Fondamenti di Informatica

Università di Verona
Imbriani Paolo - VR500437
Professor Isabella Mastroeni

October 13, 2025

Contents

1 Cosa è l'informatica?

La domanda chiave di questo corso è: *"Cosa è l'informatica?"*. Ci sono diversi definizioni a seconda del contesto, ma in generale l'informatica è lo studio dei processi che trasformano l'informazione. Possiamo vedere, storicamente, diverse definizioni come quella in inglese come "Computer Science" che vede l'informatica come studio della calcolabilità, della computazione e dell'informazione.

1.1 Perché la calcolabilità

Si studia la calcolabilità perché ci aiuta a capire cosa possiamo fare con gli strumenti che abbiamo. Quando descriviamo un programma, quanto tempo ci mette e quanto spazio utilizza è una domanda importante per capire se il programma è efficiente o meno. Anche in senso dei linguaggi di programmazione per capire se usiamo quello giusto per il problema che stiamo cercando di risolvere. Chiaramente è un qualcosa che in continuo sviluppo perché si evolve in base alla tecnologia che abbiamo a disposizione.

Uno dei pionieri è stato **Hilbert** che si chiedeva se la matematica fosse formalizzabile come insieme finito (non contraddittorio) di assiomi? Godel dimostrò che non è possibile rappresentare la matematica come un insieme finito di assiomi in maniera non contradditoria, dicendo che in ogni sistema formale ci sono proposizioni vere che non possono essere dimostrate all'interno del sistema.

Nel tentativo di rispondere a queste (ed altre) domande si è costruito un modello che permette di comprendere profondamente ilr agionamento computazionale, permettendo di applicarlo ad ogni disciplina. Cosa è calcolabile e cosa non lo è?

1.1.1 Macchina di Turing

Anche Turing si pose questa domanda e propose la **Macchina di Turing** come modello di calcolo. Una sola macchina (programmabile) per tutti i problemi. La macchina è universale (interprete):

$$Init(P,x) = \begin{cases} P(x) & \text{se } P \text{ è un programma che termina} \\ \uparrow & \text{se } P \text{ è un programma che non termina} \end{cases}$$

Se un problema è intuitivamente calcolabile, allora esisterà una macchina di Turing (o un dispostivo equivalente, come il computer) in grado di risolverlo, cioè calcolarlo. I modelli equivalenti possono essere:

- Lambda calcolo
- Funzioni ricorsive

- Linguaggi di programmazione (Turing completi)

I problemi non calcolabili sono infinitamente più numerosi di quelli calcolabili.

1.1.2 Limiti dell'informatica

L'informatica ha più limiti di quanto si possa pensare, definiti dall'equivalenza di Turing e l'incompletezza di Godel che ci dicono che non possiamo risolvere tutti i problemi. Ci sono anche limiti fisici e tecnologici come:

- Dati non osservabili (teorema di Shannon)
- Dati non controllabili (velocità della luce)

1.2 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 $(v, x, ...)$
  - 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 v variabile, alloar e $\forall v.\phi$ e $\exists v.\phi$ sono formule.

1.3 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 $x \bar{\in} A \leftrightarrow x \notin A$

1.4 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 $\forall y \in S.yRx$

1.5 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 a tutti gli effetti una proprietà che dato un oggetto stabilisce se esso all'interno di insieme o no.

$$\text{Problemi} \equiv \text{Funzioni} \quad f : \mathbb{N} \to \mathbb{N}$$

Ci chiediamo se questa funzioni siano tutte calcolabili (intuitivamente). Da il teorema che abbiamo citato nei precedenti paragrafi (quello dell'incompletezza) sappiamo che non è così. Cerchiamo di vedere insiemisticamente perché questo è giustificato.

Definizione 2.1: Intuitivamente Calcolabile

Qualcosa che è **intuitivamente calcolabile** è qualcosa è che riusciamo a descrivere attraverso un algoritmo, ovvero una sequenza finita di passi discreti elementari.

La funzione di tipo:

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

è un **insieme** di associazioni input-output.

