

# Fondamenti dell'informatica

VR443470

febbraio 2023

# Indice

<b>1 Introduzione alla materia</b>	<b>3</b>
1.1 Cardinalità degli insiemi . . . . .	3
1.2 Alcune notazioni . . . . .	4
1.3 Teorema di Cantor (1874) . . . . .	5
1.4 Problema decisionale e ipotesi del continuo . . . . .	6
<b>2 Linguaggi regolari ed automi a stati finiti</b>	<b>7</b>
2.1 Alfabeti e Linguaggi . . . . .	7
2.2 Operazioni sui linguaggi . . . . .	8
2.3 Automa a stati finiti . . . . .	10
2.3.1 Automi deterministici (DFA) . . . . .	12
2.3.2 Esempio esercizio (automi deterministici) . . . . .	13
2.3.3 Automi non-deterministici (NFA) . . . . .	15
2.3.4 Teorema Rabin-Scott (1959) . . . . .	16
2.3.5 Esempio esercizio (automi non-deterministici) . . . . .	17
2.3.6 <b>Esercizi da esame</b> . . . . .	18
2.3.7 Automi con $\epsilon$ -transizioni ( $\epsilon$ -NFA) . . . . .	29
2.3.8 Teorema dell'equivalenza di $\epsilon$ -NFA e NFA . . . . .	30
<b>3 Espressioni regolari</b>	<b>32</b>
3.1 Espressioni regolari . . . . .	32
3.1.1 Teorema McNaughton & Yamamada (1960) - Equivalenza tra DFA e ER . . . . .	33
3.1.2 Proprietà di chiusura . . . . .	35
<b>4 Proprietà dei linguaggi regolari</b>	<b>36</b>
4.1 Teorema di Myhill-Nerode (1957-58) . . . . .	36
4.2 Pumping Lemma . . . . .	42
4.2.1 <b>Esercizi da esame</b> . . . . .	44
<b>5 Linguaggi liberi dal contesto</b>	<b>58</b>
5.1 Linguaggi liberi dal contesto . . . . .	58
<b>6 Funzioni parziali ricorsive di S.C. Kleene</b>	<b>63</b>
6.1 Funzioni primitive ricorsive . . . . .	63
6.2 Macchina di Turing . . . . .	65
6.3 Funzioni parziali ricorsive di S.C. Kleene . . . . .	67
6.4 Il teorema s-m-n . . . . .	68
6.5 Problemi insolubili . . . . .	68
<b>7 Insiemi ricorsivi e ricorsivamente enumerabili</b>	<b>69</b>
7.1 Insiemi ricorsivi e ricorsivamente enumerabili . . . . .	69
<b>8 Teoremi di Ricorsione e Teorema di Rice</b>	<b>71</b>
8.1 Primo teorema di ricorsione . . . . .	71
8.2 Secondo teorema di ricorsione . . . . .	71
8.3 Teorema di Rice . . . . .	71

# 1 Introduzione alla materia

Prima di iniziare con la presentazione di alcuni concetti fondamentali, si definisce l'**invariante induttiva**: pensando a qualsiasi linguaggio di programmazione, una generica condizione è sempre vera prima, durante e dopo un ciclo. Questo concetto ritornerà in futuro.

## 1.1 Cardinalità degli insiemi

Il motivo dell'interesse di un ripasso di un argomento trattato in passato è giustificato dal fatto che i dati manipolati in informatica sono (e)numerabili, ovvero è possibile metterli in corrispondenza biunivoca con i numeri naturali, essendo essi stessi rappresentati da numeri (binari).

Di seguito viene mostrato un richiamo ai concetti di base in relazione alla cardinalità di insiemi:

- ★ **Cardinalità.** Se  $S$  è un insieme, la sua *cardinalità* si rappresenta con il simbolo  $|S|$ .
- ★ **Equipotenza.** Due insiemi  $A$  e  $B$  sono *equipotenti* se esiste una funzione biiettiva del tipo  $f : A \rightarrow B$  (cioè una funzione sia iniettiva che suriettiva; approfondimento: [link](#), oppure qui di seguito).  
La *rappresentazione* matematica è la seguente  $A \approx B$ .  
La relazione  $|A| \leq |B|$  è possibile se esiste una funzione iniettiva  $f : A \rightarrow B$ . Si osservi che la funzione  $f$  stabilisce una corrispondenza tra gli elementi dei due insiemi. Infatti, l'**iniettività** assicura che la corrispondenza è stabilita elemento per elemento, mentre la **suriettività** assicura che la quantità degli oggetti nei due insiemi coincide.
- ★ **Insiemi finiti e infiniti.** Negli *insiemi finiti*, la cardinalità è un numero naturale corrispondente al numero di oggetti contenuti nell'insieme. Invece, negli *insiemi infiniti* la  $|A|$  rappresenta la collezione degli insiemi  $Y$  tale che  $Y \approx A$ . Questa collezione viene chiamata **cardinalità** di  $A$ . Quindi, è vero che se  $A \subseteq B$  allora si deduce che  $|A| \leq |B|$ .
- ★ **Insieme numerabile.** Un insieme  $A$  viene detto *numerabile* se è finito o equipotente all'insieme dei numeri naturali  $\mathbb{N}$  (ovvero,  $A \approx \mathbb{N}$ ).  
La cardinalità degli insiemi infiniti numerabili è denotata con  $\aleph_0$ .  
Un insieme  $A$  è finito se  $|A| < \aleph_0$ . Quindi, un insieme è numerabile se  $|A| \leq \aleph_0$  (ovvero se è finito, quindi minore, oppure se è un insieme infinito numerabile rappresentato come  $\aleph_0$ , quindi uguale).

## 1.2 Alcune notazioni

Se un generico *linguaggio di programmazione* viene indicato con la lettera  $\mathcal{L}$  e un generico *algoritmo* di un programma viene indicato con la lettera  $A$ , allora se un *algoritmo viene implementato in un linguaggio di programmazione*, è possibile scrivere la notazione insiemistica  $A \in \mathcal{L}$ . In un linguaggio di programmazione è possibile scrivere infiniti programmi, ovvero l'insieme dei numeri naturali  $\mathbb{N}$ .

Esistono due tipi di *rappresentazioni*:

- ☛ **Rappresentazione intensionale.** Rappresenta solo l'algoritmo, più nello specifico solamente quella specifica parte di codice (esempio a fine elenco).
- ☛ **Rappresentazione estensionale.** Rappresenta l'insieme ma tramite una forma più estesa (esempio a fine elenco).

L'*esecuzione* di un determinato algoritmo si indica con delle parentesi quadre più spesse  $\llbracket A \rrbracket$ . Quindi, la sua *rappresentazione intensionale* è solamente  $A$ , mentre la sua *rappresentazione estensionale* è data da  $\llbracket A \rrbracket(i) = o$  ( $i$  è input e  $o$  è output). La rappresentazione estensionale può essere anche nel seguente modo  $\llbracket M \rrbracket \in \{f \mid f = \llbracket M \rrbracket\}$  con  $f = \{(x, f(x)) \mid x \in \mathbb{N}\}$ .

Un programma restituisce uno o più risultati come numeri naturali  $\mathbb{N}$ , prendendo in input dei numeri naturali  $\mathbb{N}$ . Quindi, più formalmente si può scrivere  $\mathbb{N} \rightarrow \mathbb{N}$ . Questa rappresentazione non è altro che la definizione dei *problem* esistenti. Difatti, l'informatica si pone il dubbio che esista una certa soluzione ( $f$ ), scritta sotto forma di algoritmo appartenente ad un linguaggio di programmazione, tale che la sua esecuzione dia la soluzione. Più formalmente:

$$\mathbb{N} \rightarrow \mathbb{N} \ni f \quad \exists A \in \mathcal{L} : \llbracket A \rrbracket = f$$

### 1.3 Teorema di Cantor (1874)

Il seguente teorema ha come conseguenza che **esistono insiemi non numerabili**. Questo risultato si attribuisce a Georg Cantor, matematico tedesco, nel 1874.

La **dimostrazione** è importante da capire. Essa utilizza una tecnica, detta dimostrazione diagonale, che è alla base di gran parte dei risultati principali che stabiliscono i fondamenti dell'informatica come scienza (Dauben, 1979; [Official Cambridge article link](#)).

**Teorema 1 (Cantor).**

$$|\mathbb{N}| < |\mathbb{N} \rightarrow \mathbb{N}| \quad (1)$$

*La cardinalità di  $\mathbb{N}$  (numero di programmi per risolvere problemi) è strettamente più piccolo della cardinalità delle funzioni  $\mathbb{N} \rightarrow \mathbb{N}$  (numero di problemi esistenti).*

**Dimostrazione.** Si supponga per assurdo che  $|\mathbb{N}| = |\mathbb{N} \rightarrow \mathbb{N}|$ . Questo implica che esistono funzioni numerabili come per esempio  $f_0, f_1, f_2, \dots, f_x, \dots$

La genialità di Cantor si manifesta quando pensa ad una funzione  $g(x)$  così definita:

$$g(x) = f_x(x) + 1 \quad \text{con } g : \mathbb{N} \rightarrow \mathbb{N}$$

Con ovviamente  $x \in \mathbb{N}$ . La funzione  $g(x)$  prende un numero naturale e restituisce un numero naturale, quindi è correttamente identificabile come un problema (definizione di problema a pagina 4) e matematicamente formalizzabile come  $\mathbb{N} \rightarrow \mathbb{N}$ .

Dunque, prendendo qualsiasi funzione  $f$  numerata  $x$ -esima, essa sarà diversa dalla funzione  $g$  numerata  $x$ -esima poiché sempre aumentata di 1:

$$f_x(x) \neq g(x) \rightarrow f_x(x) \neq f_x(x) + 1$$

QED

Questo teorema purtroppo non è possibile applicarlo agli algoritmi informatici poiché se al posto della funzione  $f_x(x)$  venisse inserito un algoritmo e quest'ultimo non terminasse mai, dunque sostituibile con  $\infty$ , la somma +1 non verrebbe mai eseguita. Per esempio, quando un programma entra in un loop che non gli consente di eseguire le istruzioni successive.

## 1.4 Problema decisionale e ipotesi del continuo

Un **alfabeto** è una sequenza di simboli con cui è possibile scrivere gli algoritmi risolutivi. L'alfabeto utilizzato nelle realizzazioni tecnologiche è l'**alfabeto binario**  $\Sigma = \{0, 1\}$ .

Un **problema decisionale** è la versione associata ad un dato problema informatico  $f : \mathbb{N} \rightarrow \mathbb{N}$ , ovvero alla funzione:

$$d_f : \mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\} \quad (2)$$

Definita nel seguente modo:

$$d_f((x, y)) = \begin{cases} 1 & \text{se } y = f(x) \\ 0 & \text{altrimenti} \end{cases}$$

Un problema decisionale non è altro che una funzione con co-dominio  $\{0, 1\}$  che è in grado di decidere se una data coppia  $(x, y) \in \mathbb{N} \times \mathbb{N}$  appartiene ad  $f$ .

Essendo un problema decisionale una funzione associata ai problemi in informatica, allora esiste la relazione:

$$\mathbb{N} \rightarrow \{0, 1\} \subseteq \mathbb{N} \rightarrow \mathbb{N}$$

Dunque, sicuramente sarà vera la seguente condizione:

$$|\mathbb{N} \rightarrow \{0, 1\}| \leq |\mathbb{N} \rightarrow \mathbb{N}|$$

Ma sarà vera anche la seguente:

$$|\mathbb{N} \rightarrow \{0, 1\}| = |\mathbb{N} \rightarrow \mathbb{N}|$$

**Dimostrazione.** È chiaro che la seguente relazione è vera:

$$|\mathbb{N} \times \mathbb{N}| = |\mathbb{N}|$$

Allora, vale anche:

$$|\mathbb{N} \rightarrow \{0, 1\}| = |\mathbb{N} \times \mathbb{N} \rightarrow \{0, 1\}|$$

Si prenda qualsiasi funzione del tipo  $f : \mathbb{N} \rightarrow \mathbb{N}$ . A tale funzione, viene associato l'insieme:

$$S_f = \{(i, o) \mid f(i) = o\} \subseteq \mathbb{N} \times \mathbb{N}$$

In cui  $i$  indica l'input e  $o$  indica l'output. Viene scritta la sua relativa equazione caratteristica:

$$f_{S_f}(x, y) = \begin{cases} 1 & \text{se } (x, y) \in S_f \\ 0 & \text{altrimenti} \end{cases}$$

Allora si può affermare con certezza:

$$|\mathbb{N}| < |\mathbb{N} \rightarrow \mathbb{N}| = |\mathbb{N} \rightarrow \{0, 1\}| = |2^{\mathbb{N}}| = |\mathbb{R}| \quad (3)$$

L'ultima uguaglianza è possibile grazie all'**ipotesi del continuo**. QED

## 2 Linguaggi regolari ed automi a stati finiti

### 2.1 Alfabeti e Linguaggi

Qui di seguito si lasciano una serie di definizioni utili per il futuro:

- ☛ **Simbolo.** Entità primitiva astratta che non viene definita formalmente (come punto, linea, etc.).  
Per esempio, lettere e caratteri numerici sono simboli.
- ☛ **Alfabeto.** Rappresentato con la lettera greca Sigma  $\Sigma$ , è un **insieme finito di simboli**.
- ☛ **Stringa (o parola).** Sequenza finita di simboli giustapposti, ovvero messi uno affianco all'altro.  
Per esempio, se  $a, b, c$  sono simboli, allora  $abcb$  è una stringa.
- ☛ **Lunghezza di una stringa.** Viene denotata con  $|w|$ , in cui  $w$  è una stringa, e rappresenta il **numero di occorrenze di simboli che compongono una stringa**. Ad esempio,  $|abcb| = 5$ .  
Attenzione che la **stringa vuota** viene denotata con  $\varepsilon$  ed è la **stringa costituita da zero simboli**:  $|\varepsilon| = 0$ .
- ☛ **Concatenazione di stringhe.** Due stringhe  $v$  e  $w$  sono concatenate quando si rappresentano nel seguente modo  $vw$ . Si **ottiene facendo seguire alla prima stringa la seconda**.  
La concatenazione è un'operazione che ammette come identità la stringa vuota  $\varepsilon$ .
- ☛ **Linguaggio formale.** Detto anche linguaggio  $L$ , è un **insieme di stringhe di simboli da un alfabeto  $\Sigma$** . L'insieme vuoto  $\emptyset$  e l'insieme  $\{\varepsilon\}$  sono due linguaggi formali di qualunque alfabeto. L'insieme  $\emptyset$  **non contiene elementi**, mentre l'insieme  $\{\varepsilon\}$  **ne contiene uno**, ovvero la stringa vuota → sono insiemi diversi!
- ☛ **Sequenze finite.** Rappresentate con  $\Sigma^*$ , è il **linguaggio costituito da tutte le stringhe su un fissato alfabeto  $\Sigma$** . Quindi, viene considerato anche il **linguaggio più grande esistente**, ovvero il limite superiore. Tuttavia, questo non implica che sia il linguaggio più efficiente o potente (argomento approfondito in futuro). In parole povere, è l'insieme delle sequenze di stringhe finite.  
Definizione matematica:

$$\Sigma^* = \{a_1 \cdots a_n \mid n \geq 0, a_i \in \Sigma\}$$

Sia quindi  $L \subseteq \Sigma^*$  un **generico linguaggio formale** sull'alfabeto  $\Sigma$ .

## 2.2 Operazioni sui linguaggi

Sia  $\Sigma$  un alfabeto e  $L, L_1, L_2$  insiemi di stringhe di  $\Sigma^*$ . Le **operazioni sui linguaggi** sono principalmente tre:

- ☞ **Concatenazione.** La **concatenazione** di  $L_1$  e  $L_2$ , denotata con  $L_1 \cdot L_2$  è l'insieme:

$$L_1 L_2 = \{xy \in \Sigma^* \mid x \in L_1, y \in L_2\}$$

Si definisce dunque:

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

E facendo attenzione al fatto  $L^0 = \{\varepsilon\} \neq \emptyset$ .

- ☞ **Complemento.** Il **complemento** di un linguaggio è denotato con  $\bar{L}$ :

$$\bar{L} = \{\sigma \mid \sigma \in \Sigma^*, \sigma \notin L\}$$

- ☞ **Chiusura di Kleene.** Viene denotata con  $L^*$  ed è l'insieme così definito:

$$L^* = \bigcup_{n \geq 0} L^n$$

- ☞ **Chiusura positiva.** Denotata con  $L^+$  è l'insieme così definito:

$$L^+ = \bigcup_{n \geq 1} L^n$$

E si verifica immediatamente che  $L^+ = LL^*$ . Quindi, questo operato si può derivare dalla chiusura e dalla concatenazione.

