collezione di espressioni regolari che descrivano insiemi via via più ampi di cammini nel diagramma di transizione di A .

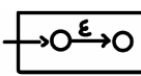
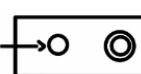
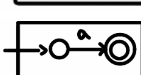
Usiamo $R_{ij}^{(k)}$ come nome dell'espressione regolare il cui linguaggio è l'insieme di stringhe w tale che w sia l'etichetta di un cammino dallo stato i allo stato j in A , e il cammino non abbia alcun nodo intermedio il cui numero sia maggiore di k .

Si noti che i punti di partenza e di arrivo del cammino non sono intermedi; dunque non si richiede che i e j siano inferiori o uguali a k .

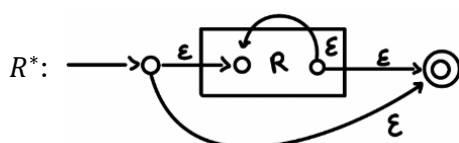
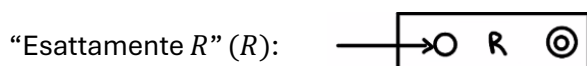
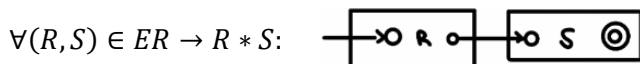
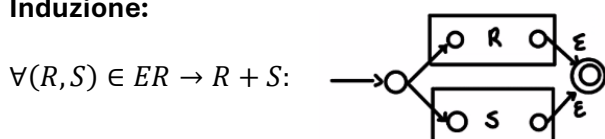
Teorema 1.6: Simmetrico all'1.5

Sia $L = L(R)$, con $R \in ER$, allora $\exists A \rightarrow DFA: L = L(A)$

Dimostrazione per induzione strutturale:

- ε costruiamo tale sotto-automa: 
- \emptyset costruiamo tale sotto-automa: 
- a costruiamo tale sotto-automa: 

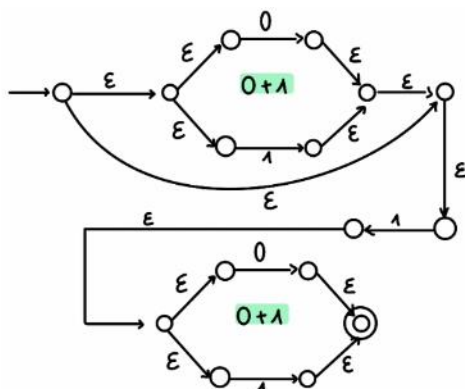
Induzione:



In accordo con la logica induttiva, se tutto ciò vale per il caso base, vale per tutti gli altri, ovvero il teorema vale.

Esercizio:

“Costruire un ϵ NFA che riconosca $(0 + 1)^*1(0 + 1)$ ”



Proprietà delle espressioni regolari

- Unione:
 - Commutativa: $L + M = M + L$
 - Associativa: $(L + M) + N = L + (M + N)$
 - \emptyset neutro: $\emptyset + L = L + \emptyset = L$
 - Distributiva: $L(M + N) = LM + LN$ e $(M + N)L = ML + NL$
 - Idempotente: $L + L = L$
- Concatenazione:
 - Non commutativa: $LM \neq ML$
 - Associativa: $(LM)N = L(MN)$
 - ϵ neutro: $L\epsilon = \epsilon L = L$
 - \emptyset annichilatore: $\emptyset L = L\emptyset = \emptyset$
 - Non idempotente: $L.L \neq L$
- Star di Kleene:
 - Non idempotente: $(L^*)^* \neq L^*$
 - $\emptyset^* = \{\epsilon\}$
 - $\epsilon^* = \epsilon$
 - $L^+ = L.L^* = L^*L$
 - $L^* = L^+ + \epsilon$

Pumping Lemma (per i linguaggi regolari o “gioco a due”)

Come si risponde alla domanda: “come si dimostra che un linguaggio è regolare?”

Sappiamo che se un linguaggio regolare è tale se:

1. Esiste un linguaggio di tipo tre che lo genera o, quantomeno, una grammatica regolare che lo genera.
Mostrare una tra questi due oggetti citati potrebbe essere un’alternativa per dimostrare la regolarità di un linguaggio.
2. Si potrebbe mostrare un automa (a stati finiti) che riconosca tale linguaggio.
3. Si potrebbe mostrare un’espressione regolare (regex) che lo definisca.

