Fondamenti di informatica

UniVR - Dipartimento di Informatica

Fabio Irimie

1° Semestre 2025/2026

Indice

1 Introduzione

1.1 Cos'è l'informatica?

È una scienza che studia la calcolabilità, cioè cerca di capire che problemi si possono risolvere con un programma. Nasce dall'unione di matematica, ingegneria e logica. Il computer è solo uno strumento, mentre la matematica è il linguaggio con cui si creano algoritmi che permettono di risolvere i problemi.

1.2 Origini dell'informatica

Hilbert, nel 1900, si pose l'obiettivo di formalizzare tutta la matematica con un insieme finito e non contraddittorio di assiomi. Nel 1931, invece, Gödel dimostrò che l'informatica non potrà mai rappresentare tutta la matematica, perché ci saranno sempre proposizioni vere ma non dimostrabili tramite il calcolo. Ci si iniziò a chiedere se esistessero modelli di calcolo meccanici in grado di risolvere tutti i problemi. Nel 1936, Turing propose la macchina di Turing, una **sola** macchina programmabile in grado di risolvere tutti i problemi risolvibili:

$$\mathrm{Int}(P, x) = \begin{cases} P(x) & \text{se } P(x) \text{ termina} \\ \uparrow & \text{se } P(x) \text{ non termina} \end{cases}$$

dove P è un programma e x è un input. La macchina di Turing è un modello teorico di calcolatore, che non esiste fisicamente, ma è in grado di simulare qualsiasi altro calcolatore. Da questo modello deriva la concezione di calcolabilità, cioè se un problema è intuitivamente calcolabile, allora esiste un programma in grado di risolverlo.

Altri modelli di calcolo che sono stati proposti sono:

- Lambda-calcolo
- Funzioni ricorsive
- Linguaggi di programmazione (Turing-completi)

Definizione utile 1.1. La Turing-completezza è la proprietà di un linguaggio di programmazione di essere in grado di simulare una macchina di Turing, cioè di poter risolvere qualsiasi problema risolvibile.

1.2.1 Calcolabilità

Un programma è calcolabile se termina, ma non è detto che termini in un tempo ragionevole. Non esistono algoritmi che possono dire se un programma termina o meno. Questo è un esempio di problema non calcolabile.

I problemi non calcolabili sono infinitamente più numerosi di quelli calcolabili

1.3 Nozioni di base

1.3.1 Basi di logica

Alcune nozioni di logica che ci serviranno in seguito:

- **Linguaggio del primo ordine:**
  - Simboli relazionali $(p, q, ...)$
  - Simboli di funzione $(f, g, ...)$
  - Simboli di costante $(c, d, ...)$
- **Simboli logici:**
  - Parentesi $(,)$ e virgola
  - Insieme numerabile di variabili $(\nu, x, ...)$
  - Connettivi logici $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$
  - Quantificatori $(\forall, \exists)$
- **Termini:**
  - Variabili
  - Costanti
  - f simbolo di funzione m-ario $t_1, t_2, \ldots, t_m$ termini, allora $f(t_1, t_2, \ldots, t_m)$ è un termine.
- **Formula atomica:** p simbolo di relazione n-ario, $t_1, t_2, \ldots, t_n$ termini, allora $p(t_1, t_2, \ldots, t_n)$ è una formula atomica.
- **Formula:**
  - Formula atomica
  - ϕ formula, allora $\neg\phi$ è una formula
  - ϕ e ψ formule, allora $(\phi \wedge \psi)$, $(\phi \vee \psi)$, $(\phi \rightarrow \psi)$, $(\phi \leftrightarrow \psi)$ sono formule.
  - ϕ formula e ν variabile, alloar e $\forall\nu.\phi$ e $\exists\nu.\phi$ sono formule.

1.3.2 Nozioni sugli insiemi

- $x \in A$ signfiica che x è un elemento dell'insieme A
- $\{x|P(x)\}$ si identifica insieme costuito dagli x che soddisfano la proprietà (o predicato) $P(x)$
- $A \subseteq B$ significa che A è un sottoinsieme di B se ogni elemento di A è anche in B
- $\mathcal{P}(S)$ denota l'insieme delle parti di S, ovvero l'insieme di tutti i sottoinsiemi di S $(\mathcal{P}(S) = \{X|X \subseteq S\})$
- $A \backslash B = \{x|x \in A \wedge x \notin B\}, A \cup B = \{x|x \in A \vee x \in B\}, A \cap B = \{x|x \in A \wedge x \in B\}$
- $|A|$ denota la cardinalità di A, ovvero il numero di elementi in A.
- $\bar{A}$ denota il complemento di A, ovvero $x \bar{\in} A \leftrightarrow x \notin A$

1.3.3 Nozioni sulle relazioni

- Prodotto cartesiano:

$$A_1 \times A_2 \times \cdots \times A_n = \{\langle a_1, a_2, \ldots, a_n\rangle | a_1 \in A_1, \ldots, a_n \in A_n\}$$

- Una **relazione** (binaria) è un sottoinsieme del prodotto cartesiano di (due) insiemi; dati A e B, $R \subseteq A \times B$ è una relazione su A e B
  - **Riflessiva:** $\forall a \in S$ si ha che aRa
  - **Simmetrica:** $\forall a, b \in S$ se aRb allora bRa
  - **Antisimmetrica:** $\forall a, b \in S$ se aRb e bRa allora $a = b$
  - **Transitiva:** $\forall a, b, c \in S$ se aRb e bRc allora aRc
- Per ogni relazione $R \subseteq S \times S$ la chiusura transitiva di R è il più piccolo insieme R^* tale che $\langle a, b\rangle \in R \wedge \langle b, c\rangle \in R \rightarrow \langle a, c\rangle \in R^*$
- Una relazione è detta **totale** su S se $\forall a, b \in S$ si ha che $aRb \vee bRa$
- Una relazione R di *di equivalenza* è una relazione binaria riflessiva, simmetrica e transitiva.
- Una relazione binaria $R \subseteq S \times S$ è un **pre-ordine** se è riflessiva e transitiva.
- R è un ordine parziale se è un pre-ordine antisimmetrico.
- $x \in S$ è **minimale** rispetto a R se $\forall y \in S.y \not R x$ (ovvero $\neg(yRx)$)
- $x \in S$ è **minimo** rispetto a R se $\forall y \in S.xRy$
- $x \in S$ è **massimale** rispetto a R se $\forall y \in S.x \not R y$ (ovvero $\neg(xRy)$)
- $x \in S$ è **massimo** rispetto a R se $\forall y \in S.yRx$

1.3.4 Nozioni sulle funzioni

- Una relazione f è una **funzione** se $\forall a \in A$ esiste uno ed un solo $b \in B$ tale che $(a, b) \in f$
- A dominio e B codominio di f. Il range di f è l'insieme di tutti i valori che f può assumere.
- f è **iniettiva** se $\forall a_1, a_2 \in A$ se $a_1 \neq a_2$ allora $f(a_1) \neq f(a_2)$
- Se $f : A \mapsto B$ è sia iniettiva che suriettiva allora è **biiettiva** e quindi esiste $f^{-1} : B \mapsto A$