Esempio 2.1

$$\begin{aligned} f = \text{quadrato} &= \{(0,0), (1,1), (2,4), (3,9), (4,16), ...\} \\ &= \{(x, x^2) \mid n \in \mathbb{N}\} \end{aligned}$$

Quindi f è un insieme di coppie in $\mathbb{N} \times \mathbb{N}$. Quindi $|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$ (cardinalità di $\mathbb{N} \times \mathbb{N}$ Quindi

$$f \subseteq \mathbb{P}(\mathbb{N} \times \mathbb{N}) = \mathbb{P}(\mathbb{N})$$

Per esempio se:

$$A = \{1, 2, 3\} \text{ allora } \mathbb{P}(A) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}\}$$

dove $|\mathbb{P}(A)| = 2^{|A|}$.

$$|\mathbb{N}| = \omega < |P(\mathbb{N})|) = |\mathbb{R}|$$

e quindi l'insieme delle funzioni **non è numerabile**.

2.1 Quale funzioni numerabili ci sono?

Σ =alfabeto finito di simboli che uso per il programma/algoritmo

$$\Sigma = s_1, s_2, s_3, \dots$$

quindi un programma non è nient'altro che un sottoinsieme finito di Σ^* (tutte le stringhe finite che posso formare con l'alfabeto).

$$\Sigma = a, b, c$$

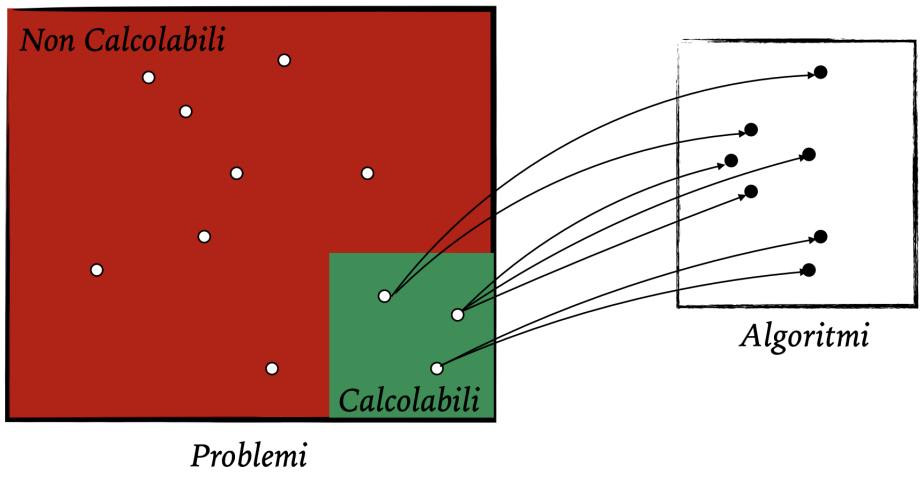
$$\Sigma^* = \{\emptyset, a, b, c, ab, ba, ac, cd, bc, cb, \dots\}$$

in questo caso, la sequenza di simboli in Σ^* è numerabile, perché

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

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

e di conseguenza l'insieme dei programmi è numerabile. Una veloce constatazione che possiamo fare è vedere quindi che l'insieme delle funzioni calcolabili è numerabile, perché ogni funzione calcolabile è associata ad almeno un programma che la calcola.



3 Principio di induzione

Il principio di induzione ha senso solo su insieme infiniti e serve per dimostrare che una proprietà vale per tutti gli elementi. Esistono due metodi di induzione:

- Induzione matematica
- Induzione strutturale

Tratteremo nello specifico caso in questo corso **l'induzione matematica.** Partiamo da un insieme A infinito con una relazione $< : (A, <)$ è una relazione d'ordine non riflessiva perché non è minore stretto. Utilizziamo l'induzione matematica e quindi $A = \mathbb{N}$ e $<$ è l'ordinamento stretto tra numeri. Una relazione d'ordine deve essere *ben fondata* vuol dire che non esistono catene discendenti infinite. Una catena discendente è una sequenza infinita di elementi

$$a_0 > a_1 > a_2 > a_3 > \dots$$

In un tipo di relazione riflessiva è sicuramente non ben fondata perché posso fare continuare ad inserire lo stesso numero all'infinito.

$$m \text{ minimale in } A \ : \ b \in A \text{ è minimale se } \forall b' < b . b' \notin A$$

Esempio 3.1

Se prendiamo $\{1, 2, 3\}$ con relazione d'ordine di contenimento allora esistono più minimali come $\{1, 2\}$ o $\{2, 3\}$. Quindi se $A = \mathbb{N} \implies \exists b$ minimo $\forall x \subseteq \mathbb{N}$.