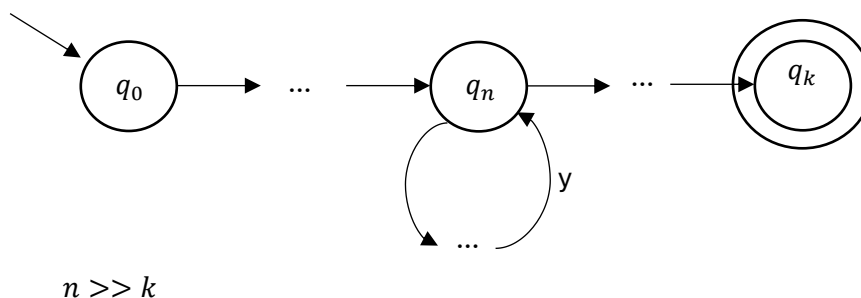
Un’alternativa molto più formale consiste nell’utilizzo del **pumping lemma**, ovvero:

Se L è regolare allora esisterà una costante che dipende (solo) dal linguaggio tale che $\forall w \in L, |w| \geq n$ esiste una scomposizione di w in $w = xyz$ (sottostringhe) tale che valgano le seguenti condizioni:

1. $y \neq \varepsilon$
2. $|xy| < n$
3. $xy^iz \in L \quad \forall i \geq 0$

Funziona perché: se c'è un DFA (a k -stati) che riconosce tale linguaggio, per una stringa $w, |w| = n$ ($n > k$) per il principio della colombaria verrà ripercorso più di una volta un certo percorso che riconoscerà sempre la stringa (y).

Graficamente:



$xyz \in L \rightarrow$ ipotesi fondamentale.

Se un linguaggio viola il pumping lemma, allora non è regolare.

Dimostrazione del Pumping Lemma per linguaggi regolari

Utilizziamo il pumping lemma nel verso negativo, cosa significa? Significa che non lo usiamo per dimostrare che un linguaggio è regolare, ma al contrario, lo usiamo per dimostrare che il linguaggio in questione non è regolare.

$(\forall w \in L)(|w| \geq n)(\exists xyz = w)$ dove “ $xyz = w$ ” è una decomposizione, deve rispettare le condizioni:

- a. $y \neq \varepsilon$
- b. $|xy| < n$
- c. $(\forall i)(xy^iz \in L)$

$$L \text{ reg.} \rightarrow \exists A = (Q_A, \Sigma_A, \delta_A, q_{0A}, F_A) \quad DFA$$

$$L = L(A)$$

$$n = |Q_A|$$

$$w = a_1, a_2, \dots, a_m$$

$$m > n \rightarrow \text{necessario}$$

Definiamo lo stato i -esimo in cui si arriva dopo aver letto la stringa:

$$p_i = \hat{\delta}_A(q_{0A}, (a_1, \dots, a_i))$$

Siccome l'automa legge una stringa di lunghezza $m > n$ (dove n è il numero di stati), per il principio della colomba possiamo affermare che esistono due stati che coincidono $p_i = p_j$.

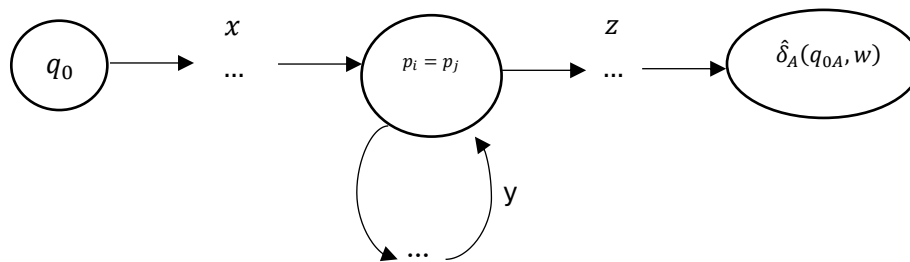
$$\exists (1 \leq i \leq j \leq m) \quad p_i = p_j$$

$$w = xyz$$

$$x = a_1, \dots, a_i \rightarrow \text{prefisso da } [a_1, a_i]$$

$$y = a_{i+1}, \dots, a_j \rightarrow \text{suffisso da } [a_{i+1}, a_j]$$

$$z = a_{j+1}, \dots, a_m \rightarrow \text{suffisso da } [a_{j+1}, a_m]$$



$$\hat{\delta}_A(q_{0A}, w) = \hat{\delta}_A(q_{0A}, xy^kz)(\forall x)$$

Prototipi di linguaggi regolari, liberi o non liberi (da contesto)

Esempi:

1. Libero: $\{0^n 1^n | n \geq 1\}$; $\{0^n 1^k | n \geq k\}$
2. Regolare: $\{0^n 1^k | n \geq k, k < 7\} \rightarrow$ essendo finito può essere riconosciuto da un FA
3. Non libero: $\{0^n 1^n 2^n | n \geq 1\} \rightarrow$ è dipendente da contesto, di tipo 1

Un linguaggio è regolare quando conta, di fatto, un solo oggetto in maniera indefinita (**tutti i linguaggi finiti sono regolari**).