2 Funzioni calcolabili

Un insieme è una proprietà ed è rappresentato da una funzione che indica se un elemento appartiene o meno all'insieme. I problemi da risolvere (in questo corso) hanno come soluzione una funzione sui naturali:

$$f : \mathbb{N} \to \mathbb{N}$$

Questo tupo di funzione è un **insieme** di associazioni input-output. Un esempio è la funzione quadrato:

$$f = \text{quadrato} = \{(0,0),(1,1),(2,4),(3,9),\ldots\} = \{(n, n^2) | n \in \mathbb{N}\}$$

Quindi f è un insieme di coppie in $\mathbb{N} \subseteq \mathbb{N} \times \mathbb{N}$ la cui cardinalità è: $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$. Di conseguenza la funzione è un sottoinsieme di $\mathbb{N} \times \mathbb{N}$:

$$f \subseteq \mathbb{N} \times \mathbb{N} \quad f \in \mathcal{P}(\mathbb{N} \times \mathbb{N})$$

Dove $\mathcal{P}(\mathbb{N} \times \mathbb{N})$ è l'insieme delle parti di $\mathbb{N} \times \mathbb{N}$, cioè l'insieme di tutti i sottoinsiemi di $\mathbb{N} \times \mathbb{N}$.

Il numero di funzioni è:

$$f : \mathbb{N} \to \mathbb{N} = |\mathcal{P}(\mathbb{N} \times \mathbb{N})| = |\mathcal{P}(\mathbb{N})|$$

Esempio 2.1. Un esempio di insieme delle parti per l'insieme $A = \{1,2,3\}$ è:

$$\mathcal{P}(A) = \{\varnothing, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$$

E le cardinalità sono:

$$|A| = 3 \quad |\mathcal{P}(A)| = 8 = 2^3$$

$$\downarrow$$

$$|\mathbb{N}| = \omega < \underbrace{|\mathcal{P}(\mathbb{N})| = 2^{\omega}}_{\substack{\text{Insieme delle funzioni,} \\ \text{non numerabile}}} = |\mathbb{R}|$$

Si ha quindi che **l'insieme delle funzioni non è numerabile**

Ci si chiede se queste funzioni sono tutte calcolabili:

Definizione 2.1. Una funzione **intuitivamente calcolabile** è una funzione descrivibile attraverso un algoritmo, cioè una sequenza finita di passi discreti elementari.

2.1 Quante funzioni numerabili ci sono?

Consideriamo Σ come un alfabeto finito, cioè una seguenza di simboli utilizzabili per scrivere un programma o algoritmo.

$$\Sigma = \{s_1, s_2, s_3, \ldots, s_n\}$$

$$\downarrow$$

Programma $\subseteq$ sqeuenze di simboli in Σ

Con Σ^* si descrive l'insieme di tutte le sequenze finite di simboli in Σ, quindi l'insieme di tutti i possibili programmi è:

$$\text{Programmi} \subseteq \Sigma^*$$

Esempio 2.2. Se $\Sigma = \{a, b, c\}$ allora la sequenza di tutti i possibili simboli è:

$$\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, \ldots\}$$

dove ϵ è la stringa vuota.
La cardinalità di Σ^* è infinita numerabile (anche se Σ è finito).

$$|\Sigma^*| = |\mathbb{N}|$$

Si ha quindi che l'insieme dei programmi è numerabile:

$$|\text{Programmi in } \Sigma| \leqslant |\Sigma^*| = |\mathbb{N}|$$

e questo implica che l'insieme delle **funzioni calcolabili è numerabile**

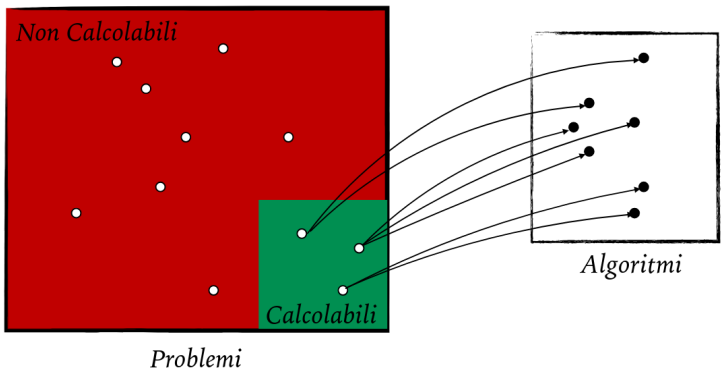


Figura 1: Cardinalità delle funzioni

Esempio 2.3. Prendiamo ad esempio la seguente funzione (serie di fibonacci):

$$f(n) \subseteq \mathbb{N} \times \mathbb{N}$$

Dove:

$$f(0) = 1, f(1) = 1, f(2) = 2$$
$$f(3) = 3, f(4) = 5, f(5) = 8$$
$$f(6) = 13, f(7) = 21, \ldots$$

Definiamo un algoritmo ricorsivo:

$$\begin{cases} f(0) = 1 = f(1) \\ f(x+2) = f(x+1) + f(x) \end{cases}$$

Trovare un algoritmo non è possibile per tutte le funzioni, ma solo per quelle calcolabili.

Nell'insieme delle funzioni calcolabili ci sono:

- Funzioni totali, cioè definite per ogni input $n \in \mathbb{N}$ e terminano sempre
- Funzioni parziali, cioè non definite per ogni $n \in \mathbb{N}$

2.2 Funzioni vs Insiemi

Una funzione può essere vista come un linguaggio $\mathcal{L}_f$ tale che:

$$f : \mathbb{N} \to \mathbb{N} \quad \leftrightarrow \quad \mathcal{L}_f = \left\{ 1^{f(x)} \mid x \in \mathbb{N} \right\} \quad \Sigma = \{1\}$$

Questo linguaggio permette di dire se un input appartiene o meno al linguaggio:

$$\sigma \in \Sigma^*$$

$$\downarrow$$

$$\begin{cases} \sigma \in \mathcal{L}_f & \text{se appartiene al linguaggio} \\ \sigma \notin \mathcal{L}_f & \text{se non appartiene al linguaggio} \end{cases}$$

Parliamo di insiemi invece che di funzioni dove gli elementi dell'insieme dipendono dal calcolo della funzione.