Quindi preso A insieme ben fondato con ordinamento $<$ allora: π proprietà definita sugli elementi di

$$A : \pi \subseteq A \text{ allora } \forall a \in A . \pi(a) \iff \forall a \in A . [[\forall b < a . \pi(b)] \to \pi(a)]$$

Se dimostriamo π per ogni elemento più piccolo di a allora π vale per a.

$$Base_A = \{a \in A | a \text{ minimale}\}$$

Quindi

$$\overbrace{\forall A \in Base_A \, . \, \pi(a)}^{\text{Base}} \ \underbrace{\forall a \in A . Base_A}_{\text{passo induttivo}} . \ \overbrace{\forall b < a . \pi(b)}^{\text{ipotesi induttiva}} \ \to \underbrace{\pi(a)}_{\text{tesi}}$$

Esempio 3.2

Prendiamo come esempio il seguente enunciato:

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

Base: $n = 1$

$$A = \mathbb{N}\backslash\{0\} \;\; Base_A = \{1\} \rightarrow \sum_{i=1}^{1} i = 1$$

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

Caso base dimostrato.

Passo induttivo: prendo $n \in \mathbb{N}$ e applico l'ipotesi induttiva: per ogni $k < n$ vale la tesi.

$$\text{Tesi da dimostrare: } \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

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

Sappiamo sicuramente che $n - 1 < n$ e per ipotesi induttiva:

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

$$\frac{n(n-1) + 2n}{2} = \frac{n(n-1+2)}{2} = \frac{n(n+1)}{2} \quad \square$$

e quindi la tesi vale perché siamo arrivati alla stessa espressione che volevamo dimostrare.

3.1 Linguaggi formali

Definizione 3.1: Linguaggio formale

Un linguaggio formale è un insieme di stringhe composte da simboli in un alfabeto finito Σ.

Σ^* denota il linguaggio di tutte le stringhe dell'alfabeto Σ, se Σ non è vuota allora Σ^* è infinito e numerabile. Solitamente un linguaggio formale $\mathcal{L}$ è un sottoinsieme di Σ^* tipicamente infiniti ma non è necessario:

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

I linguaggi finiti sono sicuramente regolari perché *posso sempre costruire un automa a stati finiti* che li riconosce.

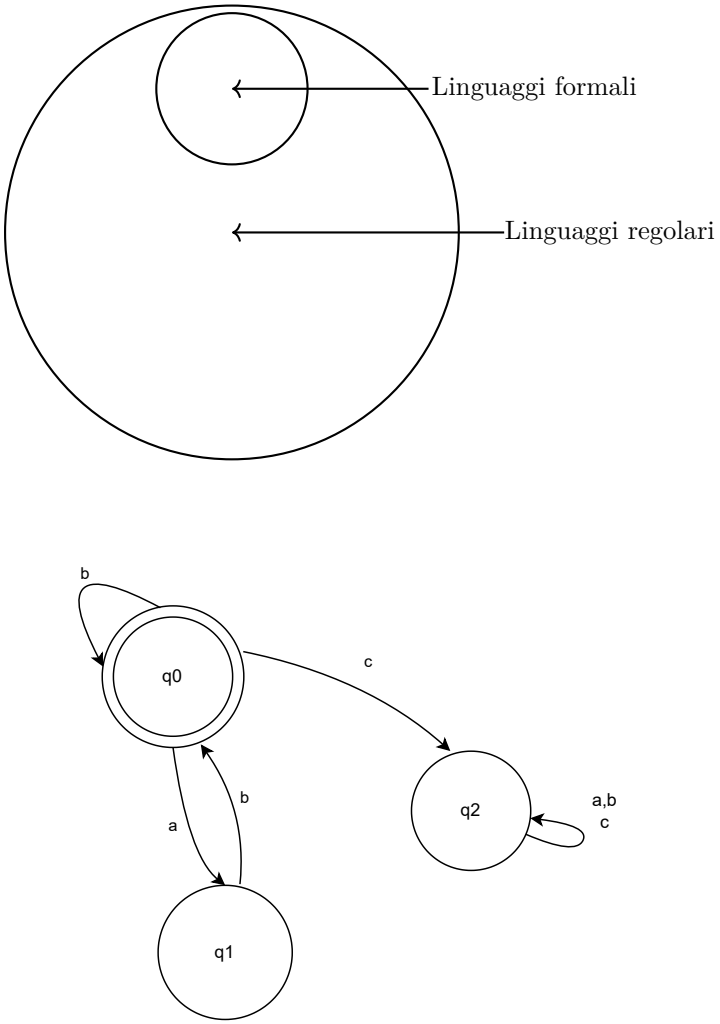


Figure 1: Esempio di automa a stati finiti

Come abbiamo detto in precedenza, una funzione è *calcolabile* se possiamo pensare ad un algoritmo per calcolarla. Tra le funzioni calcolabili, ci sono quelle totali, ovvero quelle che terminano per ogni input.