- ☞ **Unione.** L'**unione** tra due linguaggi, che corrisponde ad un ***or logico***, è così definita:

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

- ☞ **Intersezione.** L'**intersezione** tra due linguaggi, che corrisponde ad un ***and logico***, è così definita:

$$L_1 \cap L_2 = \{\sigma \mid \sigma \in L_1 \wedge \sigma \in L_2\}$$

## Esempio di esercizio su linguaggi e alfabeti

Dati i seguenti dati:

**Alfabeto:**  $\Sigma = \{a, b\}$

**Linguaggio:**  $L = \{a, b\}$

Si costruisce l'insieme  $L^*$ , ovvero il linguaggio costituito da tutte le stringhe su un fissato alfabeto. Per definizione uguale anche a  $\Sigma^*$ :

$$L^* = \{\varepsilon, a, b, aa, ab, ba, bb, \dots\} = \Sigma^*$$

In cui la prima lettera, ovvero  $\varepsilon$ , può essere rappresentato con  $L^0$ ;

Le lettere  $\{a\}$  e  $\{b\}$ , insieme a  $L^0$ , possono essere rappresentate con  $L^1$ ;

Le lettere  $\{aa\}$ ,  $\{ab\}$ ,  $\{ba\}$ ,  $\{bb\}$ , insieme a  $L^0$  e  $L^1$ , possono essere rappresentate con  $L^2$ ;

E così via. Date che sono infinite stringhe componibili, il linguaggio viene associato a  $\Sigma^*$ .

### 2.3 Automa a stati finiti

Un **automa a stati finiti** è un modello matematico di un sistema avente un input ed eventualmente un output, a valori discreti. Il sistema può essere in uno stato tra un insieme finito di stati possibili. Essendo in uno stato, l'automa ha la possibilità di **tenere traccia della storia precedente**.

Analizzando letteralmente le parole di “automa a stati finiti”:

- ☞ **Automa.** Macchine che lavorano indipendentemente dall'intervento dell'essere umano.
- ☞ **A stati finiti.** Con **stato** si intende lo stato effettivo della macchina. Mentre con **finiti** si intende che lo stato al tempo  $t$  è effettivamente finito, ovvero è una quantità di informazione finita.

Soltamente, viene rappresentato con una testina che legge da un nastro, quest'ultimo contenente la sequenza di simboli dell'alfabeto dati in input all'automa. La testina che legge si sposta sempre nella stessa direzione, consumando la sequenza in input. La testina si può trovare in un certo **stato**; a seconda dello stato  $q$  e del simbolo  $s_i$  letto, la testina si porta in un certo altro stato (o rimane nello stesso) e si sposta a destra per apprestarsi a leggere il simbolo successivo dalla sequenza.

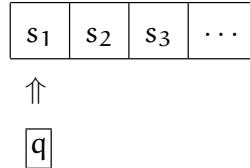


Figura 1: Dispositivo che rappresenta un esempio generalistico di un automa a stati finiti.

Una volta terminata la lettura, l'automa è in grado di fornire il risultato di accettazione o di refutazione della stringa (parola) letta. Il **comportamento** dell'automa si **definisce** in maniera univoca mediante una tabella, chiamata **matrice di transizione**, come ad esempio:

	<i>a</i>	<i>b</i>
<i>q</i> <sub>0</sub>	<i>q</i> <sub>1</sub>	<i>q</i> <sub>2</sub>
<i>q</i> <sub>1</sub>	<i>q</i> <sub>1</sub>	<i>q</i> <sub>0</sub>
<i>q</i> <sub>2</sub>	<i>q</i> <sub>1</sub>	<i>q</i> <sub>0</sub>

Tabella 1: Matrice di transizione.

Tuttavia, la rappresentazione più comune e chiara è il **grafo**:

- Gli **archi** rappresentano le *transizioni* etichettate con il simbolo in lettura;
- Lo **stato iniziale** è rappresentato con la *freccia “start”*;
- Gli **stati finali** sono cerchiati due volte.

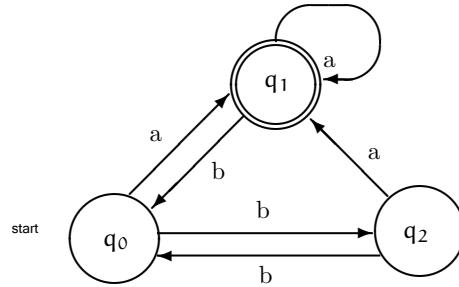


Figura 2: Esempio di grafo (freccia start mancante).

Un **linguaggio vuoto** è il più piccolo dei linguaggi, si rappresenta con il simbolo  $\emptyset$  e si riconosce poiché **non ha stati finiti** (quindi nessun nodo con il doppio cerchio). Infine, il **linguaggio più grande** è il linguaggio più grande, si rappresenta con il simbolo  $\Sigma^*$ , esiste dunque la relazione  $\emptyset \subseteq \Sigma^*$  e si riconosce perché **ha solo stati finiti** (quindi nessun nodo con un solo cerchio).

### 2.3.1 Automi deterministici (DFA)

Un **automa a stati finiti deterministico** (DFA) è una che, date determinate condizioni, è possibile determinare la serie di operazioni. Viene rappresentato con una quintupla del tipo  $\langle Q, \Sigma, \delta, q_0, F \rangle$  in cui:

- $Q$  è un insieme di **stati finiti**;
- $\Sigma$  è un **alfabeto finito**, cioè un alfabeto di input;
- $\delta : Q \times \Sigma \rightarrow Q$  è la **funzione di transizione** che dato lo stato  $q \in Q$  in cui si trova la macchina ed un simbolo  $a \in \Sigma$  in lettura del nastro, produce il prossimo stato  $\delta(q, a) \in Q$  in cui si troverà la macchina;
- $q_0$  è lo **stato iniziale**;
- $F \subseteq Q$  è l'insieme degli **stati finali**, anche detti **di accettazione**.
- $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$  si ottiene dalla funzione  $\delta$  ed è definita nel seguente modo:

$$\begin{cases} \hat{\delta}(q, \varepsilon) = q \\ \hat{\delta}(q, wa) = \delta\left(\hat{\delta}(q, w), a\right) \end{cases}$$

Sia  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un automa a stati finiti deterministico. Una **stringa**  $x$  è detta **accettata** da un  $M$  se  $\hat{\delta}(q_0, x) \in F$ . Inoltre, il **linguaggio** è **accettato dalla macchina**  $M$ , denotato come  $L(M)$ , è l'insieme di tutte le stringhe accettate da  $M$ , ovvero:

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

**N.B. A lezione al posto di  $w$  e  $x$  è stato utilizzato  $\sigma$ .**

Un linguaggio  $L$  viene detto **linguaggio regolare** se è accettato da qualche automa a stati finiti deterministico (DFA), ovvero se esiste  $M$  tale che  $L = L(M)$ .

### 2.3.2 Esempio esercizio (automati deterministici)

**Esercizio.**

Il grafo dell'esercizio è il seguente:

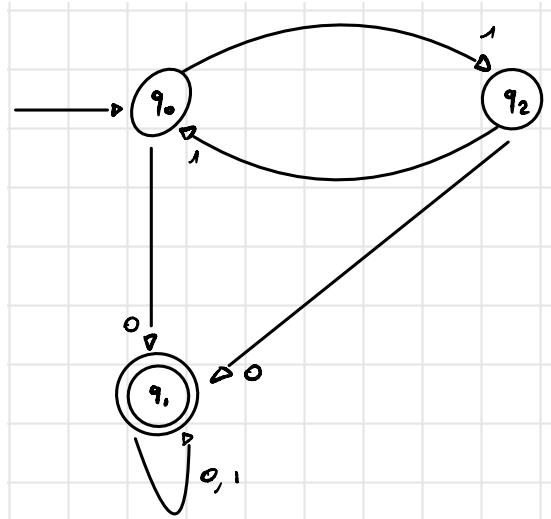


Figura 3: Grafo di un automa a stati finiti deterministico.

I dati forniti sono i seguenti:

**Linguaggio:**  $L(M) = \{x \in \{0, 1\}^* \mid x \text{ contiene almeno uno } 0\}$

**Alfabeto:**  $\Sigma = \{0, 1\}$

Si dimostri che  $L(M)$  è  $\subseteq$  e  $\supseteq$  del linguaggio che è stato dato. Quindi, è necessario verificare le condizioni con:

$$\{x \in \{0, 1\}^* \mid x \text{ contiene almeno uno } 0\}$$

**Risoluzione.**

La stringa  $x$  è sicuramente formata da:

$$x = 1^n 0 w \in \{0, 1\}^* \text{ e } n \in \mathbb{N}$$

In cui ci deve essere un numero  $n$  di concatenazioni di numeri 1, uno 0 come viene specificato “ $x$  contiene almeno uno 0” e infine una qualsiasi stringa dell’alfabeto  $w$ . L’**obiettivo** è dimostrare che:

$$\forall w \in \{0, 1\}^* : \delta(q_1, w) = q_1$$

Ovvero che dato qualsiasi simbolo appartenente all’alfabeto, lo stato successivo sia sempre  $q_1$ , ovvero quello finale!

Si supponga che la cardinalità di  $w$ , ovvero la lunghezza della stringa, sia uguale a  $m$ , cioè  $|w| = m$ . Questo implica che lo stato non varia:

$$m = 0 \implies w = \varepsilon \quad \text{e quindi} \quad \hat{\delta}(q_1, w) = q_1$$

Invece, il caso in cui la stringa non sia vuota:

$$m \geq 0 \text{ e } m + 1 \implies w = \sigma 0 \quad \text{oppure} \quad w = \sigma 1$$

Per **induzione** si riesce a dimostrare che:

$$\hat{\delta}(q_1, \sigma 0) = \delta\left(\hat{\delta}(q_1, \sigma), 0\right) = \delta(q_1, 0) = q_1$$

Grazie all'induzione è stato dimostrato che se era vero per  $m$ , allora è vero anche per  $m + 1$ , quindi qualsiasi  $m$  (maggiore o uguale a zero ovviamente). Quindi, il **primo passo dell'esercizio è stato concluso, ovvero quello di dimostrare che la  $w$  è sempre accettata soltanto se posta dopo il primo zero** (sequenza conclusiva dell'automa a stati finiti deterministico).

Adesso si prosegue la dimostrazione dimostrando che per ogni numero naturale, con una sequenza di 1 si finisce nello stato  $q_0$  o  $q_2$ , ovvero l'automa non termina.

Caso base:

- $n = 0 : \hat{\delta}(q_0, \varepsilon) = q_0 \in \{q_0, q_2\}$
- $n \geq 0 : \hat{\delta}(q_0, 1^{n+1}) = \hat{\delta}(q_0, 1^n 1) = \delta\left(\hat{\delta}(q_0, 1^n)\right) \in \{q_0, q_2\}$

✓**Dimostrato** che ogni stringa in forma  $x \in L$  (ogni stringa nel linguaggio), è accettata dall'automa. Ovvero  $L \subseteq L(M)$ .

L'esercizio si conclude con la dimostrazione  $L \supseteq L(M)$ - Quindi se  $x$  non contiene almeno uno zero ( $x = 1^n$ ), allora si può affermare con certezza che  $x$  non è in  $L(M) = \left\{ \sigma \mid \hat{\delta}(q_0, \sigma) = q_1 \right\}$ . Quindi formalmente:

$$\forall n : 1^n \implies \hat{\delta}(q_0, x) \in \{q_0, q_2\} \quad \checkmark \text{Dimostrato}$$

### 2.3.3 Automi non-deterministici (NFA)

Un **automa a stati finiti non-deterministico** (NFA) è una quintupla  $\langle Q, \Sigma, \delta, q_0, F \rangle$  dove  $Q, \Sigma, q_0$  e  $F \subseteq Q$  mantengono il significato visto per gli automi deterministici (pagina 12), mentre la **funzione di transizione**  $\delta$  è definita come:

$$\delta : Q \times \Sigma \longrightarrow P(Q) = 2^{|Q|}$$

Ovvero è una relazione tra stati.

In particolare, si tratta di un concetto fondamentale in informatica che definisce un **modello (ideale)** di calcolo parallelo su cui si fonda la moderna analisi della complessità degli algoritmi.

Adesso è possibile avere  $\delta(q, a) = \emptyset$  per qualche  $q \in Q$  ed  $a \in \Sigma$ , poiché l'**automa non può avere transizioni per alcuni simboli in input**.

Si definisce la funzione  $\hat{\delta} : Q \times \Sigma^* \longrightarrow P(Q)$  nel seguente modo:

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

Inoltre, si dice che **una stringa  $x$  è accettata da un automa a stati finiti non-deterministico NFA  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$**  se:

$$\hat{\delta}(q_0, x) \cap F \neq \emptyset$$

In altre parole, una stringa è accettata quando una di queste computazioni raggiunge uno stato finale dopo aver consumato la sequenza in input.

Invece, si dice che **un linguaggio è accettato da un automa a stati finiti non-deterministico NFA  $M$**  se corrisponde all'insieme delle stringhe accettate:

$$L(M) = \left\{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset \right\}$$

La rappresentazione a grafo rimane pressoché immutata. L'**unica differenza** è che da un nodo possono uscire più archi (o nessuno) etichettati dallo stesso simbolo.

### 2.3.4 Teorema Rabin-Scott (1959)

**Teorema 2 (Rabin-Scott).** Sia  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  un automa a stati finiti non deterministico (NFA). Allora esiste un automa a stati finiti deterministico  $M'$  tale che  $L(M) = L(M')$ .

**Dimostrazione.** Si definisce l'automa a stati finiti non deterministico con la quintupla  $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$  e con le seguenti proprietà:

$$\star \Sigma' = \Sigma.$$

$\star Q' = P(Q)$ , sarebbe più preciso definire  $Q' = \{q_1, \dots, q_{2|Q|}\}$  e poi stabilire una corrispondenza biunivoca fra tali stati e gli elementi di  $P(Q)$ . Tuttavia, così facendo rimane più chiara la dimostrazione.

$$\star q'_0 = \{q_0\}.$$

$$\star F' = \{P \subseteq Q \mid P \cap F \neq \emptyset\}$$

$$\star \delta'(P, a) = \bigcup_{p \in P} \delta(p, a), \text{ per ogni } P \in P(Q)$$

Si mostra, **per induzione**, sulla lunghezza della stringa di input  $\sigma \in \Sigma^*$  che:

$$\forall \sigma \in \Sigma^* : \hat{\delta}(q_0, x) = \hat{\delta}'(q'_0, x) \quad (4)$$

In cui la parte di sinistra è per gli automi non deterministici ( $\hat{\delta}(q_0, x)$ ), mentre la parte di destra è per gli automi deterministici ( $\hat{\delta}'(q'_0, x)$ ).

**Caso base.**<sup>1</sup>

$$|\sigma| = 0 \iff \sigma = \varepsilon : \hat{\delta}(q_0, \varepsilon) = \{q_0\} = q'_0 = \hat{\delta}'(q'_0, \varepsilon)$$

**Passo induttivo.**

$$\forall \sigma \in \Sigma^* : |\sigma| \leq n : \hat{\delta}(q_0, \sigma) = \hat{\delta}(q_0, \sigma)$$

Quindi che non ci siano gli apici ' come nell'equazione 4. Si dimostra induttivamente:

$$\hat{\delta}'(q'_0, \sigma a) = \delta'\left(\hat{\delta}'(q'_0, a), a\right) = \delta'\left(\hat{\delta}(q_0, \sigma), a\right) = \bigcup_{p \in \hat{\delta}(q_0, \sigma)} \delta(p, a) = \hat{\delta}(q_0, \sigma a)$$

In cui  $p \in \hat{\delta}(q_0, \sigma)$  e  $\hat{\delta}(q_0, \sigma a)$  sono **macchine deterministiche**.

La dimostrazione si conclude con le seguenti ovvie relazioni:

$$\sigma \in L \iff \hat{\delta}(q_0, \sigma) \cap F \neq \emptyset \iff \hat{\delta}'(q'_0, \sigma) \cap F \neq \emptyset \iff \hat{\delta}'(q'_0, \sigma) \in F' \iff \sigma \in L(M')$$

QED

---

<sup>1</sup>Il simbolo  $\iff$  indica "se e solo se".

### 2.3.5 Esempio esercizio (automati non-deterministici)

#### Esercizio.

Il grafo dell'esercizio è il seguente:

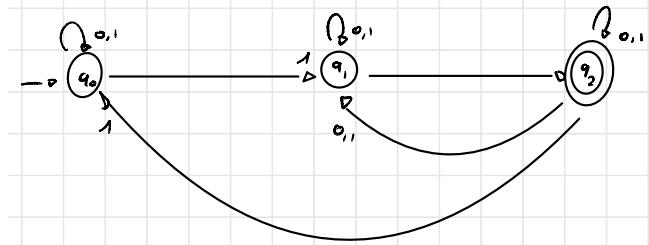


Figura 4: Grafo di un automa a stati finiti non-deterministico.