Esempio 2.4. Prendiamo ad esempio le seguenti funzioni:

- Funzione costante (Finite)

  $$f(x) = 2 \to \mathcal{L}_f \text{ è finito}$$

- Funzione lineare (Regolari)

  $$f(x) = 2x \to \mathcal{L}_f \text{ è infinito numerabile}$$

  C'è bisogno di una memoria finita per determinare se la stringa appartiene al linguaggio

- (Context free)

  $$f(\sigma) = \sigma\sigma^{\text{reverse}}$$

  $$\sigma = abc \quad \sigma^{\text{reverse}} = cba$$

  Per calcolare questa funzione c'è bisogno di una memoria illimitata, cioè non si può sapere a priori quanta ce n'è bisogno, è sufficiente uno stack.

- Decidibile

$$f(x) = x^2$$

Per calcolare questa funzione c'è bisogno di una memoria illimitata

Le funzioni calcolabili sono divise in classi secondo la gerarchia di Chomsky:

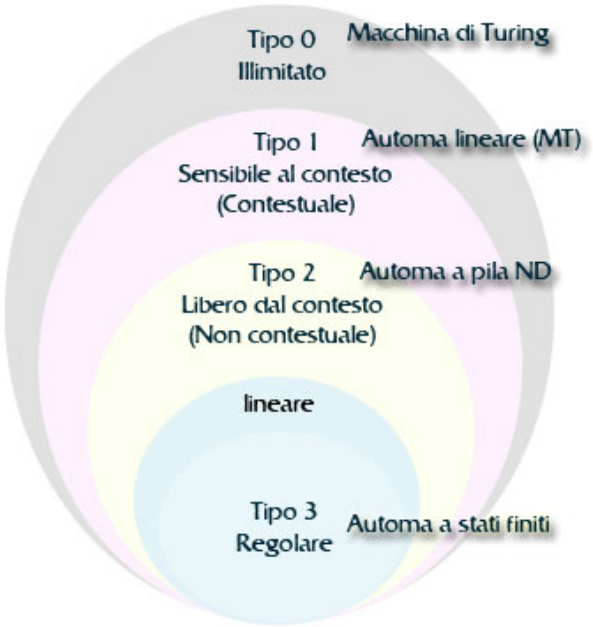


Figura 2: Gerarchia di Chomsky

3 Principio di induzione

Il principio di induzione è un meccanismo di definizione e dimostrazione che funziona **solo su insiemi infiniti**. Esistono due metodi di induzione:

- Induzione matematica
- Induzione strutturale

In questo corso tratteremo solo l'induzione matematica.

Un insieme A infinito con una relazione di ordine non riflessiva (senza l'uguale perchè l'elemento non è in relazione con sè stesso): $< : (A, <)$. $A = \mathbb{N}$ e $<$ è l'ordinamento stretto tra numeri naturali. La relazione di ordine deve essere **ben fondata**, quindi non devono esserci catene discendenti infinite, cioè una sequenza di elementi in ordine decrescente infinita:

$$a_0 > a_1 > a_2 > a_3 > \ldots \rightarrow \text{non ben fondata}$$

Una relazione di ordine riflessiva non è ben fondata, perchè esistono catene infinite:

$$a_0 \geqslant a_1 \geqslant a_2 \geqslant a_3 \geqslant a_3 \geqslant a_3 \geqslant \ldots \rightarrow \text{non ben fondata}$$

b minimale in $A : b \in A$ b è minimale se $\forall b' < b \;.\; b' \notin A$ Ad esempio: $\{1, 2, 3\}$ ha come minimali (di contenimento) $\{1, 2\}$ e $\{2, 3\}$.

Definizione 3.1 (Principio di induzione)**.** Se A è un insisme ben fondato (con ordinamento $<$), e Π è una proprietà definita sugli elementi di $A : \Pi \subseteq A$, allora:

$$\forall a \in A \;.\; \underbrace{\Pi(a)}_{\substack{a \text{ soddisfa } \Pi}} \iff \underbrace{\forall a \in A \;.\; [[\forall b < a \;.\; \Pi(b)] \Rightarrow \Pi(a)]}_{\substack{\text{Se dimostriamo } \Pi \text{ per ogni elemento più piccolo di } a, \\ \text{allora } \Pi \text{ vale anche per } a}}$$

Consideriamo come caso base gli elementi minimali di A:

$$\text{Base}_A = \{a \in A \mid a \text{ minimale}\}$$

Se si dimostra che Π vale per tutti gli elementi minimali di A (la base):

$$\overbrace{\underbrace{\forall a \in \text{Base}_A \;.\; \Pi(a)}_{\substack{\text{Dimostriamo } \Pi \text{ per ogni} \\ \text{elemento minimale di } A}}}^{\text{Base}} \wedge \underbrace{\forall a \in A \setminus \text{Base}_A}_{\text{Passo induttivo}} \;.\; \underbrace{\forall b < a \;.\; \Pi(b)}_{\text{Ipotesi induttiva}} \quad \underbrace{\Rightarrow \Pi(a)}_{\text{Tesi da dimostrare}}$$

Esempio 3.1. Dimostriamo che:

$$\forall n \in \mathbb{N} \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

L'insieme è:

$$A = \mathbb{N} \setminus \{0\} = \{1, 2, 3, \ldots\}$$

- **Base**

$$\text{Base}_A = \{1\}$$

  - dimostriamo la base

$$\sum_{i=1}^{1} i = n(n+1)/2 = 1(1+1)/2 = 1$$

- **Passo induttivo**: Prendo $n \in \mathbb{N}$

  - Ipotesi induttiva, cioè per ogni $m < n$ vale la proprietà:

$$\forall \underset{\in \mathbb{N}}{m} < n \;.\; \sum_{i=1}^{m} i = \frac{m(m+1)}{2}$$

Dobbiamo dimostrare la proprietà per n:

$$\sum_{i=1}^{n} i = \sum_{i=1}^{n-1} i + n$$

$n - 1 < n$ quindi vale l'ipotesi induttiva

$$\sum_{i=1}^{n-1} i = \frac{(n-1)(n-1+1)}{2} = \frac{(n-1)n}{2}$$

$$\downarrow$$

$$\begin{aligned}
\sum_{i=1}^{n} i &= \sum_{i=1}^{n-1} i + n \\
&= \frac{(n-1)n}{2} + n \\
&= \frac{(n-1)n + 2n}{2} \\
&= \frac{n^2 - n + 2n}{2} \\
&= \frac{n(n+1)}{2} \quad \square
\end{aligned}$$

È quindi dimostrato che:

$$\forall n \in \mathbb{N} \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

3.1 Linguaggi formali

Definizione 3.2. Un linguaggio formale è un insieme di stringhe costruite su un alfabeto finito Σ .

Solitamente un linguaggio formale $\mathcal{L}$ è un sottoinsieme di Σ^*, tipicamente infiniti, ma non necessariamente:

$$\mathcal{L} \subseteq \Sigma^*$$

I linguaggi sono divisi in:

- Linguaggi finiti
- Linguaggi regolari, il modello utilizzato è l'automa a stati finiti.