Esempio 3.3

Pensiamo ad una funzione $f \subseteq \mathbb{N} \times \mathbb{N}$, la possiamo descrivere come associazione di coppie:

$$f(0) = 1, f(1) = 1, f(2) = 2, f(3) = 3, f(4) = 5, f(5) = 8, \ldots$$

Tuttavia servirebbe scrivere una quantità infinita di coppie per descrivere la funzione.

Quindi proviamo a farlo ma stavolta ricorsivamente.

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

3.2 Funzioni e insiemi

In realtà ci sta un forte legame tra funzioni e insiemi perché per esempio dovessimo prendere una funzione $f : \mathbb{N} \to \mathbb{N}$, questa funzione può essere vista come un linguaggio $L_f = \{1^{f(x)} | x \in \mathbb{N}\}$ che è l'insieme delle stringhe composte da $f(x)$ simboli 1. Il nostro linguaggio $\Sigma = \{1\}$ e quindi il linguaggio è un sottoinsieme di Σ^*. Il nostro linguaggio ci può dire se un oggetto in input fa parte o no dell'insieme. Parliamo di **insiemi** invece che di funzioni dove gli elementi dell'insieme dipendono dal calcolo della funzione.

Esempio 3.4

Se io dovessi scrivere una funzione costante del tipo $f(x) = 2$. In questo caso riconosciamo che il linguaggio L_f è finito perché

$$L_f = \{1\}$$

L'unica combinazione possibile è una singola stringa. Il linguaggio si dice finito che è sottoinsieme di linguaggi totali.

Esempio 3.5

In questo caso la funzione $f(x) = 2x$ è una funzione lineare. Ho bisogno di una memoria finita. Mi sposto tra queste informazioni per determinare se la stringa in input fa parte o no del linguaggio. Questa funzione è anche detta **regolare**.

Esempio 3.6

$f(\sigma) = \sigma_{\text{reverse}}$ quindi per esempio:

$$f(abc) = cba$$

Tuttavia per questa funzione non mi basta più una memoria finita ma illimitata essendo che non posso sapere a priori quanta memoria abbia bisogno la funzione (è sufficiente uno stack). Questo tipo di funzione è detta anche **context free**.

Esempio 3.7

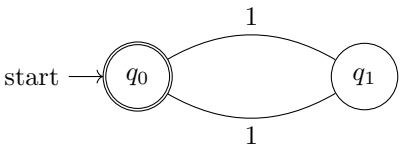
Nel caso di $f(x) = x^2$ sappiamo sicuramente che possiede un output preciso ma avrei bisogno di una memoria illimitata per rappresentarla.

4 Linguaggi Regolari - Automa a stati finiti

Quando parliamo di memoria è facilmente codificabile in termini di stati. Dal punto di vista grafico possiamo rappresentarli come nodi collegati da archi. Prendiamo per esempio il linguaggio L_f di prima:

$$L_f = \{1^{2n} \mid n \in \mathbb{N}\}$$

$$\Sigma = 1$$



Definizione 4.1: Automa a stati finiti

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

è un automa a stati finiti deterministico dove:

- Q è un insieme finito di stati (ogni stato rappresenta "un informazione")
- Σ è un alfabeto finito di simboli
- $\delta : Q \times \Sigma \to Q$ è la funzione di transizione (descrive come evolve il calcolo a partire dallo stato raggiunto e dal simbolo letto)
- $q_0 \in Q$ è lo stato iniziale
- $F \subseteq Q$ è l'insieme di stati finali (o di accettazione)

$\Sigma \backslash Q$	q_0	q_1
1	q_1	q_0

Ci sta poi la funzione $\hat{\delta} : Q \times \Sigma^* \to Q$ che estende la funzione di transizione e descrive lo stato

che raggiungiamo leggendo una sequenza di simboli.

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

anche chiamata chiusura transitiva di δ. Quindi mi permette di calcolare lo stato raggiunto leggendo una stringa di simboli.