#### Risoluzione.

La  $Q$  è formata da  $\{q_0, q_1, q_2\}$ , mentre la  $P(a)$  ha i seguenti insiemi:

$$P(a) = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$$

Le espressioni in rosso sono sequenze finite che terminano.

Infine, l'automa non-deterministico è riconducibile a un automa deterministico:

$$\delta'(s, a) = \bigcup_{q \in S} \delta(q, a)$$

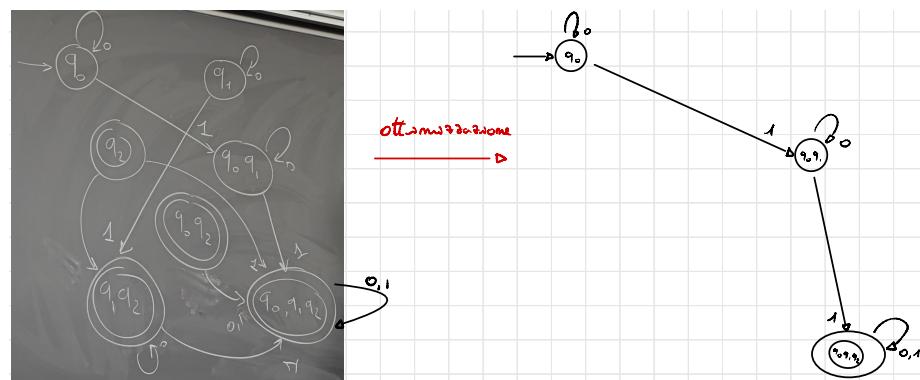


Figura 5: Rappresentazione di una conversione da non-deterministico a deterministico con relativa ottimizzazione.

### 2.3.6 Esercizi da esame

#### Esercizio 1

I dati dell'esercizio sono i seguenti:

$$L = \{x \mid |x_0| = 2\mathbb{N}\}$$

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

*Risoluzione.*

L'automa da costruire si basa sulla condizione del linguaggio, ovvero che  $x_0$  (**numero degli zeri di**  $x$ ) sia uguale a  $2\mathbb{N}$  (**numero pari**).<sup>2</sup>

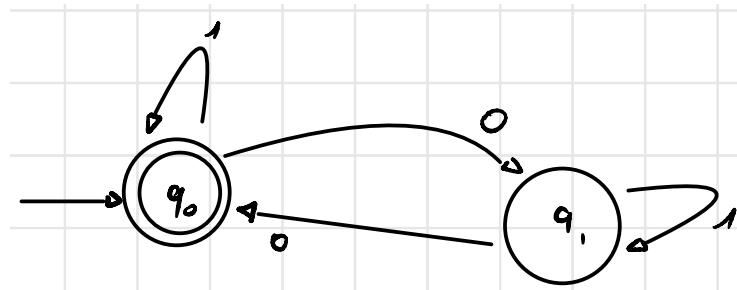


Figura 6: Grafo dell'automa a stati finiti deterministico dell'esercizio 1.

L'**unica osservazione** da effettuare è che lo stato  $q_0$  è finito poiché la macchina deve terminare sia per un numero pari di zeri, ma anche nel caso in cui venga si presenti una stringa vuota poiché zero è pari.

---

<sup>2</sup>Un numero dispari si rappresenta come  $2\mathbb{N} + 1$ .

**Dimostrazione.** Si vuole dimostrare inizialmente che  $\sigma \in L \iff \delta(q_0, \sigma) \in F$  tramite l'induzione.

**Caso base.**

1.  $x = \varepsilon \in L \rightarrow \delta(q_0, \varepsilon) = q_0 \in F$
2.  $x = 0 \notin L \rightarrow \delta(q_0, 0) = q_1 \notin F$

In altre parole, se la stringa in entrata è vuota, cioè  $\varepsilon$ , allora l'automa rimane in  $q_0$ , cioè lo stato finito e quindi appartiene al linguaggio e all'insieme  $F$ . Nel caso in cui la stringa non è vuota, dallo stato  $q_0$  si finisce in  $q_1$ , il quale non appartiene né al linguaggio né all'insieme  $F$  dato che non l'automa non finisce in uno stato di fine.

**Ipotesi induttiva.**

Per  $x$  con  $|x| < n$ , cioè con la lunghezza della stringa  $x$  minore di  $n$ , allora  $x \in L \implies \delta(q_0, x) \in F$ .

Allora si ipotizzi una  $x$  di lunghezza  $n + 1$  che dunque non appartiene né al linguaggio:

$$x' = x0 \notin L$$

Né all'insieme  $F$ :

$$\delta(q_0, x') = \delta(q_0, x0) = \delta(q_0, 0) = q_1 \notin F$$

Dallo stato  $q_0$  con una stringa  $x$  si rimane in  $q_0$  (formalmente  $\delta(q_0, x) = q_0$ ). È stato inserito uno 0 nella dimostrazione, ma era possibile dimostrarlo anche tramite 1, cioè:

$$x' = x1 \notin L$$

Né all'insieme  $F$ :

$$\delta(q_0, x') = \delta(q_0, x1) = \delta(q_0, 1) = q_0 \in F$$

QED

**Dimostrazione 2.** Questa seconda dimostrazione ha l'obbiettivo di dimostrare, pardon per il gioco di parole, l'espressione  $\sigma \notin L \implies \delta(q_0, \sigma) \notin F$  tramite l'induzione. La differenza dalla dimostrazione precedente sta nel dimostrare la *non appartenenza* al linguaggio e all'insieme degli stati finiti  $F$ .

**Caso base.**

Si utilizza quello della dimostrazione precedente, quindi:

1.  $x = \varepsilon \in L \implies \delta(q_0, \varepsilon) = q_0 \in F$
2.  $x = 0 \notin L \implies \delta(q_0, 0) = q_1 \notin F$

**Ipotesi induttiva.**

Per  $x$  con  $|x| < n$ , cioè con la lunghezza della stringa  $x$  minore di  $n$ , allora  $x \notin L \implies \delta(q_0, x) \notin F$ .

In questa dimostrazione si considerano le stesse casistiche precedenti, quindi con 0 e 1:

$$\begin{aligned} x' = x0 \in L &\implies \delta(q_0, x') = \delta(q_0, x_0) = \delta(q_1, 0) = q_0 \in F \\ x' = x1 \in L &\implies \delta(q_0, x') = \delta(q_0, x_1) = \delta(q_1, 1) = q_1 \notin F \end{aligned}$$

QED

### **Esercizio 2**

I dati dell'esercizio sono i seguenti:

$$L = \{x \mid |x_0| = 1\}$$
$$\Sigma = \{0, 1\}$$

*Risoluzione.*

L'automa da costruire si basa sulla condizione del linguaggio, ovvero che  $x_0$  (**numero degli zeri di  $x$** ) sia uguale a 1.

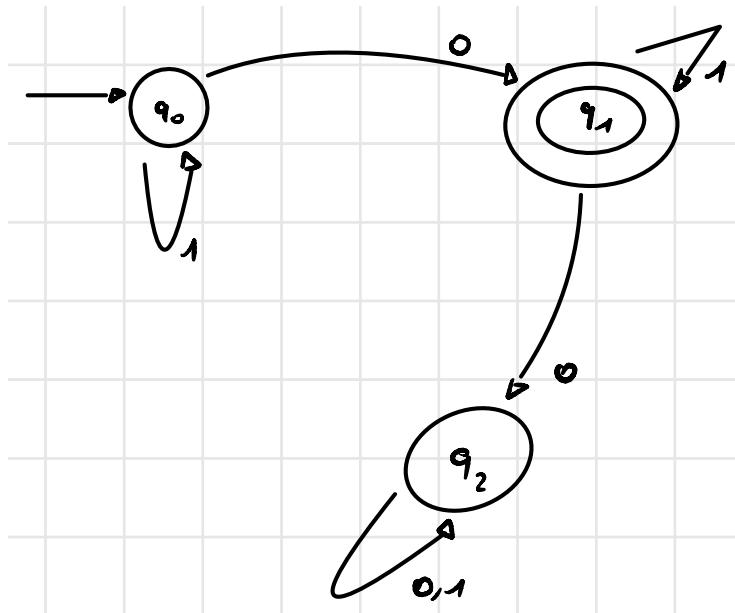


Figura 7: Grafo dell'automa a stati finiti deterministico dell'esercizio 2.

Sono **due le osservazioni** da fare:

1. Dato che è necessario solo uno zero per terminare, lo stato  $q_1$  è obbligatoriamente uno stato finito;
2. Lo stato  $q_2$  esiste solamente per bloccare la macchina nel caso in cui la sequenza sia errata, ovvero nel momento in cui vengano inseriti più zeri. Questo stato particolare prende il nome di "stato pozzo".

**Dimostrazione.** Si vuole dimostrare che  $\sigma \in L \iff \delta(q_0, \sigma) \in F$  tramite l'induzione.

**Caso base.**

1.  $x = \varepsilon \notin L \rightarrow \delta(q_0, \varepsilon) = q_0 \notin F$
2.  $x = 0 \in L \rightarrow \delta(q_0, 0) = q_1 \in F$
3.  $x = 00 \notin L \rightarrow \delta(\underbrace{q_0, 0}_q, 0) = \delta(q_1, 0) = q_2 \notin F$

Con il caso base sono stati ricoperti tutti gli stati.

**Ipotesi induttiva.**

Si vuole dimostrare che  $\sigma \in L \implies \delta(q_0, \sigma) \in F$ , allora l'ipotesi induttiva è: se  $|x| < n$  allora  $x \in L \implies \delta(q_0, x) \in F$ . Per farlo, si dimostrano le due casistiche classiche:

- $x' = x0 \notin L \rightarrow \delta(\overbrace{q_0, x}^{q_1}, 0) = \delta(q_1, 0) = q_2 \notin F$
- $x' = x1 \in L \rightarrow \delta(\overbrace{q_0, x}^{q_1}, 1) = \delta(q_1, 1) = q_1 \in F$

QED

**Dimostrazione 2.** Dopo la prima dimostrazione, adesso si vuole dimostrare che  $\sigma \notin L \implies \delta(q_0, \sigma) \notin F$  tramite l'induzione.

Come caso base si utilizza quello visto nella precedente dimostrazione.

### Ipotesi induttiva.

Si vuole dimostrare che  $\sigma \notin L \implies \delta(q_0, \sigma) \notin F$ , allora l'ipotesi induttiva è: se  $|x| < n$  allora  $x \notin L \implies \delta(q_0, x) \notin F$ . Per farlo, si devono trovare delle casistiche particolari:

- a.  $x = 1^n$
- b.  $x = 1^n 0 1^n 0 \{0, 1\}^n$

Le casistiche “generali” in cui una stringa non appartiene al linguaggio, e quindi all’insieme degli stati finali  $F$ , è quando la stringa è formata da una concatenazione di 1 (caso a), oppure quando una stringa è formata da una concatenazione di 1, uno zero, un’altra concatenazione di 1 e una concatenazione di simboli dell’alfabeto, quindi quando ci sono almeno due zeri (caso b).

Adesso si procede con la dimostrazione dei vari casi:

a.  $x = 1^n$

- $x' = x0 \in L \implies \delta(\overbrace{q_0, x}^{q_0} 0) = \delta(q_0, 0) = q_1 \in F$
- $x' = x1 \notin L \implies \delta(\overbrace{q_0, x}^{q_0} 1) = \delta(q_0, 1) = q_0 \notin F$

b.  $x = 1^n 0 1^n 0 \{0, 1\}^n$

- $x' = x0 \notin L \implies \delta(\overbrace{q_0, x}^{q_2} 0) = \delta(q_2, 0) = q_2 \notin F$
- $x' = x1 \notin L \implies \delta(\overbrace{q_0, x}^{q_2} 1) = \delta(q_2, 1) = q_2 \notin F$

QED

### **Esercizio 3**

I dati dell'esercizio sono i seguenti:

$$L = \{0^n \mid n = 3\mathbb{N}\} \xrightarrow{\text{forma alternativa}} L = \{0^{3\mathbb{N}}\}$$
$$\Sigma = \{0, 1\}$$

#### *Risoluzione.*

L'automa da costruire si basa sulla condizione del linguaggio, ovvero che  $n$  (**valore della variabile  $n$** ) sia uguale a  $3\mathbb{N}$  (**multipli di 3**).

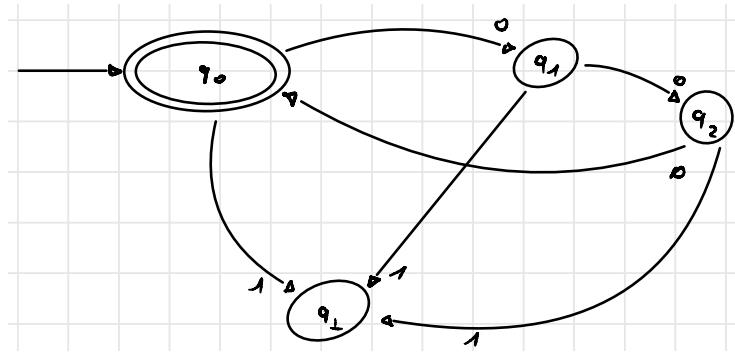


Figura 8: Grafo dell'automa a stati finiti deterministico dell'esercizio 3.

L'unica **osservazione** da fare riguarda lo stato  $q_{\perp}$  è il suo significato. Nell'esercizio 2 a pagina 21 si parla di "**stato pozzo**", in questo esercizio lo stato  $q_{\perp}$  è la stessa identica cosa.

**Dimostrazione.** Si vuole dimostrare che  $\sigma \in L \iff \delta(q_0, \sigma) \in F$  tramite l'induzione.

**Caso base.**

1.  $x = \varepsilon \in L \rightarrow \delta(q_0, \varepsilon) = q_0 \in F$
2.  $x = 0 \notin L \rightarrow \delta(q_0, 0) = q_1 \notin F$
3.  $x = 00 \notin L \rightarrow \delta(\underbrace{q_0, 0}_q, 0) = \delta(q_1, 0) = q_2 \notin F$
4.  $x = 1 \notin L \rightarrow \delta(q_0, 1) = q_\perp \notin F$

Con il caso base sono stati ricoperti tutti gli stati.

**Ipotesi induttiva.**

Si vuole dimostrare che  $\sigma \in L \implies \delta(q_0, \sigma) \in F$ , allora l'ipotesi induttiva è: se  $|x| < n$  allora  $x \in L \implies \delta(q_0, x) \in F$ . Per farlo, si dimostrano le due casistiche classiche:

- $x' = x0 \notin L \rightarrow \delta(\overbrace{q_0, x}^{q_0}, 0) = \delta(q_0, 0) = q_1 \notin F$
- $x' = x1 \notin L \rightarrow \delta(\underbrace{q_0, x}_{q_0}, 1) = \delta(q_0, 1) = q_\perp \notin F$

QED

**Dimostrazione 2.** Dopo la prima dimostrazione, adesso si vuole dimostrare che  $\sigma \notin L \implies \delta(q_0, \sigma) \notin F$  tramite l'induzione.

Come caso base si utilizza quello visto nella precedente dimostrazione.

### Ipotesi induttiva.

Si vuole dimostrare che  $\sigma \notin L \implies \delta(q_0, \sigma) \notin F$ , allora l'ipotesi induttiva è: se  $|x| < n$  allora  $x \notin L \implies \delta(q_0, x) \notin F$ . Per farlo, si devono trovare delle casistiche particolari:

- a.  $x = 0^{3\mathbb{N}+1}$
- b.  $x = 0^{3\mathbb{N}+2}$
- c.  $x = \{0\}^n 1 \{0, 1\}^n$

Le casistiche “generali” in cui una stringa non appartiene al linguaggio, e quindi all'insieme degli stati finali  $F$ , è quando:

- La stringa è formata da una concatenazione di 0 multipli di 3 ma aggiungendo 1 al risultato, quest'ultimo non è più multiplo (caso a);
- Stessa situazione del punto precedente ma aggiungendo 2 come valore (caso b);
- La stringa è formata da una concatenazione di zeri, poi almeno un 1 e successivamente una serie di concatenazioni di 0 e/o 1.

Adesso si procede con la dimostrazione dei vari casi:

a.  $x = 0^{3\mathbb{N}+1}$

- $x' = x0 \notin L \implies \delta(\overbrace{q_0, x}^{q_1} 0) = \delta(q_1, 0) = q_2 \notin F$
- $x' = x1 \notin L \implies \delta(\overbrace{q_0, x}^{q_1} 1) = \delta(q_1, 1) = q_\perp \notin F$

b.  $x = 0^{3\mathbb{N}+2}$

- $x' = x0 \in L \implies \delta(\overbrace{q_0, x}^{q_2} 0) = \delta(q_2, 0) = q_0 \in F$
- $x' = x1 \notin L \implies \delta(\overbrace{q_0, x}^{q_2} 1) = \delta(q_2, 1) = q_\perp \notin F$

c.  $x = \{0\}^n 1 \{0, 1\}^n$

- $x' = x0 \notin L \implies \delta(\overbrace{q_0, x}^{q_\perp} 0) = \delta(q_\perp, 0) = q_\perp \notin F$
- $x' = x1 \notin L \implies \delta(\overbrace{q_0, x}^{q_\perp} 1) = \delta(q_\perp, 1) = q_\perp \notin F$

QED

## Esercizi con linguaggi e relativi grafi

Di seguito si riportano alcuni linguaggi dati come esercizio ed i relativi grafi.