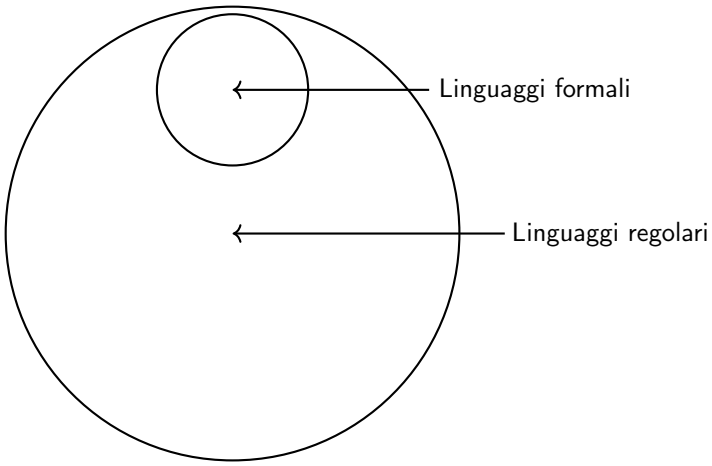


Figura 3: Linguaggi formali e linguaggi regolari

3.2 Linguaggi regolari (automi a stati finiti, DFA)

Il meccanismo più semplice per una memoria finita è l'automa a stati finiti

Consideriamo il linguaggio:

$$\mathcal{L}_f = \left\{1^{2n} \mid n \in \mathbb{N}\right\}$$

ha bisogno di due stati q_0 e q_1. Lo stato q_0 rappresenta l'informazione di essere di lunghezza pari, mentre lo stato q_1 rappresenta l'informazione di essere di lunghezza dispari.

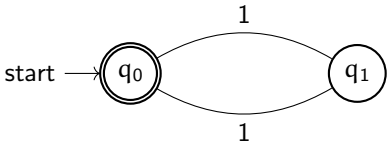


Figura 4: Automa a stati finiti per il linguaggio $\mathcal{L}_f = \left\{1^{2n} \mid n \in \mathbb{N}\right\}$

L'automa a stati finiti **deterministico** è definito come una quintupla:

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

dove:

- Q è un insieme **finito** di stati. Ogni stato rappresenta un'informazione
- Σ è un insieme **finito** di simboli (alfabeto). Ogni simbolo è un elemento atomico che posso leggere e che compone le stringhe da riconoscere
- $q_0 \in Q$ è uno stato e identifica lo stato iniziale. Lo stato finale viene indicato con un doppio cerchio
- $F \subseteq Q$ è l'insieme degli stati finali (di accettazione)

- $\delta : Q \times \Sigma \to Q$ È una **funzione di transizione** che dato uno stato e un simbolo, restituisce lo stato successivo ed è come se fosse una tabella che associa ad ogni coppia (stato, simbolo) uno stato:

$\Sigma \setminus Q$	q_0	q_1
1	q_1	q_0

  La funzione deve essere **totale**, cioè deve essere definita per ogni coppia $(q, a) \in Q \times \Sigma$, quindi la tabella deve essere completa.

- $\hat{\delta} : Q \times \Sigma^* \to Q$ Descrive lo stato che raggiungono leggendo una sequenza di simboli

$$\begin{cases} \hat{\delta}(q, \epsilon) = q \\ \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a) \end{cases} \qquad w \in \Sigma^*, \quad a \in \Sigma$$

  È quindi la **chiusura transitiva** di δ

3.2.1 Come si dimostra che un linguaggio è regolare?

Esempio 3.2. Prendiamo in considerazione il seguente linguaggio:

$$L = \{\sigma \mid \sigma \text{ contiene almeno due } 1\}$$

$$\Sigma = \{0, 1\}$$

Con le seguenti stringhe si ha:

- $\sigma = 011 \in \mathcal{L}$
- $\sigma = 1000100 \in \mathcal{L}$
- $\sigma = 00010 \notin \mathcal{L}$

L'informazione che codifica lo stato iniziale deve essere coerente con ε (stringa vuota)

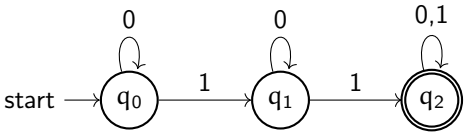


Figura 5: Automa a stati finiti per il linguaggio $\mathcal{L}$

Un lingauggio L è riconosciuto da $M = (Q, \Sigma, \delta, q_0, f)$ (DFA, Deterministic Finite Automaton) se: $L = L(M)$ dove $L(M)$ è il linguaggio di M definito come:

$$L(M) = \left\{\sigma \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, \sigma) \in F\right\}$$

Cioè sono tutte le stringhe che partendo da q_0 fanno raggiungere uno stato finale.

Definizione 3.3. Per dimostrare che L è regolare dobbiamo costruire M (almeno un M) e **dimostrare che** $L = L(M)$

$L = L(M)$ è un uguaglianza insiemistica e si dimostra con due contenimenti:

$$L = L(M) \equiv L \subset L(M) \wedge L(M) \subset L$$

- Se un elemento si trova nel primo insieme, allora si trova anche nel secondo

$$L \subseteq L(M) \equiv \sigma \in L \Rightarrow \sigma \in L(M) \equiv \sigma \in L \Rightarrow \hat{\delta}(q_0, \sigma) \in F$$

- Se un elemento si trova nel secondo insieme, allora si trova anche nel primo

$$L(M) \subseteq L \equiv \sigma \in L(M) \Rightarrow \sigma \in L \equiv \hat{\delta}(q_0, \sigma) \in F \Rightarrow \sigma \in L$$

  o per contrapposizione:

$$\sigma \notin L \Rightarrow \hat{\delta}(q_0, \sigma) \notin F$$

Questo dimostra che il linguaggio è regolare perchè è riconosciuto da un automa.

Esempio 3.3. Riprendendo l'esempio precedente:

$$L = \{\sigma \in \Sigma^* \mid \sigma \text{ contiene almeno due } 1\}$$

$$\Sigma = \{0, 1\}$$

$M =$

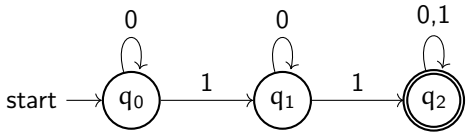


Figura 6: Automa a stati finiti per il linguaggio $\mathcal{L}$

Dimostriamo per induzione sulla lunghezza delle stringhe $\sigma \in \Sigma^*$ che se $x \in L$ allora $\hat{\delta}(q_0, x) \in F$ e se $x \notin L$ allora $\hat{\delta}(q_0, x) \notin F$.