Esercizio: dimostriamo che $L = \{0^n 1^n | n \geq 1\}$ non è regolare:

Prendiamo una stringa $w = 0^k 1^k, w \in L \quad n = k$

$$|w| = 2k < n$$

Prendiamo **una qualsiasi** scomposizione $w = xyz \mid y \neq \varepsilon \wedge xy > n$ poiché abbiamo una stringa composta da k-zeri e k-uno, allora $xy = 0^\alpha$ per qualche α , sappiamo che $y \neq \varepsilon$, prendendo xz notiamo che al suo interno avremo almeno uno zero in meno degli 1, quindi possiamo concludere che se $xz \notin L$ violando il pumping lemma e perciò il linguaggio non è regolare.

Altre proprietà dei linguaggi regolari

NB: D'ora in poi mi avvarrò della licenza poetica, abusando della notazione insiemistica per semplificare l'espressione "è un linguaggio regolare" in " $\in LR$ ".

Sono tutte da dimostrare per l'esame:

1. Se $L, M \in LR$:
 - a. $L \vee M$
 - b. $L \wedge M$
 - c. $L - M$

I linguaggi regolari sono chiusi rispetto a unione, intersezione e differenza.

2. Dato un linguaggio regolare anche il suo complementare è regolare.
3. Se L, M sono regolari, allora $L + M$ è regolare (concatenazione).
4. Se L è regolare, allora L^* è regolare.
5. Se L è regolare, allora L^R è regolare (linguaggio inverso "al contrario").
6. Se ρ è un omomorfismo sia $\rho(L)$ che $\rho^{-1}(L)$ è regolare.
Omomorfismo: $\rho(L) = \{ab^n, ccb^n\}$ è un mapping che sostituisce ogni simbolo con una stringa di simboli.

Dimostrazione per l'unione:

Se L, M sono regolari, esistono due espressioni regolari R, M tali che:

$L = L(R)$ (il linguaggio generato da R è proprio L) e $M = L(S)$ (il linguaggio generato da S è proprio M).

Per definizione $L(R + S) = L(R) \vee L(S) = L \vee M$, essendo $(R + S)$ un'espressione regolare anche $L \vee M$ è un'espressione regolare poiché generata da $(R + S)$.

Dimostrare che se L è regolare allora lo è anche il suo complementare $L^c = (\Sigma^* - L) \rightarrow xz \notin L$

Se L è regolare, esiste un DFA:

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

$$L = L(A)$$

$$B = (Q, \Sigma, \delta, q_0, Q - F)$$

$$L^c = L(B) \rightarrow \text{va dimostrato}$$

Una stringa appartiene ad L se e solo se non appartiene ad L^c ...completare per esercizio.

Dimostrare che l'insieme dei linguaggi regolari è chiuso rispetto all'intersezione:

$$L \wedge M = (L^c \vee M^c)^c \rightarrow \text{De Morgan}$$

Breve refresh su De Morgan:

1. $\overline{(A \wedge B)} = \bar{A} \vee \bar{B}$
2. $A \wedge B = \overline{(\bar{A} \vee \bar{B})}$

Quanto costa produrre da una rappresentazione a un'altra dello stesso linguaggio?

- $O(n^3 \cdot 2^n)$: costo di produzione di un DFA che riconosca lo stesso linguaggio riconosciuto da un NFA con o senza ϵ -transizioni.
- $O(n)$ (Costo lineare): costo di produzione di un NFA che riconosca lo stesso linguaggio riconosciuto da un DFA.
- $O(n^3 \cdot 4^{n^3 \cdot 2^n}) = O(n^3 \cdot 2^n) + O(n)$: costo di produzione di un'espressione regolare che produca lo stesso linguaggio riconosciuto da un DFA.
- $O(n^3)$: costo di produzione di un NFA a partire da un'espressione regolare.
- $O(x)$ dove $x \in [1, n^2]$: costo per capire se un linguaggio è vuoto tramite un FA.

NB: n è il numero di stati.

Dimostrare che $L = \{1^k \mid k \text{ sia un quadrato perfetto}\}$ non è regolare.

$$w_i \{ \epsilon, 1, 1111, \dots \}$$

NB: quadrato perfetto vuol dire che $\sqrt{x} \in \mathbb{N}$.