### **Esercizio 4**

Il linguaggio e l'alfabeto:

$$L = \{0^n 1^m \mid n, m > 0\}$$
$$\Sigma = \{0, 1\}$$

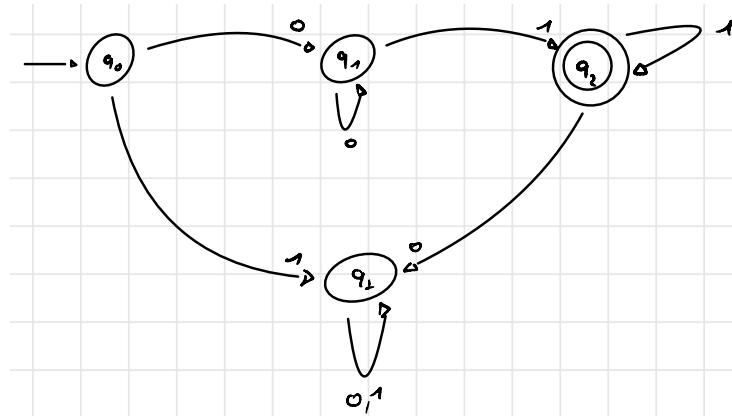


Figura 9: Grafo dell'automa a stati finiti deterministico dell'esercizio 4.

### Esercizio 5

Il linguaggio e l'alfabeto:

$$L = \{x \mid |x_0| > 0, |x_1| > 1\}$$
$$\Sigma = \{0, 1\}$$

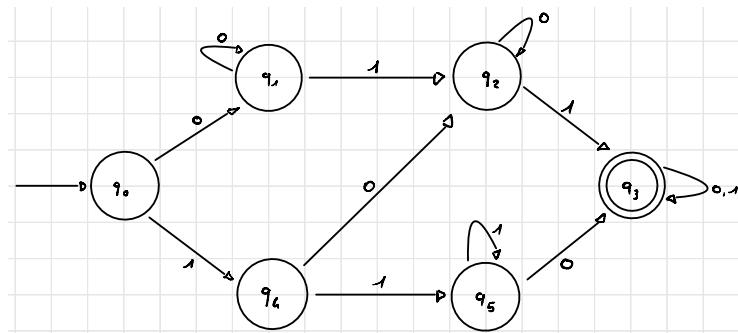


Figura 10: Grafo dell'automa a stati finiti deterministico dell'esercizio 5.

### 2.3.7 Automi con $\varepsilon$ -transizioni ( $\varepsilon$ -NFA)

Il terzo tipo di automa che estende il modello non-deterministico precedente ma che è equivalente dal punto di vista dei linguaggi accettati. L'idea è che l'automa compia transizione di stato anche senza leggere alcun simbolo. Questo sarà modellato ammettendo transizioni etichettate  $\varepsilon$ .

Un **automa a stati finti non deterministico con  $\varepsilon$  transizioni**, brevemente  $\text{ASFND}+\varepsilon$  oppure  $\varepsilon$ -NFA, è una quintupla:

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

Dove  $Q, \Sigma, \delta, q_0$  e  $F \subseteq Q$  sono definiti come per gli automi non-deterministici, mentre la **funzione di transizione**  $\delta$  è definita come:

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \longrightarrow P(Q)$$

Ne consegue che la funzione  $\hat{\delta} : Q \times \Sigma^* \longrightarrow P(Q)$  nel caso dei  $\varepsilon$ -NFA risulta più complessa.

Si introduce una nuova **operazione** chiamata  **$\varepsilon$ -chiusura** ( $\varepsilon$ -closure) che, applicata ad uno stato, **restituisce l'insieme degli stati raggiungibili da esso (compreso sè stesso) mediante  $\varepsilon$ -transizioni**. Essa non è altro che un insieme di stati:

$$\varepsilon\text{-closure}(P) = \bigcup_{p \in P} \varepsilon\text{-closure}(p)$$

Quindi, la  $\hat{\delta}$  è possibile definirla nel seguente modo:

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

In questo caso è possibile che  $\hat{\delta}(q, a)$  sia diverso da  $\delta(q, a)$ . Come, per esempio, nel seguente automa:



Figura 11: Esempio di automa con  $\hat{\delta}$  e  $\delta$  diversi.

“Consumando” la stringa  $a$  si giunge nello stato  $q'$ :  $\delta(q, a) = \{q'\}$ . Invece, con la stringa  $a\varepsilon$  si giunge allo stato  $q''$  e si descrive come:  $\hat{\delta}(q, a) = \bigcup_{p \in \hat{\delta}(q, \varepsilon)} \varepsilon\text{-closure}(\delta(p, a)) = \{q', q''\}$ .

Infine, si definisce il **linguaggio accettato dall'automa**:

$$L(M) = \left\{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) \cap F \neq \emptyset \right\}$$

E inoltre  $\hat{\delta}(q, x) = \varepsilon\text{-closure}(\hat{\delta}(q, x))$ .

### 2.3.8 Teorema dell'equivalenza di $\varepsilon$ -NFA e NFA

**Obiettivo:** con il seguente teorema si dimostra che la classe dei linguaggi riconosciuti dagli automi  $\varepsilon$ -NFA non estende propriamente quella dei linguaggi riconosciuti da automi NFA.

**Teorema 3 (Equivalezza).** *Sia una quintupla di un automa a stati finiti non-deterministico  $\varepsilon$  ( $\varepsilon$ -NFA):*

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

Allora esiste un automa a stati finiti non-deterministico (NFA)  $M'$  tale che  $L(M) = L(M')$ .

**Dimostrazione.** Si definisce l'automa a stati finiti non-deterministico  $M'$  con la quintupla:

$$M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$$

Con le caratteristiche simili alla NFA:

- $Q' = Q$
- $\Sigma' = \Sigma$
- $q'_0 = q_0$
- $F' = \begin{cases} F \cup (\varepsilon\text{-closure}(q_0)) & \text{se } \varepsilon\text{-closure}(q_0) \cap F \neq \emptyset \\ F & \text{altrimenti} \end{cases}$
- $\delta'(q, a) = \hat{\delta}(q, a)$

Si osservi che nel sistema degli stati finiti  $F$ , l'unione con lo stato iniziale  $q_0$  produce uno stato finito (se l'intersezione non è nulla!).

L'obiettivo è dimostrare che siano uguali:

$$\hat{\delta}(q_0, x) \cap F \neq \emptyset \iff \hat{\delta}'(q_0, x) \cap F' \neq \emptyset$$

Quindi, in altri termini:

$$\forall x \in \Sigma^* \longrightarrow \hat{\delta}(q_0, x) = \hat{\delta}'(q_0, x) \quad \text{per } |x| \geq 1$$

**Caso base.**

Entrambe hanno l'operazione  $\varepsilon$ -closure.

**Passo induttivo.**

$$\begin{aligned}
 \hat{\delta}'(q_0, \overbrace{xa}^{n+1}) &= \bigcup_{p \in \hat{\delta}'(q_0, x)} \delta'(p, a) \\
 &\xrightarrow{\text{per definizione}} \bigcup_{p \in \hat{\delta}'(q_0, x)} \hat{\delta}(p, a) \\
 &\xrightarrow{\text{per induzione}} \bigcup_{p \in \hat{\delta}(q_0, x)} \hat{\delta}(p, a) \\
 &\xrightarrow{\text{applicando l'operazione}} \bigcup_{p \in \hat{\delta}(q_0, a)} \varepsilon\text{-closure}(\delta(p, a)) \\
 &\xrightarrow{\text{si conclude}} \hat{\delta}(q_0, xa)
 \end{aligned}$$

Si **conclude la dimostrazione** verificando graficamente che  $F = F'$ . Per farlo, si pensi al caso in cui  $F' \neq F$ , ovvero un caso impossibile. Questo perché lo stato  $q_0$  è **per definizione** contenuto sia nell'insieme  $F$ , sia nell'insieme  $F'$  (dimostrazione eseguita sopra).

Quindi, lo stato  $q_0$ , se presente in uno dei due insiemi, sarà sicuramente presente anche nell'altro, come si vede dall'immagine 13.

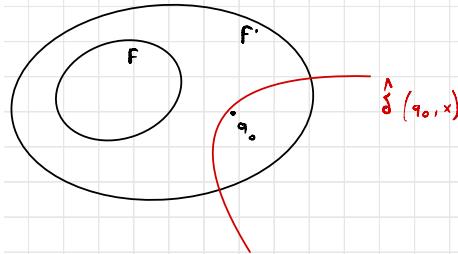


Figura 12: Dimostrazione grafica per evidenziare l'impossibilità della presenza dello stato  $q_0$  solamente in un insieme.

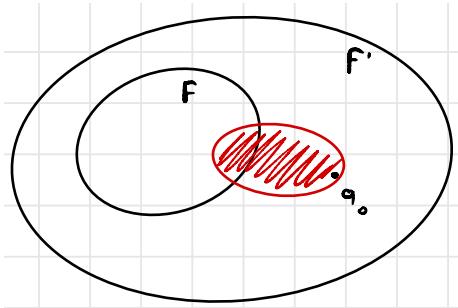


Figura 13: Dimostrazione grafica della corretta rappresentazione dei due insiemi e dello stato  $q_0$ .

QED

## 3 Espressioni regolari

### 3.1 Espressioni regolari

Sia  $\Sigma$  un alfabeto, si definiscono **espressioni regolari su  $\Sigma$**  e gli **insiemi** che esse denotano tramite le seguenti caratteristiche:

- I.  $\emptyset$  è una espressione regolare che indica l'**insieme vuoto**;
- II.  $\varepsilon$  è una espressione regolare che indica l'**insieme**  $\{\varepsilon\}$ ;
- III. Per ogni simbolo  $a \in \Sigma$ ,  $a$  è una espressione regolare che indica l'**insieme**  $\{a\}$ ;
- IV. Se  $r$  e  $s$  sono espressioni regolari che denotano rispettivamente gli insiemi  $R$  e  $S$ , allora:
  - $(r + s)$  denota l'insieme  $R \cup S$
  - $(rs)$  denota l'insieme  $R \cap S$  oppure con un'altra notazione  $RS$
  - $(r^*)$  denota l'insieme  $R^*$

Se  $r$  è una espressione regolare, si indica con  $L(r)$  il linguaggio denotato con  $r$ , ovvero l'insieme di stringhe di  $\Sigma^*$  che essa denota. Inoltre, varrà l'eguaglianza  $r = s$  se e solo se  $L(r) = L(s)$ .

Quindi, riassumendo la definizione:

- ☛  $L(\emptyset) = \emptyset$
- ☛  $L(\varepsilon) = \{\varepsilon\}$
- ☛  $L(a) = \{a\}$
- ☛  $L(r + s) = L(r) \cup L(s)$
- ☛  $L(r \cdot s) = L(r) \cdot L(s)$
- ☛  $L(r^*) = L(r)^*$

**Esempi:**

- $\{a^n b^m c^h \mid m, n, h \geq 0\} \longrightarrow a^* b^* c^*$
- $\{a^n b^m c^h \mid m, n, h > 0\} \longrightarrow aa^* bb^* cc^*$

**Proprietà**

- ★  $e_1 + (e_2 + e_3) = e_1 \cdot e_2 + e_1 \cdot e_3$
- ★  $(e_1 + e_2) \cdot e_3 = e_1 \cdot e_3 + e_2 \cdot e_3$
- ★  $e_1 + (e_2 + e_3) = (e_1 + e_2) + e_3$
- ★  $e_1 \cdot (e_2 \cdot e_3) = (e_1 \cdot e_2) \cdot e_3$
- ★  $e \cdot \varepsilon = \varepsilon \cdot e = e$
- ★  $e \cdot \emptyset = \emptyset \cdot e = \emptyset$
- ★  $e + e = e$
- ★  $\emptyset^* = \varepsilon$

### 3.1.1 Teorema McNaughton & Yamamada (1960) - Equivalenza tra DFA e ER

**Teorema 4 (McNaughton & Yamamada).** *Sia  $r$  una espressione regolare, allora esiste una macchina  $M$  (automa) a stati finiti non-deterministico +  $\varepsilon$  tale che  $L(M) = L(r)$ . Più formale:<sup>3</sup>*

$$r \in ER \implies L(r)$$

**Dimostrazione.** Si dimostra che esiste un automa a stati finiti non-deterministico +  $\varepsilon$  tale che  $L(r) = L(M)$ . Formalmente:

$$\exists M \underbrace{\text{ASFND} + \varepsilon}_{\varepsilon-\text{NFA}} \text{ t.c. } L(r) = L(M)$$

Per farlo, si hanno i seguenti tre casi base.

#### Caso base.

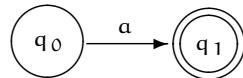
- L'automa  $\varepsilon$  è composto da un solo stato finito:



- L'automa  $\emptyset$  è composto da uno stato iniziale e uno stato finale che non viene mai raggiunto:



- L'automa  $a$  ha uno stato iniziale che accetta una stringa  $a$ , la quale la porta allo stato finale:



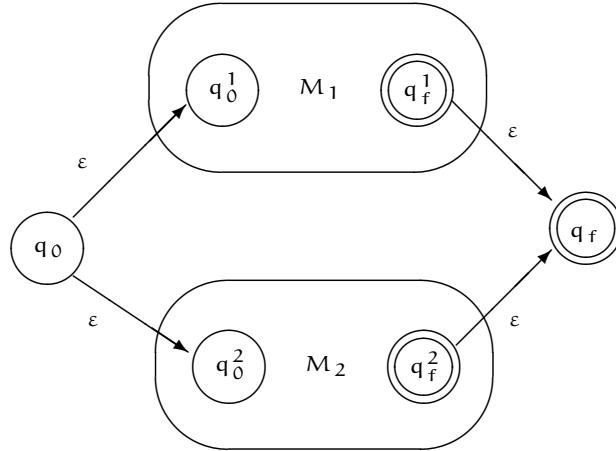

---

<sup>3</sup>ER = espressione regolare.

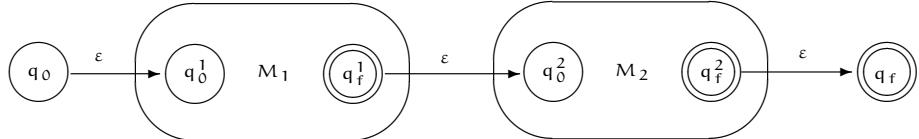
### Passo induttivo.

Ipotizzando che  $r$  ed  $s$  siano espressioni regolari  $r, s \in ER$  e si creano due macchine per le espressioni regolari:  $L(r) = M_1$  e  $L(s) = M_2$ . Si analizzano tre casi:

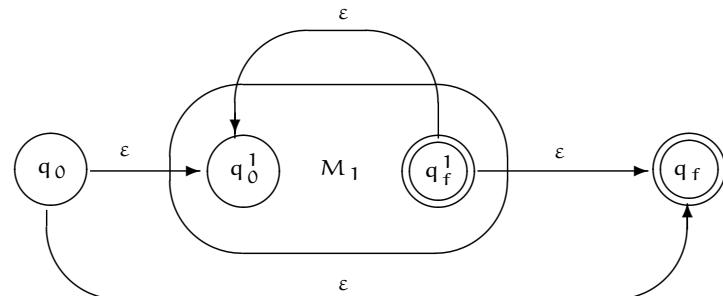
- $r + s$ . Questo automa riconosce il linguaggio  $L(r) \cup L(s)$



- $r \cdot s$ . Questo automa riconosce il linguaggio  $L(r) \cap L(s)$



- $r^*$ . Questo automa riconosce il linguaggio  $L(r)^*$



QED

### 3.1.2 Proprietà di chiusura

**Teorema 5 (Proprietà di chiusura).** I linguaggi regolari sono *chiusi rispetto alle operazioni di unione, concatenazione e chiusura di Kleene*.

**Dimostrazione.** Immediata dalla definizione di espressione regolare e dal Teorema di McNaughton & Yamada (1960). QED

**Teorema 6 (Complementazione).** I linguaggi regolari sono *chiusi rispetto all'operazione di complementazione*. Formalmente:

Se  $L \subseteq \Sigma^*$  è regolare, anche  $\bar{L} = \Sigma^* \setminus L$  è regolare

**Dimostrazione.** Sia  $M = \langle Q, \Sigma', \delta, q_0, F \rangle$  l'automa a stati finiti deterministico (DFA) che riconosce il linguaggio  $L$ . Si assume che  $\Sigma' = \Sigma$ , allora  $M' = \langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$  riconosce il linguaggio  $\bar{L}$ . QED

**Teorema 7 (Intersezione).** I linguaggi regolari sono *chiusi rispetto all'intersezione*.

**Dimostrazione.** Immediata dal fatto che  $L_1 \cap L_2 = \overline{(L_1 \cup L_2)}$  QED

## 4 Proprietà dei linguaggi regolari

### 4.1 Teorema di Myhill-Nerode (1957-58)

Prima di introdurre il teorema di Myhill-Nerode (1957-58), si espongono alcuni concetti base.