$|\sigma| = 0$ non è **mai** sufficiente come base, ma è eventualmente la base **solo** per una delle due dimostrazioni. Bisogna quindi prendere la lunghezza più piccola che permette di avere sia $\sigma \in L$ che $\sigma \notin L$, in questo caso è $|\sigma| = 2$. Per ogni σ tale che $|\sigma| < 2 \quad \sigma \notin L$ perchè non può contenere due 1 e non è riconosciuta da M dove il primo stato finale è raggiunto leggendo almeno due simboli.

$$\varepsilon \in L \quad \varepsilon \notin L$$

- **Base**: Controlliamo ogni stringa di lungheezza minima nel linguaggio

per provare il caso base. In questo caso la lunghezza minima è $|\sigma| = 2$

$$\begin{cases} \sigma & = 11 \in L \text{ e } \hat{\delta}(q_0, 11) = q_2 \in F \\ \sigma & = 10 \notin L \text{ e } \hat{\delta}(q_0, 10) = q_1 \notin F \\ \sigma & = 01 \notin L \text{ e } \hat{\delta}(q_0, 01) = q_1 \notin F \\ \sigma & = 00 \notin L \text{ e } \hat{\delta}(q_0, 00) = q_0 \notin F \end{cases}$$

- **Passo induttivo**: Assumiamo che valga l'**ipotesi induttiva**, cioè la tesi con un limite fissato:

$$\forall \sigma \in \Sigma^* \ . \ |\sigma| \leqslant n \ . \ \begin{cases} \sigma \in L & \Rightarrow \hat{\delta}(q_0, \sigma) \in F \\ \sigma \notin L & \Rightarrow \hat{\delta}(q_0, \sigma) \notin F \end{cases}$$

Vogliamo dimostrare che la tesi vale per $|\sigma| = n+1$ (la successiva stringa che posso considerare).

Tesi: $\begin{cases} \sigma \in L \Rightarrow \hat{\delta}(q_0, \sigma) = q_2 \ \sigma \text{ contiene almeno due 1} \\ \sigma \notin L \Rightarrow \hat{\delta}(q_0, \sigma) = q_0 \ \sigma \text{ non contiene 1} \\ \sigma \notin L \Rightarrow \hat{\delta}(q_0, \sigma) = q_1 \ \sigma \text{ contiene esattamente un 1} \end{cases}$

Ipotesi induttiva:

$$\forall \sigma \in \Sigma^* \ . \ |\sigma| \leqslant n \ . \ \text{ allora la tesi vale su } \sigma$$

Dimostrazione della tesi per σ tale che $|\sigma| = n + 1$. ($|\sigma'| = n$ quindi su σ' possiamo applicare l'ipotesi induttiva):

$$|\sigma| = n + 1 \rightarrow \sigma = \sigma' 1 \ \vee \ \sigma = \sigma' 0$$

- Supponiamo che σ appartenga al linguaggio e termini con 1:

$$\sigma \in L \wedge \sigma = \sigma' 1$$

$$\downarrow$$

  - Se $\sigma' \in L$ applico l'ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_2$$

$$\begin{aligned} \hat{\delta}(q_0, \sigma) &\overset{\sigma = \sigma' 1}{=} \hat{\delta}(q_0, \sigma' 1) \\ &= \delta(\hat{\delta}(q_0, \sigma'), 1) \\ &= \delta(q_2, 1) = q_2 \in F \end{aligned}$$

  - Se $\sigma' \notin L$ allora σ' contiene esattamente un 1:

$$\hat{\delta}(q_0, \sigma') = q_1$$

$$\hat{\delta}(q_0, \sigma' 1) = \delta(q_1, 1) = q_2$$

– Supponiamo che σ appartenga al linguaggio e termini con 0:

$$\sigma \in L \wedge \sigma = \sigma'0$$

Per definizione di L abbiamo che

$$\sigma \in L \wedge \sigma = \sigma'0 \Rightarrow \sigma' \in L$$

Dimostriamo l'ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_2$$

allora

$$\begin{aligned}\hat{\delta}(q_0, \sigma) &= \hat{\delta}(q_0, \sigma'0) \\ &= \delta(\hat{\delta}(q_0, \sigma'), 0) \\ &= \delta(q_2, 0) = q_2 \in F\end{aligned}$$

– Supponiamo che σ non appartenga al linguaggio e contiene esatatmente un 1:

* $\sigma = \sigma'0 \Rightarrow \sigma' \notin L$ e contiene esattamente un 1
  Ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_1$$

$$\Downarrow$$

$$\begin{aligned}\hat{\delta}(q_0, \sigma) &= \hat{\delta}(q_0, \sigma'0) \\ &= \delta(\hat{\delta}(q_0, \sigma'), 0) \\ &= \delta(q_1, 0) = q_1 \notin F\end{aligned}$$

* $\sigma = \sigma'1 \Rightarrow \sigma' \notin L$ e non contiene 1
  Ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_0$$

$$\Downarrow$$

$$\begin{aligned}\hat{\delta}(q_0, \sigma) &= \hat{\delta}(q_0, \sigma'1) \\ &= \delta(\hat{\delta}(q_0, \sigma'), 1) \\ &= \delta(q_0, 1) = q_1 \notin F\end{aligned}$$

– Supponiamo che σ non appartenga al linguaggio e non contiene 1

$$\sigma \notin L \wedge \sigma = \sigma'0$$

$\sigma = \sigma'1$ non è possibile per l'ipotesi che σ non contiene 1
Ipotesi induttiva:

$$\hat{\delta}(q_0, \sigma') = q_0$$

$$\Downarrow$$

$$\begin{aligned}\hat{\delta}(q_0, \sigma) &= \hat{\delta}(q_0, \sigma'0) \\ &= \delta(\hat{\delta}(q_0, \sigma'), 0) \\ &= \delta(q_0, 0) = q_0 \notin F\end{aligned}$$

Tutti i casi sono dimostrati, quindi abbiamo dimostrato che:

$$L = L(M) \quad \Rightarrow \quad L \text{ è regolare}$$

Esercizio 3.1. Consideriamo il seguente linguaggio:

$$L = \{\sigma \in \Sigma^* \mid \text{ogni sequenza di 0 è di lunghezza pari}\}$$

$$\Sigma = \{0, 1\}$$

Si può accettare anche sequenze di lunghezza 0. Alcuni esempi sono:

$$\begin{aligned} 101 &\notin L \\ 1111 &\in L \\ 10010000 &\in L \\ 00101 &\notin L \end{aligned}$$

L'automa a stati finiti M è il seguente:

- q_0: Non sono stati letti 0
- q_1: Sequenza di 0 consecutiva di lunghezza dispari
- q_2: Sequenza di 0 consecutiva di lunghezza pari

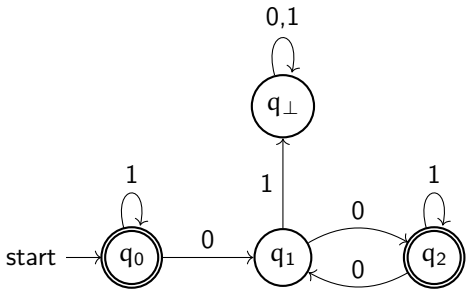


Figura 7: Automa a stati finiti per il linguaggio L

La tesi è:

- $\sigma \in L$ e non contiene 0: $\Rightarrow \hat{\delta}(q_0, \sigma) = q_0$
- $\sigma \in L$ e contiene una sequenza pari di 0: $\Rightarrow \hat{\delta}(q_0, \sigma) = q_2$
- $\sigma \notin L$ e contiene una sequenza **finale** dispari di 0: $\Rightarrow \hat{\delta}(q_0, \sigma) = q_1$
- $\sigma \notin L$ e contiene una sequenza dispari di 0 seguita da 1: $\Rightarrow \hat{\delta}(q_0, \sigma) = q_\perp$

Esercizio 3.2. Consideriamo il seguente linguaggio (ogni sequenza di 0 è di lunghezza almeno 2):

$$L = \{\sigma \in \Sigma^* \mid \exists n \geqslant 1 \,.\, \sigma = 0^n \Rightarrow n \geqslant 2\}$$

$$\Sigma = \{0, 1\}$$

L'automa a stati finiti M è il seguente:

- q_0: Non contiene 0, oppure **tutte** le sequenze di 0 sono lunghe almeno 2
- q_1: Esattamente uno 0
- q_2: Almeno due 0

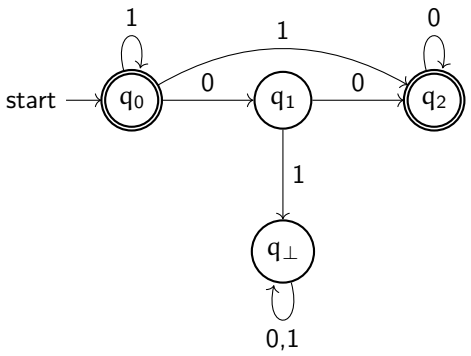


Figura 8: Automa a stati finiti per il linguaggio L

La tesi è:

- $\sigma \in L$ e $\sigma = \sigma' 1 \Rightarrow \hat{\delta}(q_0, \sigma) = q_0$
- $\sigma \in L$ e $\sigma = \sigma' 0 \Rightarrow \hat{\delta}(q_0, \sigma) = q_2$
- $\sigma \notin L$ e $\sigma = \sigma' 0$ dove l'ultima sequenza di 0 è esattamente lunga 1: $\Rightarrow \hat{\delta}(q_0, \sigma) = q_1$
- $\sigma \notin L$ e σ contiene una sequenza lunga 1 di 0: $\Rightarrow \hat{\delta}(q_0, \sigma) = q_\perp$

3.3 Automi a stati finiti non deterministici (NFA)

Un automa a stati finiti non deterministico si crea quando ad un solo simbolo sono associate più transizioni. Quando questo succede gli stati vengono considerati in parallelo. Un NFA è definito come una quintupla:

$$N = \langle Q, \Sigma, \delta, q_0, F \rangle$$

- Q è un insieme finito di stati
- Σ è un insieme finito di simboli (alfabeto)

- $q_0 \in Q$ è uno stato e identifica lo stato iniziale
- $F \subseteq Q$ è l'insieme degli stati finali
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ è una funzione di transizione che dato uno stato e un simbolo restituisce un insieme di stati **potenzialmente** raggiungibili. È possibile che esistano coppie associate all'insieme vuoto:

  $$\varnothing \in \mathcal{P}(Q)$$

  Inoltre non è obbligatorio avere un arco uscente per ogni simbolo di Σ.
- $\hat{\delta} : Q \times \Sigma^* \to \mathcal{P}(Q)$ Descrive gli stati che si possono raggiungere leggendo una sequenza di simboli:

  $$\begin{cases} \hat{\delta}(q, \epsilon) = \{q\} \\ \hat{\delta}(q, wa) = \bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a) \end{cases} \qquad w \in \Sigma^*, \quad a \in \Sigma$$

  È quindi la chiusura transitiva di δ

Esempio 3.4. Un esempio di NFA è il seguente:

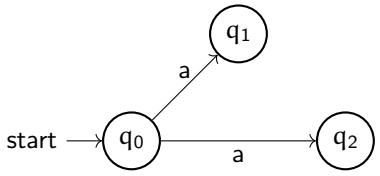


Figura 9: Esempio di NFA

$$\delta(q_0, a) = \{q_1, q_2\} \subseteq Q$$

3.3.1 Linguaggio riconosciuto da un NFA

Un linguaggio L è riconosciuto da un NFA N se:

$$L(N) = \left\{ \sigma \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, a) \cap F \neq \varnothing \right\}$$

Esempio 3.5. Consideriamo il seguente linguaggio:

$$L(N) = \{\sigma \in \Sigma^* \mid \sigma \text{ contiene almeno due } 1\}$$

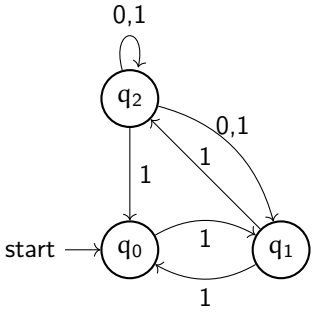


Figura 10: Esempio di NFA per il linguaggio L

Teorema 3.1 (Teorema di Rabin-Scott)**.** Ogni linguaggio riconosciuto da un NFA è riconosciuto da un DFA.

$$\forall N = (Q, \Sigma, \delta, q_0, F)\ \exists M = (Q', \Sigma, \delta', q_0', F')\ .\ L(N) = L(M)$$

Dimostrazione: Consideriamo un NFA $N = (Q, \Sigma, \delta, q_0, F)$ e costruiamo un DFA $M = (Q', \Sigma, \delta', q_0', F')$. Gli insiemi degli stati sono:

$$Q' = \mathcal{P}(Q)$$

$$Q = \{q_0, q_1, q_2\}$$

$$\Downarrow$$

$$Q' = \left\{ \overset{q_1'}{\varnothing}, \overset{q_0'}{\{q_0\}}, \overset{q_2'}{\{q_1\}}, \overset{q_3'}{\{q_2\}}, \overset{q_4'}{\{q_0, q_1\}}, \overset{q_5'}{\{q_0, q_2\}}, \overset{q_6'}{\{q_1, q_2\}}, \overset{q_7'}{\{q_0, q_1, q_2\}} \right\}$$

Lo stato iniziale rimane uguale per entrambi gli insiemi: $q_0' = \{q_0\}$.