4.1 Come si dimostra che un linguaggio è regolare?

Esempio 4.1

Prendiamo come esempio il linguaggio:

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

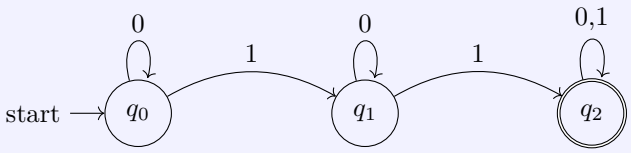
$$\Sigma = \{0, 1\}$$

Per esempio:

$$110, 101, 111, 011, 1001 \in L$$

$$0, 00, 000, 01, 10, 0000 \notin L$$

Quindi intuitivamente come possiamo definire l'automa?



Un linguaggio L è riconosciuto da M (Automa a stati finiti deterministico o DFA) se $L = L(M)$ dove $L(M)$ è il linguaggio di n definito come:

$$L(M) = \{\sigma \in \Sigma^* \mid \hat{\delta}(q_0, \sigma) \in F\}$$

Sono tutte le stringhe che partendo da q_0 (stato iniziale) raggiunge uno stato finale.

Definizione 4.2

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

$$L = L(M) \equiv L \subseteq L(M) \text{ e } L(M) \subseteq L$$

1.

$$L \subseteq L(M) \equiv \sigma. \in L \to \sigma \in L(M)$$

$$\sigma \in L \to \hat{\delta}(q_0, \sigma) \in F$$

2.

$$L(M) \subseteq L \equiv \sigma \in L(M) \to \sigma \in L$$

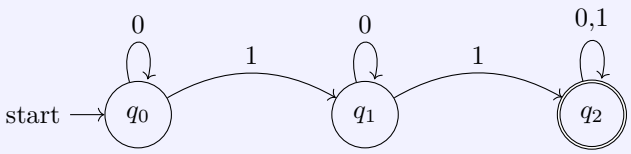
$$\hat{\delta}(q_0, \sigma) \in F \to \sigma \in L \equiv$$

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

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

Esempio 4.2

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



Dimostriamo per l'induzione:

- **Base:** $|\sigma| = 0$ non è **mai** sufficiente perché è vero solo per una delle due dimostrazioni $\Sigma \in L$ o $\Sigma \notin L$. Dobbiamo prendere la lunghezza più piccola che permette di avere sia $\sigma \in L$ che $\sigma \notin L$. Quindi la lunghezza più piccola è 2 (per ogni σ tale che $|\sigma| \leq 2, \sigma \notin L$ perché non può contenere due 1 e non è riconosciuta da M dove il primo stato finale è raggiunto se non vengono letti almeno due simboli). Se proviamo a prendere $\sigma = 11$ e $\hat{\delta}(q_0, 11) = q_2 \in F$ e quindi $\sigma \in L$. Se provassimo invece a prendere $\sigma = 00, 01, 10$ allora $\hat{\delta}(q_0, 11, 10, 01) = q_x \notin F$ e quindi $\sigma \notin L$. Quindi abbiamo controllato ogni stringa di lunghezza minima nel linguaggio per provare il caso base.
- **Passo induttivo:** L'ipotesi induttiva è la tesi con un limite fissato.

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

Cosa dobbiamo dimostrare?

$$\text{se } |\sigma| = n+1 \text{ allora } \begin{cases} \sigma \in L \to \hat{\delta}(q_0, \sigma) \in F \\ \sigma \notin L \to \hat{\delta}(q_0, \sigma) \notin F \end{cases}$$

Facciamo l'ipotesi che $\sigma = \sigma' a$ con $a \in \Sigma$ e $|\sigma'| = n$. Quindi sicuramente la stringa termina con un simbolo a che può essere 0 o 1.

$$\begin{cases} \sigma = \sigma' 0 \\ \sigma = \sigma' 1 \end{cases}$$

- Caso $\sigma = \sigma' 0 \wedge \sigma \in L$ per definizione σ' deve contenere almeno due 1 e quindi per ipotesi induttiva

$$\hat{\delta}(q_0, \sigma') \in F \to \hat{\delta}(q_0, \sigma' 0) = \hat{\delta}(\hat{\delta}(q_0, \sigma'), 0) \in F$$

- Caso $\sigma = \sigma' 1 \wedge \sigma \notin L$ se σ' contiene almeno un 1 allora σ ha un 1.
- Caso $\sigma = \sigma' 0 \wedge \sigma \notin L$ se σ' contiene almeno non ha 1 allora σ non ha 1.