Dato un insieme  $S$ , una relazione di equivalenza  $R \subseteq S \times S$  induce univocamente una partizione di  $S = S_1 \cup S_2 \cup \dots$  ove per ogni  $i$  si ha  $S_i \neq \emptyset$  e per ogni  $i, j$  con  $i \neq j$  si ha che:

$$\text{I } S_i \cap S_j = \emptyset$$

$$\text{II } (\forall a, b \in S_i) : a R b$$

$$\text{III } (\forall a \in S_i) (\forall b \in S_j) : \neg(a R b)$$

Le varie  $S_i$  sono dette **classi di equivalenza**. La notazione è la seguente (caso in cui  $a$  appartenga alla classe di equivalenza  $S_i$ ):

$$a \in S_i \longrightarrow [a]_R$$

Se  $S$  è partizionato in un numero finito di classi  $S_1 \cup S_2 \cup \dots \cup S_k$  allora  $R$  viene detto **indice finito** su  $S$ .

Date due relazioni  $R_1$  e  $R_2$  sullo stesso insieme  $S$ ,  $R_1$  è un **raffinamento** di  $R_2$  se ogni classe di equivalenza della partizione indotta da  $R_1$  è sottoinsieme di qualche classe di equivalenza della partizione indotta da  $R_2$ .

**Per esempio**, se  $S = \{2, 3, 4, 5\}$ ,  $P_1 = \{\{2, 3, 5\}, 4\}$  e  $P_2 = \{\{2\}, \{3, 5\}, \{4\}\}$  denotano le partizioni di  $S$  ottenute a partire dalle seguenti relazioni:

$$(R_1) \quad x R_1 y \iff x \text{ e } y \text{ sono entrambi primi o uguali tra loro}$$

$$(R_2) \quad x R_2 y \iff x \text{ e } y \text{ sono entrambi primi e dispari, o uguali tra loro}$$

Allora  $R_2$  è un **raffinamento** di  $R_1$ .

Ad ogni linguaggio  $L \subseteq \Sigma$  è possibile associare una relazione di tipo  $R_L \subseteq \Sigma^* \times \Sigma^*$  definita nel seguente modo:

$$x R_L y \iff (\forall z \in \Sigma^*) (xz \in L \leftrightarrow yz \in L) \quad (5)$$

Questa definizione viene chiamata **relazione sul linguaggio**.

Analogamente, se  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  è una macchina a stati finiti deterministica (DFA), è possibile associare una relazione di tipo  $R_M \subseteq \Sigma^* \times \Sigma^*$  definita come:

$$x R_M y \iff \hat{\delta}(q_0, x) = \hat{\delta}(q_0, y) \quad (6)$$

Questa definizione viene chiamata **relazione sull'automa/macchina**. Inoltre, il linguaggio  $L_q = \{x \in \Sigma^* : \hat{\delta}(q_0, x) = q\}$  è associato allo stato  $q$  dell'automa. È evidente che le classi di equivalenza  $R_M$  sono esattamente i linguaggi associati ad ogni stato dell'automa  $M$ .

**Lemma 1 (Relazioni di equivalenza).** *La relazione sul linguaggio e la relazione sull'automa (o macchina) vengono chiamate **relazioni di equivalenza**.*

Infine, una relazione  $R \subseteq \Sigma^* \times \Sigma^*$  che gode della seguente proprietà:

$$x R y \xrightarrow{\text{implica}} (\forall z \in \Sigma^*) (xz R yz)$$

(la relazione) si chiama **invariante a destra** rispetto alla concatenazione. Quindi sia la relazione sul linguaggio che sull'automa/macchina è **invariante a destra**.

**Dimostrazione invariante a destra - Relazione sull'automa/macchina.**

La relazione sull'automa/macchina è invariante destra poiché aggiungendo una lettera  $z$  a  $x$ , il comportamento è identico anche per  $y$  (dimostrazione verbale, non matematica).

QED

**Dimostrazione invariante a destra - Relazione sul linguaggio.** Si ipotizzi per assurdo che data la relazione  $x R y$ , l'espressione che se ne deriva sia **falsa**, cioè:

$$\forall z \in \Sigma^* : xz R yz$$

Se ne deriva che dunque esiste:

$$\exists z \in \Sigma^* : \cancel{xz} R_L \cancel{yz}$$

Ovvero che la relazione è falsa.

Dalla relazione sul linguaggio:

$$\exists w \in \Sigma^* : x \underbrace{zw}_{z'} \in L, y \underbrace{zw}_{z'} \in L$$

Allora per assurdo si ottiene:

$$xz' \in L, yz' \notin L \implies x R_L y$$

QED

**Teorema 8 (Myhill-Nerode, 1957-58).** Le seguenti affermazioni si equivalgono:

1.  $L \subseteq \Sigma^*$  è regolare;
2.  $L$  è l'unione di classi di equivalenza su  $\Sigma^*$  indotte da una relazione invariante a destra di indice finito<sup>4</sup>;
3.  $R_L$  è di indice finito.

**Dimostrazione.** La dimostrazione avviene per implicazione, cioè che (1)  $\Rightarrow$  (2), (2)  $\Rightarrow$  (3) e (3)  $\Rightarrow$  (1).

$$\blacksquare (1) \Rightarrow (2)$$

Sia  $L$  un linguaggio regolare, come da definizione 1. È dunque ammesso un automa a stati finiti deterministico (ASFD o DFA) con la quintupla  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  tale per cui il linguaggio regolare è uguale al linguaggio dell'automa  $L = L(M)$ . L'automa è di **indice finito** poiché il numero di stati è finito.

Per definizione di linguaggio riconosciuto da un automa, si ha la seguente uguaglianza:

$$L = \bigcup_{q \in F} \overbrace{\left\{ x \in \Sigma^* \mid \hat{\delta}(q_0, x) = q \right\}}^{L_q}$$

Dato che la relazione sull'automa  $M$  ( $R_M$ ) è invariante a destra, l'espressione  $L_q$  rappresenta i vari insiemi che costituiscono le classi di equivalenza della partizione indotta da  $R_M$ .

In termini matematici:

$$\text{Se } x \in \Sigma^* : [x]_{R_M} = L_q \quad \text{per un qualsiasi } q \in Q$$

$$|Q| < w \Rightarrow L = \bigcup_{q \in F} L_q$$

Si ricorda che  $[x]_{R_M}$  indica la classe di equivalenza.

---

<sup>4</sup>Con indice finito si intende che l'insieme viene *partizionato* in classi finite.

• (2)  $\Rightarrow$  (3)

Assumendo che il punto 2 del teorema sia vero, si vuole dimostrare che l'indice è finito. Ovvero, ogni relazione di equivalenza  $R$  che soddisfa il punto 2 è un raffinamento di  $R_L$ .

Sia  $x \in \Sigma^*$ , si vuole dimostrare che  $[x]_R \subseteq [x]_{R_L}$ , cioè che il numero degli elementi in  $R_L$  è maggiore o uguale di  $R$ .

**Ipotesi induttiva:** si assuma che  $y \in [x]_R$  sia vera (N.B.  $\Rightarrow y \in [x]_{R_L}$ ). Dall'ipotesi induttiva e dal fatto che la relazione  $R$  è invariante a destra per ipotesi, allora:

$$\forall z \in \Sigma^* : xz R yz$$

Quindi, dato che  $L$  è l'unione delle classi di equivalenza di  $R$ :

$$L = [x_1]_R \cup \dots \cup [x_n]_R$$

Ciò implica che ogni qualvolta, in generale, si ha una relazione del tipo:

$$v R w \implies v \in L \iff w \in L$$

Pertanto, sostituendo i termini generali  $v$  e  $w$  rispettivamente con  $xz$  e  $yz$ :

$$\forall z \in \Sigma^* : \underbrace{xz}_v \in L \iff \underbrace{yz}_w \in L := x R_L y$$

La relazione sul linguaggio  $R_L$  tra  $x$  e  $y$  viene ricavata per definizione (come nell'equazione 5). Si può riscrivere anche come:

$$y \in [x]_{R_L}$$

Essendo la relazione  $R$  un raffinamento della relazione sul linguaggio  $R_L$ , l'indice di  $R_L$  è minore di quello di  $R$ , che per ipotesi è finito. Infatti, essendo un raffinamento, sicuramente in  $R$  l'indice sarà maggiore. Si conclude che  $R_L$  ha indice finito.

• (3)  $\Rightarrow$  (1)

Assumendo che la relazione sul linguaggio abbia un indice finito, si deduce che esiste un automa a stati finiti deterministico tale che  $L = L(M)$ . Quindi, si costruisce la macchina nel seguente modo:

ASFD (DFA)  $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$  che riconosce  $L$

- I.  $Q'$  è l'insieme (finito per ipotesi) di classi di equivalenza di  $R_L$ , formalmente:  $\{[x]_{R_L} \mid x \in \Sigma^*\}$ ;
- II.  $\Sigma'$  è lo stesso di  $L$ , cioè un alfabeto finito;
- III.  $\delta'([x], a) = [xa]$  dato che  $R_L$  è invariante a destra, la definizione vale indipendentemente dalla scelta di  $x$ ;
- IV.  $q'_0 = [\varepsilon]$ ;
- V.  $F' = \{[x]_{R_L} \mid x \in L\}$ .

Mostrando che  $L(M') = L$  si termina la dimostrazione.

Allora si suppone che esiste una  $y$ :

$$\forall y \in \Sigma^* : \hat{\delta}([x]_{R_L}, y) = [xy]_{R_L}$$

E con le diverse cardinalità di  $y$  si dimostra che:

- $|y| = 0 \iff y = \varepsilon : \hat{\delta}([x]_{R_L}, \varepsilon) = [x]_{R_L} = [x\varepsilon]_{R_L}$
- $|y| \geq 0 : \hat{\delta}([x]_{R_L}, ya) = \delta(\hat{\delta}([x]_{R_L}, y), a) = \delta([xy]_{R_L}, a) = [xya]_{R_L}$

Analogamente, si ha lo stesso ragionamento:

$$\hat{\delta}'(q'_0, x) = \hat{\delta}'([\varepsilon], x) = [\varepsilon, x] = [x]$$

Dunque si conclude la dimostrazione:

$$x \in L(M') \iff \hat{\delta}'(q'_0, x) \in F' \iff [x] \in F' \iff x \in L$$

Che tradotto vorrebbe dire che  $x$  appartiene al linguaggio dell'automa  $M'$  se e solo se la transizione dallo stato iniziale  $q_0$  consumando  $x$  sia uno stato finale, cioè appartenente a  $F$ , se e solo se la classe di equivalenza  $x$  della relazione sul linguaggio  $R_L$  appartiene all'insieme degli stati finali se e solo se  $x$  appartiene al linguaggio. QED

## 4.2 Pumping Lemma

Il pumping lemma è un **metodo utilizzato per dimostrare che un linguaggio non è regolare**.

**Lemma 2 (Bar-Hillel, Perles, Shamir, 1961).** *Sia  $L$  un linguaggio regolare. Allora esiste una costante  $n \in \mathbb{N}$  tale che per ogni  $z \in L$  tale che  $|z| \geq n$  esistono tre stringhe  $u, v, w$  tali che:*

1.  $z = uvw$  con  $u, v, w \in \Sigma^*$ ;
2.  $|uv| \leq n$ ;
3.  $|v| > 0$ , cioè non può essere vuota;
4. per ogni  $i \geq 0$  vale che  $uv^i w \in L$  o in forma più matematica:  $\forall i \geq 0 : uv^i w \in L$ .

**Dimostrazione grafica.** Data un automa a stati finiti deterministico (ASFD) o DFA con la quintupla  $M = \langle Q, \Sigma, \delta, q_0, F \rangle$  e con il linguaggio  $L(M) = L$ . Data la stringa  $z \in L = L(M)$  e con la lunghezza  $n = |Q| + 5$ .<sup>5</sup>

Quindi, l'automa si crea nel seguente modo:

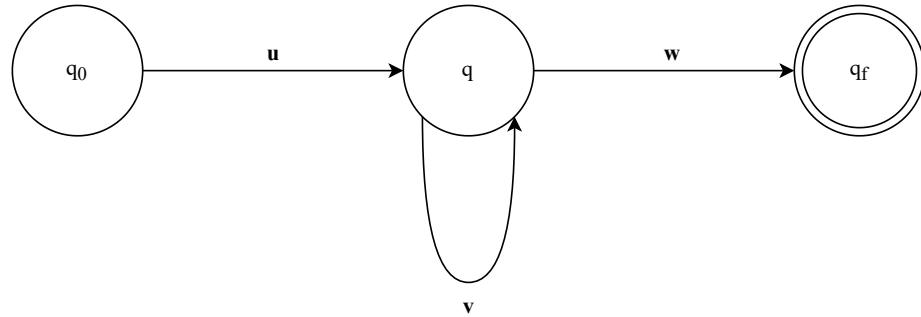


Figura 14: Automa a stati finiti deterministico.

**Punto 1, dimostrazione  $z = uvw$  con  $u, v, w \in \Sigma^*$ .**

La dimostrazione è immediata poiché si esegue creando l'automa della dimostrazione. Infatti, nell'automa si vede chiaramente la distinzione delle tre parti della stringa, ovvero la  $u$ , la  $v$  e la  $w$ .

**Punto 2, dimostrazione  $|uv| \leq n$ .**

La cardinalità di  $uv$  deve essere minore o uguale ad  $n$  poiché in caso contrario dovrebbe esserci un altro stato tra  $q_0$  e  $q$ . L'aggiunta di uno stato renderebbe la lunghezza della stringa più lunga di  $n$ .

---

<sup>5</sup>Il numero 5 è casuale, si potrebbe prendere anche 1, l'importante è che la lunghezza di  $n$  sia più grande.

**Punto 3, dimostrazione  $|v| > 0$ , cioè non può essere vuota.**

È abbastanza chiara la dimostrazione poiché se  $v$  non fosse maggiore di zero, lo stato non si ripeterebbe almeno una volta e questo andrebbe in contrasto con il punto 4 del *Pumping Lemma* in cui la  $v$  si ripete almeno una volta (con  $i = 0$  si ha una ripetizione di  $v$ ).

**Punto 4, dimostrazione  $\forall i \geq 0 : uv^i w \in L$ .**

Anche questa dimostrazione è immediata. Guardando l'automa a stati finiti è evidente che, dato un numero indefinito di  $v$ , si arriverà comunque allo stato finale una volta ricevuto il simbolo  $w$  nello stato  $q$ . QED

Si osservi come il lemma asserisca la veridicità di una **formula logica** che afferma: per ogni linguaggio  $L$ , se  $L$  è regolare, allora vale

$$\exists n \in \mathbb{N} \forall z \left( \begin{array}{l} (z \in L \wedge |z| \geq z) \Rightarrow \exists u, v, w \\ (z = uvw \wedge |uv| \leq n \wedge |v| > 0 \wedge \forall i (i \in \mathbb{N} \Rightarrow uv^i w \in L)) \end{array} \right) \quad (7)$$

Come detto all'inizio del paragrafo, l'**obiettivo** dell'utilizzo del Pumping Lemma è dimostrare che un dato linguaggio non è regolare. Per farlo, si assume che  $L$  non sia regolare e si deve mostrare che vale la negazione della formula (7), quindi:

$$\forall n \in \mathbb{N} \exists z \left( \begin{array}{l} z \in L \wedge |z| \geq n \wedge \\ \forall u, v, w \left( \begin{array}{l} z = uvw \wedge |uv| \leq n \wedge |v| > 0 \\ \Rightarrow \exists i (i \in \mathbb{N} \wedge uv^i w \notin L) \end{array} \right) \end{array} \right) \quad (8)$$

**Esempi di linguaggi non regolari**

I seguenti linguaggi sono irregolari:

- $L = \{a^n b^n \mid n \geq 0\}$
- $L = \{a^{n^2} \mid n \geq 1\}$

Al contrario, un linguaggio regolare è per esempio:

$$L = \{a^{2n} \mid n \geq 0\}$$

#### 4.2.1 Esercizi da esame

##### Esercizio 1

I dati dell'esercizio sono i seguenti:

$$L = \{1^k 0^i 1^i 0^j 1^j 0^k \mid i, j, k \geq 0\}$$

##### *Risoluzione.*

Per prima cosa si analizza il linguaggio per verificare se è regolare o no. Nel caso in cui lo fosse, si eseguono le procedure ampiamente spiegate al paragrafo 2.3.6. Tuttavia, in caso contrario, come in questo esercizio, si applica il Pumping Lemma.