Gli insiemi degli stati finali sono:

$$F' = \{P \subseteq Q \mid P \cap F \neq \varnothing\} \quad P \in \mathcal{P}(Q)$$

Quindi:

$$F = \{q_2\}$$

$$F' = \left\{ \overset{q_3'}{\{q_2\}}, \overset{q_6'}{\{q_1, q_2\}}, \overset{q_5'}{\{q_0, q_2\}}, \overset{q_7'}{\{q_0, q_1, q_2\}} \right\}$$

La funzione di transizione è definita come:

$$\delta'(P, a) = \bigcup_{q \in P} \delta(q, a) \in \mathcal{P}(Q) \quad P \in Q' = \mathcal{P}(Q),\ a \in \Sigma$$

Quindi:

$$\begin{aligned}\underbrace{\delta'(q_5', 1)}_{=\{q_1, q_2\}} &= \delta(q_1, 1) \cup \delta(q_2, 1) \\ &= \{q_0, q_2\} \cup \{q_0, q_1, q_2\} \\ &= \{q_0, q_1, q_2\} = q_7'\end{aligned}$$

Dimostriamo:

1. $\hat{\delta}(q_0, \sigma) = \hat{\delta}'(q_0', \sigma) = \{q_0\}$

   Dimostriamo per induzione su $|\sigma|$:

   - Se $\sigma = \varepsilon$ allora:

     $$\hat{\delta}'(q_0', \varepsilon) = q_0' = \{q_0\} = \hat{\delta}(q_0, \varepsilon)$$

     per le definizioni
   - Se $\sigma = \sigma' a$:

     $$\begin{aligned}\hat{\delta}'(q_0', \sigma' a) &= \delta'(\hat{\delta}'(q_0, \sigma'), a) \\ &= \delta'(\hat{\delta}(q_0, \sigma'), a) \\ &= \bigcup_{p \in \hat{\delta}(q_0, \sigma')} \delta(p, a) \\ &= \hat{\delta}(q_0, \sigma' a)\end{aligned}$$

     Definizione di $\hat{\delta}'$ non deterministica

2. $\sigma \in L(N) \iff \sigma \in L(M)$

   Dimostriamo la definizione di linguaggio riconosciuto in NFA:

   $$\begin{aligned}\sigma \in L(N) &\iff \hat{\delta}(q_0, \sigma) \cap F \neq \varnothing \\ &\iff \hat{\delta}'(q_0', \sigma) \cap F' \neq \varnothing \ ((1.)) \\ &\iff \hat{\delta}'(q_0', \sigma) \in F' \\ &\iff \sigma \in L(M) \text{ (def. linguaggio accettato in DFA)}\end{aligned}$$

3.3.2 Conversione da NFA a DFA

Prendiamo in considerazione il seguente NFA:

$$L(N) = \{\sigma \in \Sigma^* \mid \sigma \text{ contiene almeno due } 1\}$$

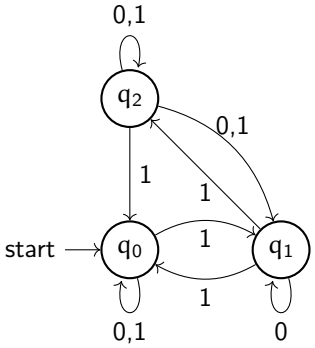


Figura 11: Esempio di NFA per il linguaggio L

Per trasformare un NFA in un DFA bisogna creare dei nuovi stati che raggruppano gli stati non deterministici. In questo caso:

- **Tabella degli stati della NFA**:

Stato	Input 0	Input 1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_1\}$	$\{q_0, q_2\}$
q_2	$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$

Tabella 1: Tabella di transizione della NFA

- **Traduzione degli stati della NFA in stati del DFA**:

		0	1
	$\varnothing$	$\varnothing$	$\varnothing$

Tabella 2: Tabella di traduzione degli stati della NFA in stati del DFA

3.4 Automi non deterministici con ε-transizioni (ε-NFA)

Questo tipo di NFA permette di cambiare stato anche senza leggere simboli:

$$q_1 \xrightarrow{\varepsilon} q_2$$

Un ε-NFA è definito come un NFA con la differenza che la funzione di transizione è definita come:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \underbrace{\mathcal{P}(Q)}_{\text{Non determinismo}}$$

Esempio 3.6. Prendiamo ad esempio il seguente ε-NFA in cui leggendo solo a si può raggiungere sia q' che q'':

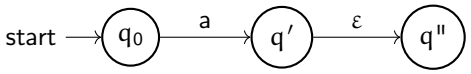


Figura 12: Esempio di ε-NFA

3.4.1 ε-closure

Per definire $\hat{\delta}$ bisogna prima definire la ε-closure.

Una ε-closure di uno stato q è l'insieme di tutti gli stati che si possono raggiungere da q seguendo archi etichettati con ε:

$$\varepsilon\text{-closure} : Q \to \mathcal{P}(Q)$$

O definito per insiemi:

$$\varepsilon\text{-closure} : \mathcal{P}(Q) \to \mathcal{P}(Q)$$
$$\varepsilon\text{-closure}(P) = \bigcup_{p \in P} \varepsilon\text{-closure}(p)$$

La funzione $\hat{\delta}$ è definita come:

$$\begin{cases} \hat{\delta}(q, \varepsilon) & = \varepsilon\text{-closure}(q) \\ \hat{\delta}(q, wa) & = \bigcup_{p \in \hat{\delta}(q,w)} \varepsilon\text{-closure}(\delta(p, a)) \end{cases}$$

Esempio 3.7. La ε-closure dell'esempio precedente è:

$$\varepsilon\text{-closure}(q') = \{q', q"\}$$

Il riconoscimento di un linguaggio è analogo a quello di un NFA:

$$L(N) = \left\{ \sigma \in \Sigma^* \;\middle|\; \hat{\delta}(q_0, \sigma) \cap F \neq \varnothing \right\}$$

Teorema 3.2. Sia $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ una ε-NFA, allora esiste una NFA M' tale che $L(M) = L(M')$.

Quindi l'insieme dei linguaggi riconosciuti da ε-NFA coincide con quello degli NFA, che a sua volta coincide con i linguaggi regolari.

$$M = \langle Q, \Sigma, \delta, q_0, F \rangle \;\; \varepsilon\text{-NFA}$$

Costruiamo una NFA

$$M' = \langle Q', \Sigma', \delta', q_0', F' \rangle$$

Dove:

$$Q' = Q, \quad \Sigma' = \Sigma, \quad q_0' = q_0$$
$$\delta'(q, a) = \hat{\delta}(q, a) \quad F' = \begin{cases} F \cup \{q_0\} \text{ se } \varepsilon\text{-closure}(q_0) \cap F \neq \varnothing \\ F \end{cases}$$

in cui il numero degli stati rimane lo stesso, l'alfabeto non cambia e neanche lo stato iniziale.

4 Espressioni regolari

Le espressioni regolari sono operazioni algebriche sui linguaggi regolari. Le operazioni che si possono fare sono:

- **Unione**: Dati i linguaggi $L_1, L_2 \subseteq \Sigma^*$

$$L_1 \cup L_2 = \{\sigma \mid \sigma \in L_1 \vee \sigma \in L_2\}$$

- **Concatenazione**: Dati i linguaggi $L_1, L_2 \subseteq \Sigma^*$

$$L_1 \cdot L_2 = \{\sigma_1\sigma_2 \mid \sigma_1 \in L_1 \wedge \sigma_2 \in L_2\} \equiv L_1L_2$$

  Ad esempio:

$$L_1 = \{0101, 010101\} \quad L_2 = \{000, 111\}$$

$$L_1 \cdot L_2 = \{0101000, 0101111, 010101000, 010101111\}$$

- **Stella di Kleene**: Dato un linguaggio $L \subseteq \Sigma^*$

$$L^* = \bigcup_{n\in\mathbb{N}} L^n$$

  Cioè la concatenazione di L con se stesso n volte, con:

$$\begin{cases} L^0 = \{\varepsilon\} \\ L^{n+1} = L \cdot L^n \end{cases}$$

  Ad esempio:

$$L = \{000, 111\}$$

$$L^0 = \{\varepsilon\}$$
$$L^1 = L \cdot L^0 = \{000, 111\}$$
$$L^2 = L \cdot L^1 = \{000000, 000111, 111000, 111111\}$$
$$L^3 = L \cdot L^2 = \left\{ \begin{array}{l} 000000000, 000000111, 000111000, 000111111, \\ 111000000, 111000111, 111111000, 111111111 \end{array} \right\}$$
$$\vdots$$
$$L^* = \{L^0, L^1, L^2, L^3, \ldots\}$$

  - L^* è l'insieme di tutte le possibili concatenazioni di stringhe appartenenti a L, compresa la stringa vuota ε
  - $L^+ = \bigcup_{n\geqslant 0} L^n$ è definito come L^* senza la stringa vuota ε:

$$L^+ = L \cdot L^*$$

Definizione 4.1. Definiamo per induzione le espressioni regolari su un alfabeto Σ:

- **Caso base**:
  - $\varnothing \subseteq \Sigma^*$ è un'espressione regolare che rappresenta il linguaggio vuoto
  - ε è un'espressione regolare che rappresenta il linguaggio:
    $$\{\varepsilon\} \subseteq \Sigma^*$$
  - $a \in \Sigma$ è un'espressione regolare che rappresenta il linguaggio:
    $$\{a\} \subseteq \Sigma^*$$
- **Passo induttivo**:

  Siamo r, s sono espressioni regolari che rappresentano il linguaggio
  $$R \subseteq \Sigma^* \land S \subseteq \Sigma^*$$
  - $r + s$ è un'espressione regolare che rappresenta il linguaggio $R \cup S$
  - $r \cdot s$ è un'espressione regolare che rappresenta il linguaggio $R \cdot S$
  - r^* è un'espressione regolare che rappresenta il linguaggio:
    $$R^*$$

Esempio 4.1. Prendiamo ad esempio l'espressione regolare:

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

che equivale a

$$\{1^n \mid n \in \mathbb{N}\} \cup \{0^n \mid n \in \mathbb{N}\} \cup \{(10)^n \mid n \in \mathbb{N}\}$$

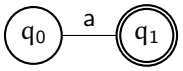
Teorema 4.1 (Teorema di equivalenza)**.** Dato un DFA $M = \langle Q, \Sigma, \delta, q_0, F\rangle$ allora esiste un'espressione regolare r tale che $L(M) = L(r)$.

$$L \text{ Regolare} \overset{\text{def}}{\iff} \underbrace{\exists M \text{ DFA} \,.\, L}_{L(M)=L} \overset{\text{Th}}{\Rightarrow} \exists r \in \underbrace{ER}_{\text{Espressioni Regolari}} \,.\, L(r) = L$$

Teorema 4.2. Data un'espressione regolare (ER) r esiste una ε-NFA M tale che: $L(r) = L(M)$

Quindi:

$\varepsilon \rightsquigarrow$  $L(M) = \{\varepsilon\}$

$\varnothing \rightsquigarrow$  $L(M) = \{\varnothing\}$

$$r \text{ ER} \Rightarrow M \quad \underbrace{\varepsilon\text{-NFA}}_{L(r)=L(M)} \iff$$
$$M' \quad \underbrace{\text{DFA}}_{L(M)=L(M')=L(r)} \iff$$
$$L(r) = L(M') \text{ è regolare}$$

Esempio 4.2. Consideriamo il seguente linguaggio:

$$L = \{\sigma \in \Sigma^* \mid \sigma \text{ contiene almeno due } 1\}$$

Per dimostrare che è regolare si costruisce il DFA M e si dimostra $L = L(M)$. Però se si ha un'espressione regolare $r = 0^*10^*10^*$ **non** dimostra che L è regolare. Bisognerebbe dimostrare $L = L(r)$

4.1 Proprietà dei linguaggi regolari

4.1.1 Proprietà di chiusura

Indica se l'insieme dei linguaggi regolari è chiuso rispetto ad alcune operazioni, cioè se applicando queste operazioni a linguaggi regolari si ottengono sempre linguaggi regolari.

Operazioni:

- $*$
- $\cup$
- $\cdot$
- $\cap \quad (L_1 \cap L_2 = \{\sigma \mid \sigma \in L_1 \wedge \sigma \in L_2\})$
- $\bar{} \quad (\bar{L} = \{\sigma \mid \sigma \text{notin} L\})$

Teorema 4.3. I linguaggi regolari sono chiusi rispetto alle operazioni di:

- Stella di Kleene
- Unione (finita)

- Concatenazione

Consideriamo i linguaggi regolari L_1, L_2. Allora:

- L_1^* è regolare
- $L_1 \cup L_2$ è regolare
- $L_1 \cdot L_2$ è regolare

Teorema 4.4. I linguaggi regolari sono chiusi rispetto alla complementazione. Dato un automa a stati finiti, il linguaggio complementare è riconosciuto dall'automa complementare, cioè quello in cui gli stati finali diventano non finali e viceversa.

Per le leggi di De Morgan, i linguaggi regolari sono chiusi per intersezione finita:

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

4.1.2 Proprietà di decidibilità

Indica se esistono algoritmi che risolvono alcuni problemi sui linguaggi regolari. Consideriamo un insieme di stringhe accettate da un DFA (linguaggiio regolare) M con n stati.

- $L(M) \neq \varnothing$ se e solo se accetta almeno una stringa di lunghezza $\leqslant n$
- $L(M)$ è infinito se e solo se accetta almeno una stringa di lunghezza l con $n \leqslant l < 2n$. Cioè se esiste un ciclo. Questo fornisce un estremo superiore per verificare se un linguaggio è infinito.

4.1.3 Proprietà dell'automa minimo

Fornisce strategie per costruire un automa minimo.

4.1.4 Condizione necessarie perchè un linguaggio sia regolare

Un linguaggio L è regolare se esiste un automa, quindi per dimostrare che un linguaggio non è regolare si può dimostrare che non esiste un automa (Pumping Lemma).

$$L \text{ regolare} \Rightarrow \Pi \quad \equiv \quad \neg\Pi \Rightarrow L \text{ non regolare}$$