Perché questo linguaggio non è regolare? Per capirlo basta vedere la sequenza accettata dal linguaggio. Infatti, in questo caso il numero di zeri (indice  $i$ ), per esempio, deve essere uguale al numero di uni (sempre indice  $i$ ). Per sapere il numero esatto di ripetizioni di zeri, per poi concatenare lo stesso numero di volte gli uni, è necessaria una memoria. Dunque, il linguaggio non è regolare.

Questo ragionamento può essere esteso sia per 1 e 0 con indice  $k$ , sia per 0 e 1 con indice  $j$ .

**Dimostrazione.** Per qualsiasi  $n$  si prende una stringa  $z$  la quale rappresenti il caso limite, ovvero quella situazione che non è gestibile da un ASFD senza memoria:

$$\forall n : z = 1^n 0^n \quad \text{con } n = 0 \text{ e } i, j = 0$$

Grazie al Pumping Lemma è noto per definizione che  $z = uvw$  e  $|uv| \leq n$ . Inoltre, la stringa con l'indice  $i$  del Pumping Lemma diventa:

$$z' = uv^i w = 1^{n+|v|(i-1)} 0^n$$

Se ne deduce che data la stringa  $z$ , la sua suddivisione è simile alla seguente figura:

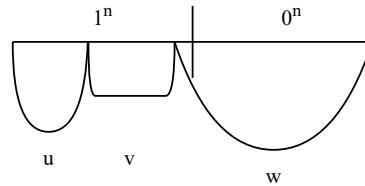


Figura 15: Rappresentazione della stringa  $z$ .

Si ipotizzi una  $i$  per rendere l'espressione più semplice da manipolare. Quindi, con  $i = 2$  l'espressione diventa:

$$z' = uv^i w = 1^{n+|v|(i-1)} 0^n = 1^{n+|v|(2-1)} 0^n = 1^{n+|v|} 0^n$$

Si analizzano gli esponenti per renderli uguali poiché gli 1 devono essere uguali, come quantità, agli 0:

$$n + |v| = n \xrightarrow{\text{approssimando}} \cancel{n} + |v| = \cancel{n}$$

Se ne conclude che  $|v| = 0$  e che per assurdo  $z \notin L$ .

QED

## **Esercizio 2**

I dati dell'esercizio sono i seguenti:

$$L = \{0^m 1^n 0^k 1^{2k} \mid m, n, k \in \mathbb{N}\}$$

### **Risoluzione.**

Il linguaggio potrebbe sembrare regolare, ma se si separa la parte che non provoca una “rottura”, si può osservare che serve una memoria e dunque è irregolare:

$$L' = (0^k 1^{2k} \mid k \in \mathbb{N})$$

**Dimostrazione.** Per qualsiasi  $k$  che faccia parte dei naturali, si cerca una stringa limite:

$$\forall k \in \mathbb{N} : z = 0^k 1^{2k}$$

Il pumping lemma assicura che sia vero che  $z = uvw$ ,  $|uv| \leq k$  e infine  $|v| > 0$ .

Si sceglie una  $i$ , ma prima si costruisce la stringa  $z'$ :

$$z' = uv^i w = 0^k 1^{2k} = 0^{k+|v|(i-1)} 1^{2k}$$

Si applica sempre la  $i = 2$  e  $z' \in L$ , cioè deve appartenere al linguaggio. Questo è vero solamente se:

$$z' \in L \iff 2k = 2(k + |v|)(i - 1)$$

Gli esponenti, per essere uguali, necessitano di una manipolazione. Infatti, il valore 2 è stato aggiunto anche a destra poiché il numero di 1 è il doppio degli zeri e dunque, per uguagliarli, si aggiunge un 2 al numero di zeri. Si sostituisce anche la  $i$  e si eseguono le semplificazioni:

$$\begin{aligned} 2k &= 2(k + |v|)(2 - 1) \\ \rightarrow 2\cancel{k} &= 2\cancel{k} + 2|v| \\ \rightarrow |v| &= 0 \end{aligned}$$

La conclusione per assurdo si conclude con  $|v| = 0$ , che non soddisfa il Pumping Lemma e dunque dimostra che il linguaggio non è regolare ma  $z' \in L'$ . QED

### **Esercizio 3**

I dati dell'esercizio sono i seguenti:

$$L = \{\omega \omega^R \mid \omega \in \{0, 1\}^*\}$$

N.B. con  $\omega^R$  si intende l'inversione (*reverse*) della stringa.

#### **Risoluzione.**

Il linguaggio è irregolare poiché per capire eseguire l'inversione di una stringa, si deve salvare la stringa originale.

**Dimostrazione.** Per qualsiasi  $n$  appartenente ai naturali, si considera la stringa limite  $z$ :

$$\forall n \in \mathbb{N} : z = \underbrace{0^n}_\omega 1 \underbrace{1 0^n}_{\omega^R}$$

Il pumping lemma ci assicura che  $z = uvw$ ,  $|uv| \leq n$  e  $|v| > 0$ . Si scrive dunque la stringa:

$$z' = uv^i w = 0^{n+|v|(i-1)} 1 1 0^n$$

Si sceglie una  $i = 0$ :

$$z = 0^{n+|v|(i-1)} 1 1 0^n = 0^{n+|v|(-1)} 1 1 0^n \longrightarrow 0n - |v| 1 = 0^n 1$$

Si analizzano gli esponenti:

$$\cancel{x} - |v| = \cancel{x} \longrightarrow |v| = 0$$

La dimostrazione si conclude perché l'assurdo è verificato.  $|v| = 0$  e la stringa  $z \notin L$ . QED

#### **Esercizio 4**

I dati dell'esercizio sono i seguenti:

$$L = \{a^m b^n \mid m > n\}$$

#### **Risoluzione.**

Il linguaggio è irregolare poiché per sapere quando  $m$  è maggiore di  $n$  è necessaria una memoria.

**Dimostrazione.** Si sceglie una qualsiasi  $n$  tale che la stringa limite:

$$\forall n : z = a^n b^{n-1}$$

Appartenga al linguaggio, cioè  $\in L$ . Il pumping lemma assicura che  $|uv| \leq n$ ,  $|v| > 0$  e  $z = uvw$ . Allora, esiste una stringa  $z'$  tale per cui:

$$z' = a^{n+|v|(i-1)} b^{n-1}$$

Viene scelta una  $i = 0$  per la dimostrazione per assurdo:

$$z' = a^{n+|v|(i-1)} b^{n-1} = a^{n-|v|} b^{n-1}$$

Analizzando gli esponenti:

$$\begin{aligned} n - |v| &> n - 1 \\ \rightarrow |v| &< 1 \\ \rightarrow |v| &= 0 \end{aligned}$$

Il linguaggio non è regolare ed è stato dimostrato per assurdo.

QED

### **Esercizio 5**

I dati dell'esercizio sono i seguenti:

$$L = \{0^P \mid P \text{ è un numero primo}\}$$

#### **Risoluzione.**

Il linguaggio è irregolare poiché per calcolare un numero primo è necessario eseguire una serie di operazioni che necessitano di memoria per salvare il risultato.

**Dimostrazione.** Per ogni  $n$  con la stringa limite  $z$ :

$$\forall n : z = 0^P \quad \text{con } P \text{ primo maggiore di } n \rightarrow P > n$$

Il pumping lemma assicura che  $|uv| \leq n$ ,  $|v| > 0$  e  $z = uvw$ . Allora, esiste una stringa  $z'$  tale per cui:

$$z' = uv^i w = 0^{P+|v|(i-1)}$$

Scegliere una  $i$  per dimostrare per assurdo non è semplice. Quindi, si ragiona in modo differente. Data una certa  $i$  che tolto il numero 1 dia un numero primo, si ha:

$$P + |v| \underbrace{(i-1)}_P$$

Andando a sostituire e raccogliendo i termini:

$$P + |v|P \longrightarrow (P)(1 + |v|)$$

Da questa espressione si evince che un numero primo, per essere tale, deve essere moltiplicato per sé stesso o per 1. Quindi, sicuramente l'espressione  $1 + |v|$  deve essere uguale a 1:

$$\begin{aligned} 1 + |v| &= 1 \\ \rightarrow \cancel{1} + |v| &= \cancel{1} \\ \rightarrow |v| &= 0 \end{aligned}$$

Il linguaggio non è regolare ed è stato dimostrato per assurdo grazie alla negazione del pumping lemma. QED

## **Esercizio 6 - Linguaggio parametrizzato**

I dati dell'esercizio sono i seguenti:

$$L(m, k) = \{a^n b^m c^k \mid n - m \leq k\}$$

L'esercizio è diverso dagli altri poiché il linguaggio è parametrizzato.

### **Risoluzione.**

Per capire se il linguaggio è regolare o non regolare, si inseriscono alcuni valori noti. Infine, si prova con l'intersezione e l'unione dell'insieme dei numeri naturali  $\mathbb{N}$ .

**Dimostrazione.** Si provano prima i valori  $m = 0, k = 0$ :

$$L(0, 0) = \{a^n b^0 c^0 \mid n \leq 0\} = \{\varepsilon\}$$

In questo caso corrisponde alla stringa vuota. Al contrario, con i valori  $m = 0, k = 1$ :

$$L(0, 1) = \{a^n b^0 c^1 \mid n \leq 1\}$$

Per  $m$  e  $k$  fissati, l'insieme è finito e quindi è regolare.

Adesso si esegue la prova con l'unione di  $m, k$  con l'insieme dei naturali  $\mathbb{N}$ :

$$\bigcup_{m, k \in \mathbb{N}} L(m, k) = \{a^n b^m c^k \mid n - m \leq k, \quad m, n, k \in \mathbb{N}\}$$

Il linguaggio risulta non regolare poiché sarebbe necessario tenere conto del risultato di  $n - m$  e successivamente confrontarlo con  $k$ . Quindi, si prosegue con la dimostrazione per assurdo e utilizzando il pumping lemma:

$$\forall n : z = a^{2n} b^n c^n$$

Il pumping lemma assicura che  $z = uvw$ ,  $|uv| \leq n$  e  $|v| > 0$ . Si definisce la stringa  $z'$ :

$$z' = a^{2n+(i-1)} b^n c^n$$

Si sceglie una  $i$ , in questo caso uguale a 2:

$$z' = a^{2n+|v|} b^n c^n$$

Analizzando gli esponenti:

$$2n + |v| \leq 2n \longrightarrow |v| \leq 0$$

Dimostrato per assurdo.

Invece, si prova l'operazione di **intersezione** di  $m, k$  con l'insieme dei naturali  $\mathbb{N}$ :

$$\bigcap_{m,k \in \mathbb{N}} L(m, n) = \emptyset$$

L'insieme è vuoto poiché basta vedere cosa accade nel caso in cui si prova a ricavare l'insieme di intersezione dal linguaggio parametrizzato con valori noti. Infatti, all'inizio della risoluzione, sono state eseguite alcune sostituzioni che hanno portato a due risultati. Eseguendo l'intersezione tra questi due insiemi, si può osservare come l'insieme risultante sia vuoto. QED

## Esercizio 7 - Linguaggio parametrizzato

I dati dell'esercizio sono i seguenti:

$$L(m) = \{\sigma = \{0,1\}^* \mid |\sigma_0| = |\sigma_1|^m\}$$

L'esercizio riguarda sempre un linguaggio parametrizzato.

### *Risoluzione.*

Anche in questo caso, si provano due casi.

**Dimostrazione.** Con il valore 0 il linguaggio dà il seguente risultato:

$$L(0) = \{\sigma = \{0,1\}^* \mid |\sigma_0| = |\sigma_1|^0 = 1\}$$

Il linguaggio in questo caso è regolare e dunque si crea l'automa e la sua relativa dimostrazione (esercizio identico a quello già visto nel paragrafo 2.3.6). Al contrario, con il valore 1 il linguaggio corrisponde a:

$$L(1) = \{\{0,1\}^* \mid |\sigma_0| = |\sigma_1|\}$$

In questo caso, il linguaggio non è regolare perché non è possibile contare il numero di 0 e 1 per eseguire successivamente il confronto. Quindi, si prosegue con il pumping lemma:

$$\forall n : z = 0^n 1^n$$

Il lemma assicura che  $z = uvw$ ,  $|uv| \leq n$  e  $|v| > 0$ . Si crea dunque la stringa  $z'$ :

$$z' = 0^{n+|v|(i-1)} 1^n$$

Si sceglie una  $i$  per dimostrare che il linguaggio non è regolare. In questo caso  $i = 2$ :

$$z' = 0^{n+|v|(i-1)} 1^n = 0^{n+|v|} 1^n$$

Si analizzano gli esponenti:

$$\begin{aligned} n + |v| &= n \\ \rightarrow \cancel{n} + |v| &= \cancel{n} \\ \rightarrow |v| &= 0 \end{aligned}$$

Si conclude la dimostrazione per assurdo aggiungendo  $z' \notin L$

QED

### Esercizio 8 - Tratto da un tema d'esame

I dati dell'esercizio sono i seguenti:

$$L(m) = \{a^n b^m a^k \mid n \in m\mathbb{N} + 1, k \in \mathbb{N}\}$$

L'esercizio riguarda sempre un linguaggio parametrizzato. Inoltre, viene richiesta anche  $M$ , ovvero l'unione:

$$M = \bigcup_{m \in \mathbb{N}} L(m)$$

#### Risoluzione.

Come per gli altri linguaggi parametrizzati, si eseguono delle prove con dei valori per cercare di capire se c'è un punto fisso oppure se il linguaggio ad un certo punto diventa irregolare.

**Dimostrazione.** Con il valore 0 il linguaggio è definito nel seguente modo:

$$L(0) = \left\{ \underbrace{a^1 a^k}_{a^{1+k}} \mid n = 1, k \in \mathbb{N} \right\}$$

Ricordando che il linguaggio ha l'alfabeto definito nel seguente modo  $\Sigma = \{a, b\}$ , l'automa è rappresentato nella figura seguente.

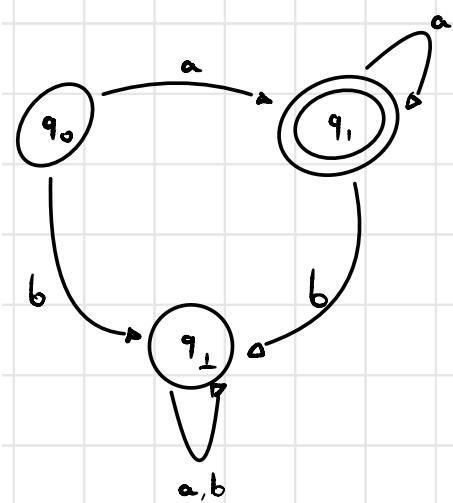


Figura 16: Grafo del linguaggio parametrizzato con il valore 0.

Si esegue la dimostrazione del linguaggio regolare.

**Dimostrazione**  $L(0)$ . Si dimostra tramite induzione.

**Caso base.**

- $x = \varepsilon \notin L(0) \quad \delta(q_0, \varepsilon) = q_0 \notin F$
- $x = a \in L(0) \quad \delta(q_0, a) = q_1 \in F$
- $x = b \notin L(0) \quad \delta(q_0, b) = q_\perp \notin F$

**Passo induttivo.**

Si dimostra per induzione che:

$$\sigma \in L(0) \implies \delta(q_0, \sigma) \in F$$

Supponendo che sia vero che la lunghezza della stringa  $x$  sia minore di  $n$ , cioè  $|x| < n$ , allora:

$$x \in L(0) \implies \delta(q_0, x) \in F$$

A  $x$  si assegna  $a^+$ , **N.B.**<sup>6</sup>, la  $a^+$ :

- $x' = xa \in L \quad \delta(q_0, xa) = \delta(q_1, a) = q_1 \in F$
- $x' = xb \notin L \quad \delta(q_0, xb) = \delta(q_1, b) = q_\perp \notin F$

Invece, adesso si dimostra il contrario:

$$\sigma \notin L(0) \implies \delta(q_0, \sigma) \notin F$$

Supponendo che sia vero che la lunghezza della stringa  $x$  sia minore di  $n$ , cioè  $|x| < n$ , allora:

$$x \notin L(0) \implies \delta(q_0, x) \notin F$$

Si prende in considerazione il caso in cui la  $x$  è:

$$x = a^* b \{a, b\}^*$$

E si conclude la dimostrazione:

- $x' = xa \notin L(0) \quad \delta(q_0, xa) = \delta(q_\perp, a) = q_\perp \notin F$
- $x' = xb \notin L(0) \quad \delta(q_0, xb) = \delta(q_\perp, b) = q_\perp \notin F$

QED

---

<sup>6</sup>Il segno del più indica un numero di concatenazione che è maggiore di zero ( $a^+ = a^n$ , con  $n > 0$ ). Al contrario,  $a^* = a^n$  con  $n \geq 0$ .

Adesso, con il linguaggio parametrizzato si sostituisce il valore 1:

$$L(1) = \{a^n b a^k \mid n > 0, k \geq 0\}$$

L'automa a stati finiti deterministico è il seguente. La dimostrazione di questo linguaggio regolare si salta solamente per evidenziare la diversificazione con altri esercizi.

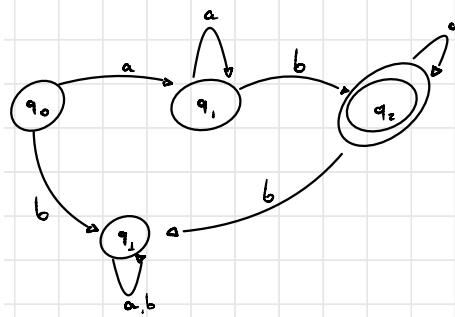


Figura 17: Automa a stati finiti deterministico del linguaggio parametrizzato con il valore 1.

Si provano altri due, tre valori per verificare se il linguaggio continua ad essere regolare oppure no:

- $L(2) = \{a^n bb a^k \mid n = 2\mathbb{N} + 1, k \geq 0\}$
- $L(3) = \{a^n bbb a^k \mid n = 3\mathbb{N} + 1, k \geq 0\}$
- $L(4) = \{a^n bbbb a^k \mid n = 4\mathbb{N} + 1, k \geq 0\}$

Nonostante siano stati provati 3 valori diversi, il linguaggio rimane regolare. Dunque, la dimostrazione si conclude dicendo che è stato **trovato un punto fisso**, ovvero il linguaggio è sempre regolare. QED

**Dimostrazione L(0).** Si dimostra la seguente unione:

$$M = \bigcup_{m \in \mathbb{N}} L(m) = \{a^n b^m a^k \mid n \in m\mathbb{N} + 1, k \geq 0, m \geq 0\}$$

Si dimostra tramite il pumping lemma che per qualsiasi  $h$  esiste una stringa  $z$  tale per cui:

$$\forall h \quad \exists z = a^{h+2} b^{h+1}$$

Il pumping lemma, per definizione, afferma che  $z = uvw$ , che  $|uv| \leq h$  e infine che  $|v| > 0$ . Supponendo che  $m = h + 1$ , che  $k = 0$  e che:

$$n = h+2 \in (h+1)\mathbb{N}+1 = \begin{cases} 1 \\ h+1+1 = h+2 \xleftarrow{\text{elemento scelto per la dimostrazione}} \\ 2(h+1)+1 = 2h+3 \\ \dots \end{cases}$$

L'obiettivo è dimostrare che “pompano” non si arriva né all'elemento precedente (1), né all'elemento successivo ( $2(h+1)+1 = 2h+3$ ). Si sceglie la classica stringa  $z$ :

$$z' = uv^i w = a^{h+2+(i-1)|v|} b^{h+1} \quad \text{con il vincolo } 0 < |v| \leq h$$

Si sceglie una  $i$  per concludere la dimostrazione, in questo caso uguale a 2:

$$z' = a^{h+2+|v|} b^{h+1} \quad \text{con il vincolo } |v| \leq h$$

Quindi l'intervallo in cui ricade la  $|v|$  è:

$$h+2 < h+2+|v| \leq 2h+2 < 2h+3 \xrightarrow{\text{si deduce}} z' \notin L$$

#### Soluzione alternativa: uguaglianza tra esponenti.

Esiste una alternativa, ovvero l'utilizzo dell'uguaglianza tra esponenti per dimostrare la negazione del pumping lemma. Per farla si prende in considerazione la stringa corrispondente alla  $|v|$ :

$$\begin{array}{ccccccc} h+2 & \rightarrow & h+2 = h+2+|v| & \rightarrow & |v|=0 & & \checkmark \\ \nearrow & & & & & & \\ h+2+|v| = & & & & & & \\ \searrow & & & & & & \\ 2h+3 & \rightarrow & 2h+3 = h+2+|v| & \rightarrow & |v|=h+1 & & \checkmark \end{array}$$

La prima espressione è dimostrata poiché chiaramente  $|v|=0$  non rispetta la definizione del pumping lemma. Anche la seconda è dimostrata perché  $|v|$  al massimo può essere  $h$  non  $h+1$ .

QED

### **Esercizio 9**

I dati dell'esercizio sono i seguenti:

$$L = \{0^{k^2}\}$$

L'esercizio riguarda un linguaggio e bisogna capire se è regolare oppure no.

#### **Risoluzione.**

Il linguaggio non è regolare e si capisce dal fatto che non è possibile costruire una macchina che concatensi una serie di 0 per  $k$  volte alla seconda.

**Dimostrazione.** Per ogni  $n$  prendendo la stringa  $z$ :

$$\forall n : z = 0^{n^2}$$

Il pumping lemma assicura che  $z = uvw$ , che  $|uv| \leq n$  e che  $|v| < 0$ . Si costruisce la stringa  $z'$ :

$$z' = 0^{n^2 + |v|(i-1)} \quad (9)$$

Scegliendo una  $i$  arbitraria, in questo caso 0. Si esegue la dimostrazione tramite uguaglianza degli esponenti e disuguaglianza.

#### Uguaglianza.

$$\begin{array}{ccc}
 \cancel{n^2 - |v|} & \nearrow & \cancel{|v| = 0} \\
 & & \rightarrow \quad |v| = 0 \quad \checkmark \\
 & \searrow & \\
 n^2 - |v| & & n^2 - |v| = (n-1)^2 \quad \rightarrow \quad \cancel{n^2 - |v|} = \cancel{n^2 - 2n + 1} \quad \rightarrow \\
 & & |v| = 2n - 1 \quad \rightarrow \quad |v| \leq n < 2n < -1 \quad \checkmark
 \end{array}$$

#### Disuguaglianza.

$$n^2 - n < n^2 - |v| < n^2$$

QED

## 5 Linguaggi liberi dal contesto

### *Introduzione*

Una grammatica è un insieme di regole che permettono di generare un linguaggio. La situazione è quindi diversa rispetto ai linguaggi regolari. Infatti, quest'ultimi sono stati definiti come linguaggi *riconosciuti* da una particolare famiglia di automi, i linguaggi in questo capitolo saranno *generati* da particolari insiemi di regole. Un ruolo fondamentale in questo contesto è giocato dalle grammatiche libere dal contesto (CF) mediante le quali viene solitamente descritta la sintassi dei linguaggi di programmazione. Dunque, si vedrà come questi linguaggi siano anche riconosciuti da particolari automi, detti *automi a pila*.

### 5.1 Linguaggi liberi dal contesto

Fino ad ora sono stati studiati i linguaggi regolari, i quali hanno visto la loro realizzazione pratica tramite gli automi. Un linguaggio, inoltre, veniva inizialmente studiato per capire se fosse regolare o no. Successivamente, si è rispettivamente eseguita la dimostrazione tramite automa (e matematica) per i primi e la negazione del Pumping Lemma per i secondi.

Si è dunque vista la necessità di introdurre un mezzo per manipolare i linguaggi non regolari. Un primo strumento che viene introdotto è la **grammatica libera dal contesto** (CF, *context free*), una quadrupla  $G = \{V, T, P, S\}$  in cui:

- $V$ , è un insieme finito di **variabili** (chiamati anche **simboli non terminali**);
- $T$ , è un insieme finito di **simboli terminali** ( $V \cap T = \emptyset$ ). È possibile vedere l'analogia con il  $\Sigma$ ;
- $P$ , è un insieme finito di **produzioni** e indica le regole grammaticali. Ogni produzione è della forma  $A \rightarrow \alpha$  in cui:
  - $A \in V$  è una variabile;
  - $\alpha \in (V \cup T)^*$ .
- $S \in V$ , è una variabile speciale chiamata **simbolo iniziale**.

## Esempi concettuali

### **Esempio 1**

Si mostra un esempio introduttivo al concetto di linguaggi liberi dal contesto.  
Dato il seguente linguaggio:

$$L = \{a^n b^n \mid n \geq 1\}$$

Si ha il seguente caso base:

$$S \longrightarrow ab$$

In cui  $S$  è sinonimo di  $L$ . Per dimostrarlo, si usa il passo induttivo:

$$S \longrightarrow a \underbrace{a^n b^n}_S b \xrightarrow{\text{quindi...}} S \longrightarrow a S b$$

Attenzione! L'ordine è importante, infatti se viene sostituita la  $S$ , la stringa rimane regolare e dunque nel linguaggio. Per esempio, in questo modo è *errato*:

$$S \longrightarrow a b S \quad \text{NO!}$$

### **Esempio 2**

Dato il seguente linguaggio:

$$L = \{a^{2n} b^{2n} \mid n \geq 0\}$$

Il simbolo iniziale è la  $\varepsilon$  e doppia  $a$ , doppia  $b$ :

$$\begin{array}{lcl} S & \longrightarrow & \varepsilon \\ S & \longrightarrow & a a S b b \end{array}$$

### **Linguaggi generati.**

Più in generale, il **linguaggio generato da  $G$**  è definito come:

$$L(G) = \left\{ w \in T^* \mid S \xrightarrow[G]{*} w \right\}$$

La freccia  $\xrightarrow[G]{*}$  indica una **derivazione finita**, ovvero con una serie finita di passi si arriva a  $w$ .

Inoltre,  $L$  è un **linguaggio libero dal contesto** (CF) se esiste una grammatica CF ( $G$ ) tale che  $L = L(G)$ . Si osservi che due grammatiche  $G_1$  e  $G_2$  sono equivalenti se  $L(G_1) = L(G_2)$

### Esempio 3

Il linguaggio è il seguente:

$$L = \{a^n b^n \mid n \geq 0\}$$

Si ottiene la grammatica del linguaggio:

$$\begin{array}{lcl} S & \longrightarrow & \varepsilon \\ S & \longrightarrow & a S b \end{array}$$

L'obbiettivo è dimostrare che il linguaggio sia  $L \subseteq L(G)$ , sia  $L = L(G)$  e infine  $L \supseteq L(G)$ .

**Dimostrazione caso  $L \subseteq L(G)$ .** Si definisce il caso base con  $n = 0$ :

$$n = 0 \xrightarrow{\text{sostituzione nella stringa}} a^0 b^0 = \varepsilon$$

Grazie alla prima regola l'affermazione è vera, dunque l'ipotesi induttiva è:

$$n \geq 0 \quad a^n b^n \implies S \xrightarrow{*} a^n b^n$$

E l'induzione è la seguente:

$$a^{n+1} b^{n+1} = a \underbrace{a^n b^n}_S b \implies S \xrightarrow{*} a S b \xrightarrow{*} a a^n b^n b = a^{n+1} b^{n+1} \quad \checkmark$$

QED

**Dimostrazione caso  $L(G) \supseteq L$ .** Si definisce il caso base con  $n = 1$ :

$$n = 1 \xrightarrow{\text{la grammatica è}} S \xrightarrow{*} \varepsilon \quad \text{con } \varepsilon \in L$$

L'ipotesi induttiva è:

$$n \geq 1 \quad \forall m \leq n : S \xrightarrow{*} x \implies x = a^k b^k \in L \quad \text{con } x \in T^*$$

E l'induzione è la seguente:

$$\underbrace{S \xrightarrow{*} a S b \xrightarrow{*} a x b}_{m+1} \xrightarrow{\text{sostituzione } x} a^{k+1} b^{k+1} \in L \quad \checkmark$$

QED

### **Esempio 4**

Il linguaggio è il seguente:

$$L = \{a^n b^h c^h d^n \mid n, h \geq 0\}$$

La grammatica generata non è con  $\varepsilon$ , invece è:

$$\begin{array}{lcl} S & \longrightarrow & \not{S} \quad \text{No} \\ S & \longrightarrow & a S d \quad \text{Sì} \end{array}$$

Per includere anche  $b$  e  $c$  si deve specificare:

$$\begin{array}{lcl} S & \longrightarrow & a S d \mid H \\ H & \longrightarrow & b H c \mid \varepsilon \end{array}$$

La grammatica ha  $H$  che può essere anche  $\varepsilon$  perché la variabile  $h$  può essere zero.

### **Esempio 5**

Dato il seguente linguaggio:

$$L = \{a^n b^n c^h \mid n, h \geq 0\}$$

In cui si raggruppa la stringa in due acronimi:

$$\begin{array}{lcl} A & = & a^n b^n \\ C & = & c^h \end{array}$$

Quindi, la grammatica relativa è:

$$\begin{array}{lcl} S & \longrightarrow & A C \\ A & \longrightarrow & a A b \mid \varepsilon \\ C & \longrightarrow & c C \mid \varepsilon \end{array}$$

### **Esempio 6**

Dato il seguente linguaggio:

$$L = \{a^n c^m b^n \mid n, m \geq 0\}$$

La grammatica che genera il linguaggio è:

$$\begin{array}{lcl} S & \longrightarrow & a S b \mid c \\ S & \longrightarrow & c C \mid \varepsilon \end{array}$$

**Esempio 7**

Dato il seguente linguaggio:

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

Il linguaggio può essere scomposto come:

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

La grammatica che genera il linguaggio è:

$$\begin{array}{lcl} S & \longrightarrow & 0 S 0 \mid A \\ A & \longrightarrow & 1 A 0 \mid \varepsilon \end{array}$$

**Esempio 8**

Dato il seguente linguaggio:

$$L = \{\sigma \in \{0, 1\}^* \mid \sigma \text{ è polindroma}\}$$

La grammatica che genera il linguaggio è:

$$\begin{array}{lcl} S & \longrightarrow & 0 \mid 1 \mid \varepsilon \\ S & \longrightarrow & 0 S 0 \mid 1 S 1 \end{array}$$

**Esempio 9**

Dato il seguente linguaggio:

$$L = \left\{ w \in \{a, b\}^+ \mid |w|_a = |w|_b \right\}$$

La grammatica che genera il linguaggio è:

$$S \longrightarrow a b \mid b a \mid a S b \mid b S a \mid S S$$

## 6 Funzioni parziali ricorsive di S.C. Kleene

### 6.1 Funzioni primitive ricorsive

Una funzione ricorsiva è una funzione dove il suo valore per un dato argomento è direttamente correlato al valore della medesima funzione su argomenti “più semplici” o al valore di funzioni “più semplici”.

Per esempio, la funzione definisce ricorsivamente la successione di Fibonacci:  
1, 1, 2, 3, 5, 8, 13, ···

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

Quindi, per **funzioni ricorsive di base** si intende le seguenti funzioni:

- La funzione **costante**  $0 : \lambda x.0$ ;
- La funzione **successore**  $S : \lambda x.x + 1$ ;
- La funzione **identità**, o  $i$ -esima proiezione,  $\pi_i : \lambda x_1 \cdots x_n. x_i$  con  $1 \leq i \leq n$ .

Con la lettera greca lambda  $\lambda$ , si parla di funzione che ha come argomento la funzione accanto e restituisce il valore dopo il punto (**lambda notation**). Per esempio, con l'espressione  $\lambda x.0$ , si intende che la funzione lambda prende la funzione  $x$  come argomento e restituisce il valore 0 come output.

Inoltre, si definisce anche una funzione  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  che viene definita:

- Per **composizione** da  $g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$  e  $h : \mathbb{N}^k \rightarrow \mathbb{N}$  se:

$$f(x_1, \dots, x_n) = h(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$$

- Per **ricorsione primitiva** da  $g : \mathbb{N}^{n-1} \rightarrow \mathbb{N}$  e  $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  se:

$$\begin{cases} f(x_1, \dots, x_{n-1}, 0) = g(x_1, \dots, x_{n-1}) \\ f(x_1, \dots, x_{n-1}, y+1) = h(x_1, \dots, x_{n-1}, y, f(x_1, \dots, x_{n-1}, y)) \end{cases}$$

Tutte e solo le funzioni definibili a partire dalle precedenti funzioni ricorsive di base mediante composizione e ricorsione primitiva definiscono l'**insieme** delle **funzioni primitive ricorsive**. In altre parole, l'idea è costruire via via funzioni effettivamente calcolabili a partire dalle funzioni di base, come 0, costruendo da queste, per composizione e ricorsione primitiva, funzioni via via più complesse.

Si definisce la **classe  $\mathcal{P}$**  come la **classe delle funzioni primitive ricorsive**. Essa corrisponde alla più piccola classe, ovvero l'intersezione di tutte le classi, di funzioni contenenti le funzioni ricorsive di base e chiuse per composizione e ricorsione primitiva.

**Teorema 9.** *Le funzioni primitive ricorsive sono totali.*

Questo teorema pone una limitazione sulle funzioni primitive. Una funzione calcolabile può essere indefinita su alcuni, o tutti, i valori di input. Le funzioni primitive ricorsive hanno inoltre un'ulteriore limitazione anche all'interno delle funzioni calcolabili e totali sui naturali. Per dimostrare questo fatto, si utilizza una funzione evidentemente calcolabile e totale, ma non appartenente a  $\mathcal{P}$ , ovvero la **funzione di Ackermann**:

$$\begin{cases} \text{ack}(0, y) &= y + 1 \\ \text{ack}(x + 1, 0) &= \text{ack}(x + 1) \\ \text{ack}(x + 1, y + 1) &= \text{ack}(x, \text{ack}(x + 1, y)) \end{cases}$$

Alcuni esempi:

$$\text{ack}(0, y) = y + 1$$

$$\text{ack}(1, y) = 2 + (y + 3) - 3$$

$$\text{ack}(2, y) = 2(y + 3) - 3$$

$$\text{ack}(3, y) = 2^{y+3} - 3$$

$$\text{ack}(4, y) = 2^{\overbrace{2 \cdot 2}^{y+3}} - 3$$

La funzione ack cresce più velocemente di qualunque funzione ricorsiva primitiva. Questa funzione, universalmente accettata come funzione calcolabile, non può essere quindi primitiva ricorsiva pur essendo totale.

**Teorema 10.** *La funzione di Ackermann ack non è ricorsiva primitiva.*

## 6.2 Macchina di Turing

Indipendentemente dal problema che si intende risolvere, gli algoritmi hanno alcune caratteristiche comuni. Si elencano qui di seguito per comprendere meglio la macchina di Turing:

1. Un algoritmo è di lunghezza finita.
2. Esiste un agente di calcolo che porta avanti il calcolo eseguendo le istruzioni dell'algoritmo.
3. L'agente di calcolo ha a disposizione una memoria dove vengono immagazzinati i risultati intermedi del calcolo.
4. Il calcolo avviene per passi discreti.
5. Il calcolo non è probabilistico.
6. Non deve esserci alcun limite finito alla lunghezza dei dati di ingresso.
7. Non deve esserci alcun limite alla quantità di memoria disponibile.
8. Sono ammesse esecuzioni con un numero di passi finito ma illimitato.

I primi tre punti (1,2,3) sono ovvi.

Il punto 4 afferma che il calcolo non avviene mediante dispositivi analogici.

Il punto 5 afferma che il calcolo non obbedisce a nessuna legge di probabilità.

Il punto 6 è ragionevole poiché, per esempio, dato un algoritmo di somma, esso deve poter funzionare per ogni possibile addendo, ovvero qualsiasi numero naturale.

Il punto 7 si assume corretto poiché limitandola, alcuni algoritmi noti per calcolare semplici funzioni non potrebbero funzionare.

Il punto 8 afferma che non essendo limitabile il numero di passi richiesti per eseguire un generico algoritmo, allora non è possibile stabilire a priori un limite massimo sul numero di passi nell'esecuzione di un algoritmo.

L'idea di A.M. Turing fu quella di creare matematicamente la **Macchina di Turing** (MdT). L'idea è quella di pensare ad una semplice macchina in grado di scrivere simboli su un nastro potenzialmente infinito, rappresentando il fatto, anche espresso nel punto 7, che nei calcoli abbiamo a disposizione una quantità illimitata di carta dove registrare i risultati parziali del calcolo.

Il nastro è fatto di celle che rappresentano la quantità di memoria unitaria. La macchina utilizzerà un alfabeto finito, col quale possiamo rappresentare una infinità (numerabile) di dati. Il corpo pesante della macchina (detto controllo) sarà realizzato mediante un automa a stati finiti deterministico (DFA).

Una MdT è quindi un dispositivo di calcolo rappresentabile con un nastro di lunghezza illimitata nel quale vengono immagazzinati i dati o sequenze di simboli del calcolo. Uno di questi simboli è  $\$$  rappresentante l'assenza di simboli (spazio bianco). Il controllo MdT ha accesso al nastro attraverso una testina di lettura e scrittura che permette di leggere o scrivere un simbolo alla volta.

Ad ogni istante nel calcolo, il simbolo presente nella casella esaminata dalla testina rappresentata l'input alla macchina, in figura il simbolo  $s$ . In risposta la macchina può decidere di modificare il simbolo e/o spostare il nastro a sinistra o a destra della casella esaminata.

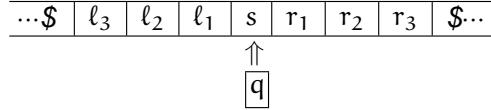


Figura 18: Il modello meccanico della Macchina di Turing.

**Teorema 11.** Una **descrizione istantanea** (ID, Instant Description) di una MdT è una quadrupla:

$$\langle q, v, s, w \rangle$$

dove  $v, w \in \Sigma^*$  rappresentano i caratteri significativi (cioè escludendo la sequenza illimitata di  $\$$  sempre presenti a destra e a sinistra) presenti sul nastro a sinistra ed a destra della testina,  $s$  è il simbolo letto dalla testina, essendo la macchina nello stato  $q$ .

Ad esempio, nella MdT in figura, la ID corrispondente è  $\langle q, l_3, l_2, l_1, s, r_1, r_2, r_3 \rangle$ .

Una MdT così descritta soddisfa pienamente i requisiti per la definizione ragionevole di algoritmo:

1. Le istruzioni della MdT sono finite.
2. La MdT è l'agente di calcolo che esegue l'algoritmo.
3. Il nastro rappresenta la memoria della macchina.
4. La MdT opera in modo discreto.
5. Ad ogni ID corrisponde una sola azione (determinismo della MdT).
6. Non esiste alcun limite all'input, essendo il nastro illimitato.
7. La capacità della memoria (nastro) è illimitata.
8. Le operazioni eseguibili sono semplici e quindi di complessità limitata.
9. Non esiste alcun limite al numero di istruzioni eseguite in quanto la medesima quintupla può essere usata più volte.
10. Possono esistere MdT che non calcolano nulla generando una sequenza infinita di ID.

Una funzione viene  $f : \mathbb{N}^n \rightarrow \mathbb{N}$  è **Turing calcolabile** se esiste una MdT  $M$  tale che, partendo dalla configurazione iniziale:

$$\dots \$ \$ \$ \underbrace{\$}_{q_0} \underline{x_1} \$ \dots \$ x_n \$ \$ \$ \dots$$

se  $f(x_1, \dots, x_n)$  è definita allora  $M$  termina nella configurazione:

$$\dots \$ \$ \$ \dots \underbrace{\$}_{q_f} \underline{f(x_1, \dots, x_n)} \$ \dots$$

si assume che a sinistra e a destra dell'input nella configurazione iniziale ci siano solo  $\$$ .

### 6.3 Funzioni parziali ricorsive di S.C. Kleene

Esiste un metodo generale per costruire funzioni parziali a partire da funzioni totali, ad esempio le primitive ricorsive.

Sia  $f$  una funzione totale  $n + 1$  aria, allora si definisce la funzione:

$$\begin{aligned}\varphi(x_1, \dots, x_n) &= \mu z. (f(x_1, \dots, x_n, z) = 0) \\ &= \begin{cases} \text{il più piccolo } z : f(x_1, \dots, x_n, z) = 0 & \text{se } z \text{ esiste} \\ \uparrow & \text{altrimenti} \end{cases}\end{aligned}$$

Tale operatore tra funzioni è detto  $\mu$ -operatore ed una funzione così definita è detta definita per minimizzazione o  **$\mu$ -ricorsione** da  $f$ . Chiaramente una funzione definita per minimizzazione è in generale, per definizione, parziale. In altre parole, si cerca lo  $z$  più piccolo che azzeri la funzione.

La classe delle funzioni **parziali ricorsive** è la minima classe  $\mathcal{P}_\mu$  di funzioni contenente  $\mathcal{P}$  e chiusa per  $\mu$ -ricorsione.

Il seguente risultato dimostra l'equivalenza delle funzioni calcolabili da MdT e le funzioni parziali ricorsive di S.C. Kleene  $\mathcal{P}_\mu$ .

**Teorema 12.**  $\varphi \in \mathcal{P}_\mu$  se e solo se  $\varphi$  è Turing-calcolabile.

Si definisce la **tesi di Church-Turing** come la classe delle funzioni “intuitivamente calcolabili” coincide con la classe delle funzioni Turing calcolabili.

**Teorema 13.** Esiste un programma chiamato **interprete** tale che  $\text{INT} : \mathbb{N} \times \mathbb{N}^n \rightarrow \mathbb{N}$ :

$$\varphi_{int}(x, y) = \begin{cases} \varphi_x(y) & \text{se } \varphi_x(y) \downarrow \\ \uparrow & \text{altrimenti} \end{cases}$$

## 6.4 Il teorema s-m-n

Il **teorema s-m-n** è dovuto a S.C. Kleene ed è fondamentale. Anch'esso rappresenta un esempio di dimostrazione basata sulla Tesi di Church-Turing.

**Teorema 14 (s-m-n).** *Per ogni coppia di interi  $m, n \geq 1$  esiste una funzione ricorsiva totale  $s_n^m$  di  $m + 1$  variabili tale che per ogni  $x, y_1, \dots, y_m$  si ha che:*

$$\forall z_1 \dots z_n. \varphi_x(y_1, \dots, y_m, z_1, \dots, z_n) = \varphi_{s_n^m(x, y_1, \dots, y_m)}(z_1, \dots, z_n)$$

Il teorema s-m-n gioca un ruolo essenziale nello sviluppo della teoria della ricorsività. Il significato è più o meno il seguente: se avete scritto un programma che implementa un certo algoritmo su  $m + n$  dati in input e da un certo momento in poi  $m$  di questi sono fissati, allora sapete ottenere un programma “**specializzato**” che accetta solo  $n$  parametri in input e che calcola la stessa funzione per questi  $n$  parametri.

**Teorema 15.** *Esiste una funzione ricorsiva  $g$  in due argomenti tale che per ogni  $x$  e  $y$ :  $\varphi_{g(x,y)} = \varphi_x \circ \varphi_y$*

---

## 6.5 Problemi insolubili

Esistono molti problemi insolubili, ma uno dei più classici è il **problema della terminazione** posto da Turing: esiste una procedura effettiva tale che dati  $x$  e  $y$  determina se  $\varphi_x(y)$  è definita o no?

Più formalmente, non esiste una funzione ricorsiva  $g$  tale che per ogni  $x$ :

$$g(x) = \begin{cases} 1 & \text{se } \varphi_x(x) \downarrow \\ 0 & \text{se } \varphi_x(x) \uparrow \end{cases}$$

**Teorema 16.** *Non esiste una funzione ricorsiva  $\varphi$  tale che per ogni  $x$  e  $y$ :*

$$\varphi(x, y) = \begin{cases} 1 & \text{se } \varphi_x(y) \downarrow \\ 0 & \text{se } \varphi_x(y) \uparrow \end{cases}$$

Non esiste una funzione ricorsiva  $f$  tale che per ogni  $x$ :

$$f(x) = \begin{cases} 1 & \text{se } \varphi_x \text{ è totale} \\ 0 & \text{se } \varphi_x \text{ non è totale} \end{cases}$$

## 7 Insiemi ricorsivi e ricorsivamente enumerabili

### 7.1 Insiemi ricorsivi e ricorsivamente enumerabili

Un insieme  $I \subseteq \mathbb{N}$  è detto **ricorsivamente enumerabile** (r.e.) se esiste una funzione ricorsiva (parziale o totale)  $\varphi$  tale che  $I = \text{dom}(\varphi)$ .

Inoltre, nel seguito indicheremo con:

- $W_x$  l'insieme r.e. associato al dominio della funzione  $\varphi_x$ :

$$W_x = \text{dom}(\varphi_x) = \{y \mid \varphi_x(y) \downarrow\}$$

essendo  $\varphi_x$  l' $x$ -esima MdT nella numerazione fissata.

- $\text{range}_x$  l'insieme  $\text{range}_x = \{y \mid \exists z \varphi_x(z) \downarrow \wedge \varphi_x(z) = y\}$  ovvero il codominio della funzione parziale ricorsiva  $\varphi_x$ .

Sia  $A \in RE$ . La funzione parziale:

$$\varphi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ \uparrow & \text{se } x \notin A \end{cases}$$

è detta **funzione semicaratteristica** dell'insieme  $A$ . È chiaro che  $A = \text{dom}(\varphi_A)$ .

Un insieme  $I \subseteq \mathbb{N}$  è detto **ricorsivo** (REC) se esiste una funzione ricorsiva (totale)  $f_1$  tale che:

$$f_1(x) = \begin{cases} 1 & \text{se } x \in I \\ 0 & \text{se } x \notin I \end{cases}$$

I seguenti esempi sono insiemi ricorsivi:

- L'insieme  $\{0, 2, 4, 6, \dots\}$  dei numeri pari;
- $\mathbb{N}$  e  $\emptyset$ ;
- Ogni insieme finito;
- Ogni insieme  $A$  tale che  $\overline{A}$  è finito.

Dato  $A$  insieme ricorsivo, si definisce la funzione parziale e chiaramente ricorsiva:

$$\varphi(x) = \begin{cases} 1 & \text{se } f_A(x) = 1 \\ \uparrow & \text{altrimenti} \end{cases}$$

Se un **insieme è ricorsivo allora è anche ricorsivamente enumerabile**:

$$A \in \text{REC} \implies A \in \text{r.e.}$$

**Teorema 17 (Teorema di Post).** *Un insieme  $A$  è ricorsivo se e solo se  $A$  e  $\overline{A}$  sono r.e.*

$$A \in \text{REC} \iff A, \overline{A} \in \text{r.e.}$$

Si definisce **la funzione**  $K = \{x \mid \varphi_x(x) \downarrow\}$ . È chiaro per la definizione che  $K = \{x \mid x \in W_x\}$ .

**Teorema 18.**  $K$  è r.e. ma non ricorsivo.

Esiste una **caratterizzazione degli insiemi** r.e. per cui le seguenti affermazioni sono equivalenti:

- $A \in \text{RE}$ ;
- $A = \text{range}(\varphi_p)$  con  $p$  parziale ricorsiva;
- $A = \emptyset$  oppure  $A = \text{range}(\varphi_p)$  con  $p$  totale ricorsiva.

**Teorema 19.** Si afferma che:  $A \in \text{REC} \iff A = \emptyset$  oppure  $A = \text{range}(f_P)$  con  $f_P$  totale ricorsiva non decrescente.

**Teorema 20.**  $A \in \text{REC}, |A| = W \iff A = \text{range}(f)$  con  $f$ , allora è **totale ricorsiva crescente**.

**Teorema 21.**  $A$  è r.e. e  $|A| = W \implies \exists B \in \text{REC}$  tale che  $|B| = W$  e  $B \leq A$ .

## 8 Teoremi di Ricorsione e Teorema di Rice

### 8.1 Primo teorema di ricorsione

**Teorema 22 (I Teorema di ricorsione di S.C. Kleene).** *Sia  $t$  una funzione ricorsiva totale. Allora esiste  $n \in \mathbb{N}$  tale che  $\varphi_n = \varphi_{t(n)}$ , dove la  $t$  rappresenta la trasformata.*

Per esempio un compilatore.

---

### 8.2 Secondo teorema di ricorsione

**Teorema 23 (II Teorema di ricorsione di S.C. Kleene).** *Se la funzione  $f$  è ricorsiva totale, allora esiste una funzione  $v : \mathbb{N} \rightarrow \mathbb{N}$  ricorsiva totale tale che:  $\forall y \in \mathbb{N} : \varphi_{f(v(y), y)} = \varphi_{v(y)}$ .*

---

### 8.3 Teorema di Rice

Il **Teorema di Rice** afferma che ogni proprietà non banale che rappresenti ciò che viene calcolato dalla MdT non è ricorsiva. Si definisce il termine *proprietà*: una **proprietà** sulle MdT è un qualsiasi sottoinsieme di  $\{\varphi_i\}_{i \in \mathbb{N}}$ . Dove quest'ultimo termine indica una enumerazione delle MdT.

Sia  $\Pi$  una proprietà sulle MdT,  $\Pi \subseteq \mathbb{N}$ . Allora  $\Pi$  è **estensionale** se per ogni  $x, y \in \mathbb{N}$  è vero:

$$(x \in \Pi \wedge \varphi_x = \varphi_y) \implies y \in \Pi$$

In altre parole, una proprietà è estensionale quando questa “parla” di cosa calcolano le MdT in essa contenute, e non di come queste macchine sono “fatte”.

La **tecnica del Padding** afferma che:

$$\forall i, k. \exists j : (\varphi_j = \varphi_i \wedge j > k)$$

**Teorema 24 (Teorema di Rice).** *Sia  $\Pi$  una proprietà estensionale.  $\Pi$  è ricorsiva se e soltanto se è banale, ovvero  $\Pi = \emptyset$  oppure  $\Pi = \mathbb{N}$ .*

Il teorema di Rice dunque afferma che ogni funzione parziale ricorsiva o MdT ammette un insieme infinito non ricorsivo di indici di funzioni o macchine equivalenti.